RouterOS

RouterOS Documentation

This document describes RouterOS, the operating system of MikroTik devices. Documentation applies for the latest stable RouterOS version.

Also available in the documentation in PDF format for offline use (updated rarely).

SwOS Documentation

For RB260, CSS326, CRS3xx, CSS610 and GPEN21 devices running SwOS, see the SwOS user manual.

MikroTik Newsletter

To follow the latest product and software news, make sure to read our newsletters in the blog section.

- **Newsletter section**
- **PDF archive of all newsletters**

Recently Updated

Neighbor discovery
5 minutes ago • updated by Edgars P. • view change
Quick Guide - L009UiGS-2HaxD-IN
about an hour ago • updated by Serhii T. • view change
Virtual Routing and Forwarding (VRF)
about 2 hours ago • updated by Olga Ļ. • view change
Quick Guide - hAP ax²
about 2 hours ago • updated by Serhii T. • view change
Quick Guide - CRS312-4C+8XG-RM
about 5 hours ago • updated by Serhii T. • view change
Quick Guide - ATL LTE18 kit
about 5 hours ago • updated by Serhii T. • view change
MikroTik wired interface compatibility
about 6 hours ago • updated by Rihards Vārna • view change
Quick Guide - CRS518-16XS-2XQ-RM
about 7 hours ago • updated by Serhii T. • view change
image-2022-12-9_14-4-59.png
about 7 hours ago • attached by Serhii T.
PoE-Out
about 7 hours ago • updated by Edgars P. • view change
L009UiGS-2HaxD-IN
yesterday at 3:11 PM • created by Serhii T.
CCR2004-16G-2S+
yesterday at 1:55 PM • updated by Serhii T. • view change
Quick Guide - 5GHz RouterBOARD ax series
yesterday at 10:11 AM • updated by Certification Department • view change
Safety and Regulatory information - NetBox 5 ax Lite

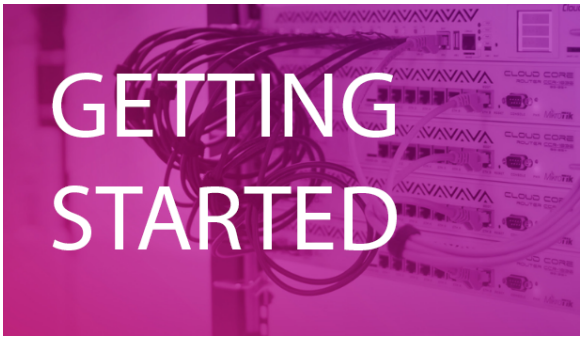
yesterday at 9:08 AM • updated by Certification Department • view change

Netinstall

yesterday at 9:02 AM • updated by Serhii T. • view change

Getting started

In This Section:



RouterOS is a stand-alone operating system based on Linux kernel. It powers MikroTik hardware devices, but is also available for virtual machines. If you are reading this document and have no prior experience with RouterOS, please use the menu on the left hand side, to learn about first steps with RouterOS. The different methods of connecting to your device are discussed under the "management tools" section.

Software Specifications

Hardware Support

RouterOS is fully compatible with MikroTik hardware it comes preinstalled on. It can also be run on 3rd party devices if they meet the following requirements

- i386 compatible architecture
- SMP – multi-core and multi-CPU compatible
- Minimum 32MB of RAM, since RouterOS v7 there is no more maximum RAM.
- IDE, SATA, USB, and flash storage medium with a minimum of 64MB of space
- Network cards supported by Linux kernel (PCI, PCI-X)
- Switch chip configuration support

Note: NVMe storage is supported only for CHR, x86, Tile, and MMIPS architecture. For specific information, please look at each product brochure o

Note: We do not recommend running v7 on hardware that does not have at least 64 MB of RAM.

Installation

- Netinstall: Full network-based installation from PXE or EtherBoot enabled network card
- CHR: RouterOS version intended for running as a virtual machine
- CD-based installation

Configuration

- MAC-based access for initial configuration
- WinBox – standalone Windows GUI configuration tool
- Webfig - advanced web-based configuration interface
- MikroTik - Android and iOS-based configuration tool
- Powerful command-line configuration interface with integrated scripting capabilities, accessible via local terminal, serial console, telnet and ssh
- API - the way to create your own configuration and monitoring applications

Backup/Restore

- Binary configuration backup saving and loading
- Configuration export and import in human-readable text format

Firewall

- Stateful filtering
- Source and destination NAT
- NAT helpers (h323, pptp, quake3, sip, ftp, irc, tftp)
- Internal connection, routing and packet marks
- Filtering by IP address and address range, port and port range, IP protocol, DSCP, and many more
- Address lists
- Custom Layer7 matcher
- IPv6 support
- PCC - per connection classifier, used in load balancing configurations
- RAW filtering to bypass connection tracking.

Routing

- Static routing
- Virtual Routing and Forwarding (VRF)
- Policy-based routing
- Interface routing
- ECMP routing
- IPv4 dynamic routing protocols: RIP v1/v2, OSPFv2, BGP v4
- IPv6 dynamic routing protocols: RIPng, OSPFv3, BGP
- Bidirectional Forwarding Detection (BFD)

MPLS

- Static Label bindings for IPv4
- Label Distribution protocol for IPv4
- RSVP Traffic Engineering tunnels
- VPLS MP-BGP based autodiscovery and signaling
- MP-BGP based MPLS IP VPN

VPN

- IPSec – tunnel and transport mode, certificate or PSK, AH, and ESP security protocols.
- IKEv2 support
- AES-NI hardware acceleration support for IPSec
- Point-to-point tunneling (OpenVPN, PPTP, PPPoE, L2TP, SSTP)
- Advanced PPP features (MLPPP, BCP)
- Simple tunnels (IPIP, EoIP) IPv4 andIPv6 support
- 6to4 tunnel support (IPv6 over IPv4 network)
- VLAN – IEEE802.1q Virtual LAN support, Q-in-Q support
- MPLS based VPNs
- WireGuard
- ZeroTier

Wireless

- IEEE802.11a/b/g wireless client and access point
- Full IEEE802.11n support
- Nstreme and Nstreme2 proprietary protocols
- NV2 protocol
- Wireless Distribution System (WDS)
- Virtual AP
- WEP, WPA, WPA2
- Access control list
- Wireless client roaming
- WMM
- HWMP+ Wireless MESH protocol
- MME wireless routing protocol

DHCP

- Per interface DHCP server
- DHCP client and relay
- Static and dynamic DHCP leases

- RADIUS support
- Custom DHCP options
- DHCPv6 Prefix Delegation (DHCPv6-PD)
- DHCPv6 Client

Hotspot

- Plug-n-Play access to the Network
- Authentication of local Network Clients
- Users Accounting
- RADIUS support for Authentication and Accounting

QoS

- Hierarchical Token Bucket (HTB) QoS system with CIR, MIR, burst and priority support
- Simple and fast solution for basic QoS implementation - Simple queues
- Dynamic client rate equalization (PCQ)

Proxy

- HTTP caching proxy server
- Transparent HTTP proxy
- SOCKS protocol support
- DNS static entries
- Support for caching on a separate drive
- Parent proxy support
- Access control list
- Caching list

Tools

- Ping, traceroute
- Bandwidth test, ping flood
- Packet sniffer, torch
- Telnet, ssh
- E-mail and SMS send tools
- Automated script execution tools
- CALEA
- File Fetch tool
- Advanced traffic generator
- WoL (Wake on LAN) sending

Other features

- Samba support
- OpenFlow support
- Bridging – spanning tree protocol (STP, RSTP), bridge firewall and MAC natting.
- Dynamic DNS update tool
- NTP client/server and synchronization with GPS
- VRRP v2 and v3 support
- SNMP
- M3P - MikroTik Packet packer protocol for wireless links and ethernet
- MNDP - MikroTik neighbor discovery protocol, supports CDP (Cisco discovery protocol)
- RADIUS authentication and accounting
- TFTP server
- Synchronous interface support (Farsync cards only) (Removed in v5.x)
- Asynchronous – serial PPP dial-in/dial-out, dial-on-demand
- ISDN – dial-in/dial-out, 128K bundle support, Cisco HDLC, x75i, x75ui, x75bui line protocols, dial-on-demand

Kernel version

- RouterOS version 6.x uses 3.3.5
- RouterOS version 7.x uses 5.6.3

Supported Encryptions

RouterOS 7 is used for the management of network (telecommunication) devices.

- RouterOS 7 includes encryption features (components), intended for data (information) security, passed through telecommunication channels and c
- All encryption features (components) are an integral part of RouterOS 7 and can not be changed by the end users.
- RouterOS 7 is intended for installation by end-users without significant support from the vendor.
- RouterOS 7 uses the following security protocols:

Supported security protocol	Encryption algorithm	Maximum key length
IPSec	DES	56 bit
	3DES	168 bit
	AES	128, 192, 256 bit
	Blowfish	448 bit
	Twofish	256 bit
	Camelia	128, 192, 256 bit
PPTP (with MPPE)	RC4	128 bit
L2TP (with MPPE)	RC4	128 bit
SNMP	DES	56 bit
	AES	128 bit
SSH	Blowfish	128 bit
	3DES	192 bit
	AES	128, 192, 256 bit
SSTP	AES	256 bit
	RC4	128 bit
Used in WinBox connection (nameless)	AES	128 bit
WEP	RC4	104 bit
WPA-TKIP	RC4	128 bit
WPA2-TKIP	RC4	128 bit
WPA-AES	AES	128 bit
WPA2-AES	AES	128 bit
HTTPS	NULL, RC4, DES, DES40, 3DES, AES	128, 192, 256 bit

Feature support based on architecture

All devices support the same features, with a few exceptions, clarified in the below table:

Architecture	Not supported	Exclusively supported
ARM		Zerotier, Container
ARM64		Zerotier, Container
MIPSBE	Zerotier, Dude server	
MMIPS	Zerotier	
SMIPS	Zerotier, DOT1X, BGP, MPLS, PIMSM, Dude server, User manager	
TILE	Zerotier	
PPC	Zerotier, Dude server	
X86 PC	Zerotier, Cloud	Container
CHR VM		

Apart from features, there are also a few differences in hardware capabilities, based on the specific model of device. For these differences, please see the below articles:

- Wifi - new driver implementation for 802.11ax devices and supported older devices https://help.mikrotik.com/docs/display/ROS/Wifi
- L3 Hardware offloading https://help.mikrotik.com/docs/display/ROS/L3+Hardware+Offloading#L3HardwareOffloading-L3HWDeviceSupport
- PTP https://help.mikrotik.com/docs/display/ROS/Precision+Time+Protocol
- Switch chip features https://help.mikrotik.com/docs/display/ROS/Switch+Chip+Features

First Time Configuration

Connecting to the Router

There are two types of routers:

- With default configuration
- Without default configuration. When no specific configuration is found, the IP address 192.168.88.1/24 is set on ether1 or combo1, or sfp1.

More information about the current default configuration can be found in the Quick Guide document that came with your device. The quick guide document will include information about which ports should be used to connect for the first time and how to plug in your devices.

This document describes how to set up the device from the ground up, so we will ask you to clear away all defaults.

When connecting the first time to the router with the default username **admin** and **no password** (for some models, check the user password on the sticker), you will be asked to reset or keep the default configuration (even if the default config has only an IP address). Since this article assumes that there is no configuration on the router you should remove it by pressing "r" on the keyboard when prompted or click on the "Remove configuration" button in WinBox.

Router without Default Configuration

If there is no default configuration on the router you have several options, but here we will use one method that suits our needs.

Connect the Routers ether1 port to the WAN cable and connect your PC to ether2. Now open WinBox and look for your router in Neighbor Discovery. See detailed example in Winbox article.

If you see the router in the list, click on the MAC address and click **Connect**.

The simplest way to make sure you have clean router, with no configuration, is to run:

```
/system reset-configuration no-defaults=yes skip-backup=yes
```

Or from WinBox (Fig. 1-1):

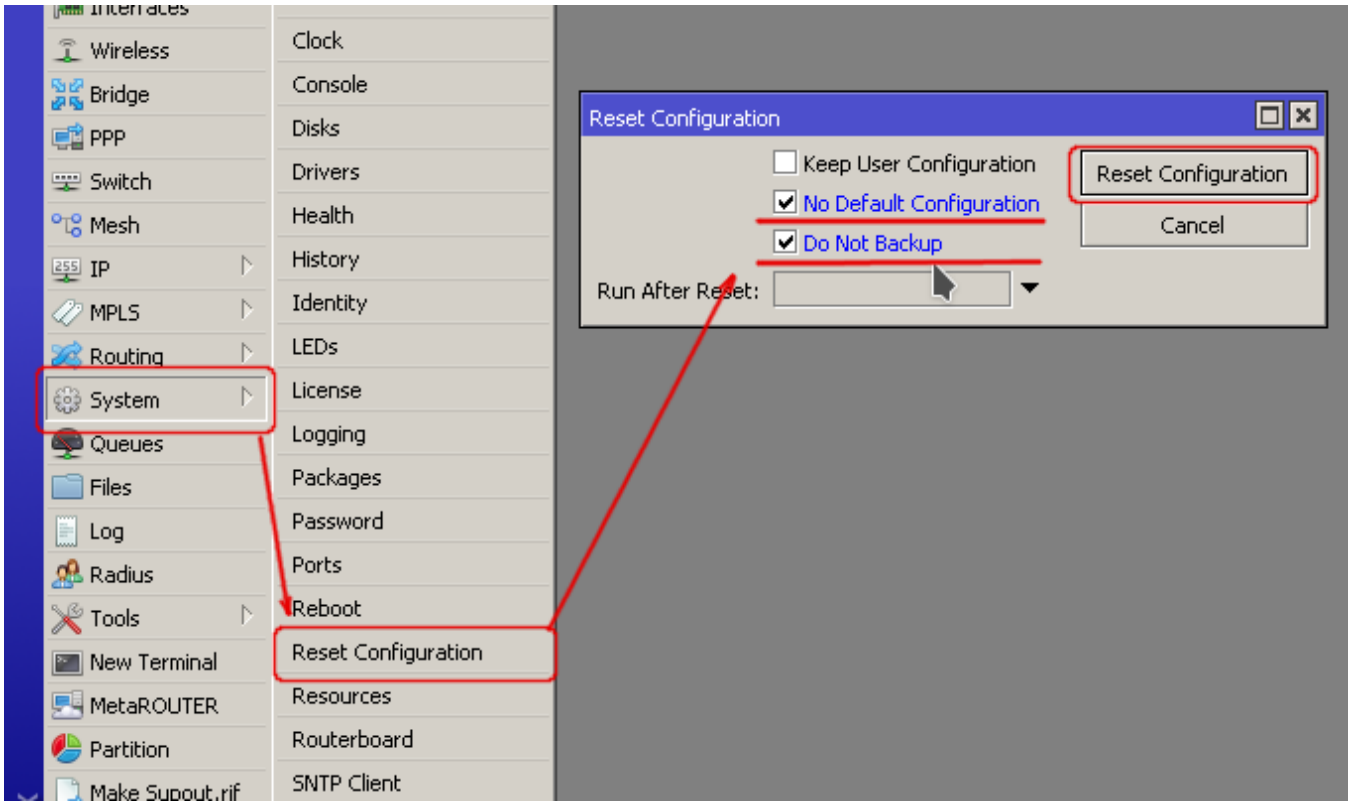


Fig. 1-1

Configuring IP Access

Since the MAC connection is not very stable, the first thing we need to do is to set up a router so that IP connectivity is available:

- add bridge interface and bridge ports;
- add an IP address to the LAN interface;
- set up a DHCP server.

Set bridge and IP address are quite easy:

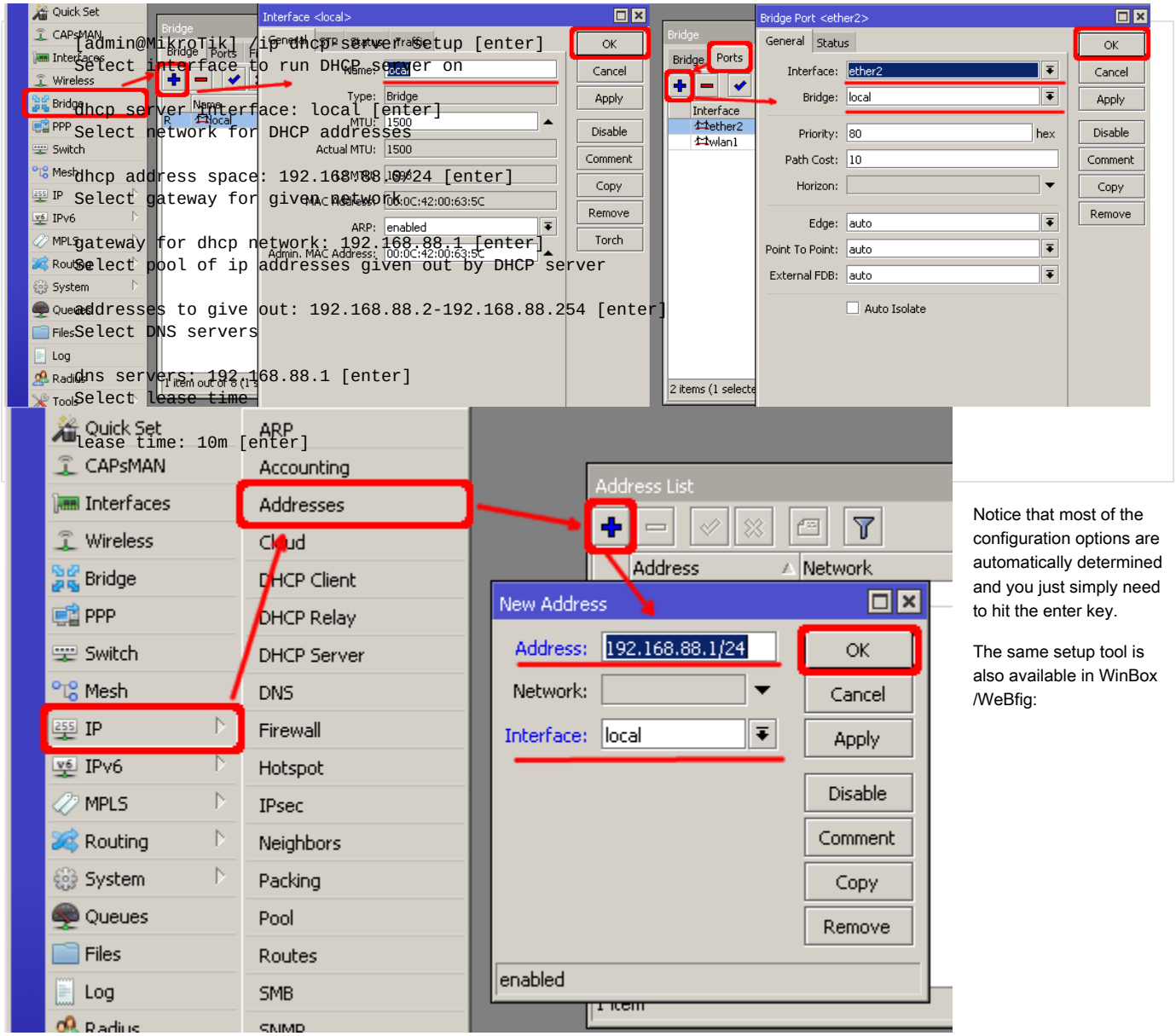
```
/interface bridge add name=local
/interface bridge port add interface=ether2 bridge=local
/ip address add address=192.168.88.1/24 interface=local
```

If you prefer WinBox/WeBfig as configuration tools:

- Open the **Bridge** window, **Bridge** tab should be selected;
- Click on the **+** button, a new dialog will open, enter bridge name **local** and click on **OK**;
- Select the Ports tab and click on the **+** button, a new dialog will open;
- select interface **ether2** and bridge **local** form drop-down lists and click on the **OK** button to apply settings;
- You may close the bridge dialog.

- Open **Ip -> Addresses** dialog;
- Click on the **+** button, a new dialog will open;
- Enter IP address **192.168.88.1/24** select interface **local** from the drop-down list and click on **OK** button;

The next step is to set up a DHCP server. We will run the **setup** command for easy and fast configuration:



Notice that most of the configuration options are automatically determined and you just simply need to hit the enter key.

The same setup tool is also available in WinBox /WeBfig:

- Open **IP -> DHCP Server** window, **DHCP** tab should be selected;
- Click on the **DHCP Setup** button, a new dialog will open, enter DHCP Server Interface **local** and click on the **Next** button;
- Follow the wizard to complete the setup.

Now connected PC should be able to get a dynamic IP address. Close the Winbox and reconnect to the router using the IP address (192.168.88.1)

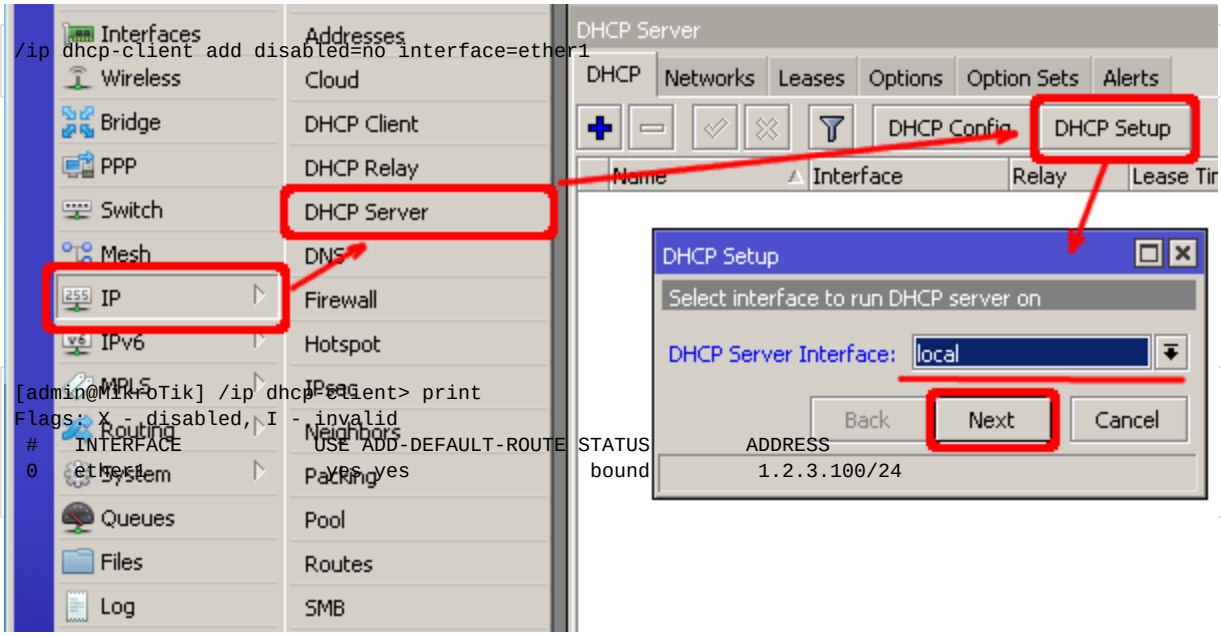
Configuring Internet Connection

The next step is to get internet access to the router. There can be several types of internet connections, but the most common ones are:

- dynamic public IP address;
- static public IP address;
- PPPoE connection.

Dynamic Public IP

Dynamic address configuration is the simplest one. You just need to set up a DHCP client on the public interface. DHCP client will receive information from an internet service provider (ISP) and set up an IP address, DNS, NTP servers, and default route for you.



After adding the client you should see the assigned address and the status should be bound

Static Public IP

In the case of static address configuration, your ISP gives you parameters, for example:

- IP: 1.2.3.100/24
- Gateway: 1.2.3.1
- DNS: 8.8.8.8

These are three basic parameters that you need to get the internet connection working

To set this in RouterOS we will manually add an IP address, add a default route with a provided gateway, and set up a DNS server

```
/ip address add address=1.2.3.100/24 interface=ether1
/ip route add gateway=1.2.3.1
/ip dns set servers=8.8.8.8
```

PPPoE Connection

PPPoE connection also gives you a dynamic IP address and can configure dynamically DNS and default gateway. Typically service provider (ISP) gives you a username and password for the connection

```
/interface pppoe-client
  add disabled=no interface=ether1 user=me password=123 add-default-route=yes use-peer-dns=yes
```

Winbox/Webfig actions:

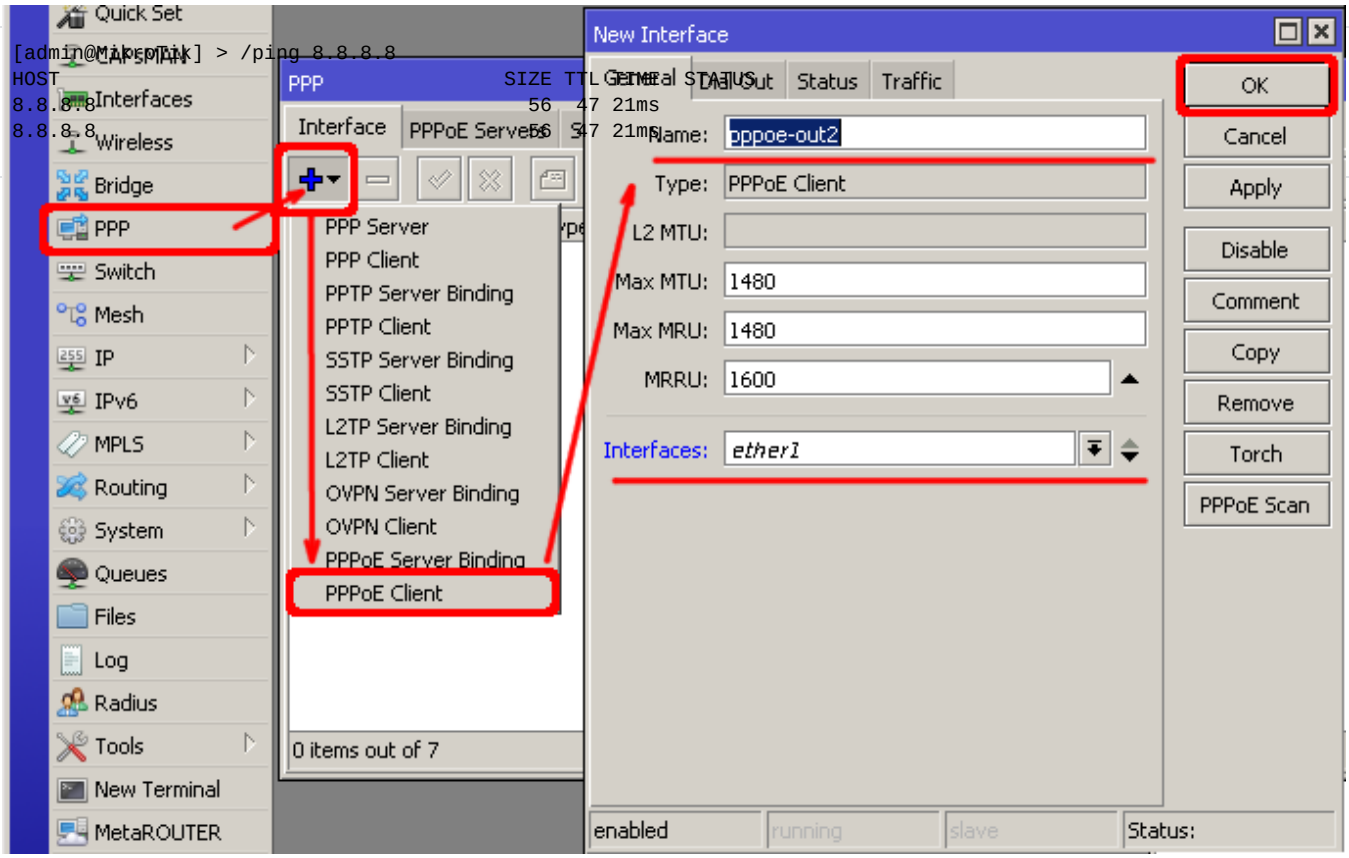
- Open **PPP** window, **Interfaces** tab should be selected;
- Click on the **+** button, and choose **PPPoE Client** from the dropdown list, a new dialog will open;
- Select interface **ether1** from the dropdown list and click on the **OK** button to apply settings.

⚠ Now in configuration **WAN** interface is **pppoe-out** interface, not **ether1**, make sure to adjust "/interface/list/member" to reflect these changes, if you are referencing interface lists in your configuration.

Verify Connectivity

After successful configuration, you should be able to access the internet from the router.

Verify IP connectivity by pinging a known IP address (google DNS server for example)



Verify DNS request

```
[admin@MikroTik] > /ping www.google.com
HOST                                     SIZE TTL TIME  STATUS
173.194.32.49                              56  55 13ms
173.194.32.49                              56  55 12ms
```

If everything is set up correctly, ping in both cases should not fail.

In case of failure refer to the Troubleshooting section

Protecting the Router

Now anyone over the world can access our router so it is the best time to protect it from intruders and basic attacks

User Password Access

MikroTik routers require password configuration, we suggest using a password generator tool to create secure and non-repeating passwords. By secure password, we mean:

- Minimum 12 characters;
- Include numbers, Symbols, Capital and lowercase letters;
- Is not a Dictionary Word or a combination of Dictionary Words;

```
/user set 0 password="!={Ba3N!40TX+GvKBzjTLIUcx/,"
```

Another option to set a password,

```
/password
```

We strongly suggest using a second method or Winbox interface to apply a new password for your router, just to keep it safe from other unauthorized access.

```
[admin@MikroTik] > / password
old password:
new password: ******
retype new password: ******
```

Make sure you remember the password! If you forget it, there is no recovery. You will need to reinstall the router!

You can also add more users with full or limited router access in **/user** menu

The best practice is to add a new user with a strong password and disable or remove the default **admin** user.

```
/user add name=myname password=mypassword group=full
/user remove admin
```

Note: login to the router with new credentials to check that the username/password is working.

MAC Connectivity Access

By default, MAC server runs on all interfaces, so we will disable the default "**all"** entry and add a local interface to disallow MAC connectivity from the WAN port. MAC Telnet Server feature allows you to apply restrictions to the interface "list".

First, create an interface list:

```
[admin@MikroTik] > /interface list add name=listBridge
```

Then, add your previously created bridge named "local" to the interface list:

```
[admin@MikroTik] > /interface list member add list=listBridge interface=local
```

Apply newly created "list" (of interfaces) to the MAC server:

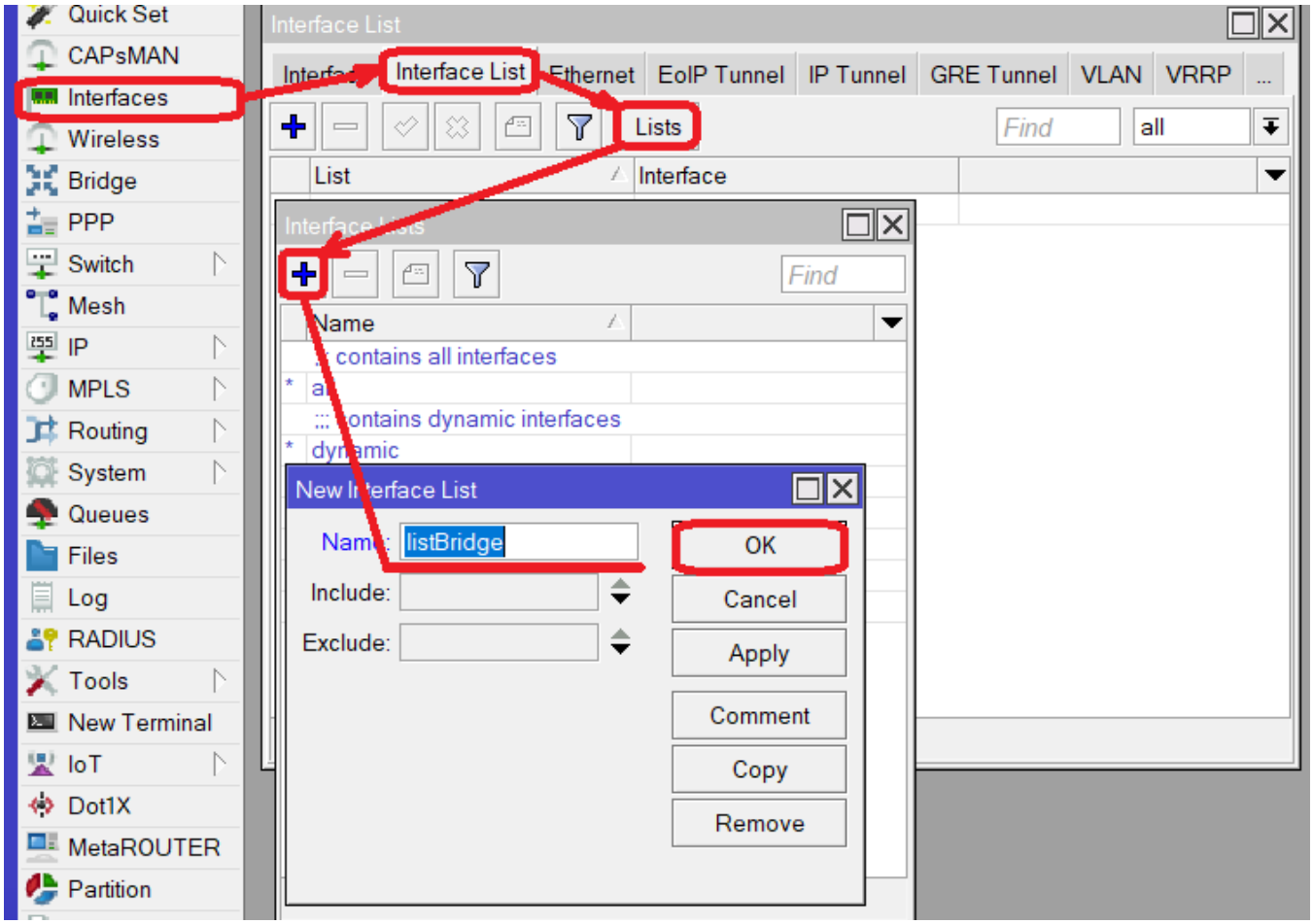
```
[admin@MikroTik] > tool mac-server set allowed-interface-list=listBridge
```

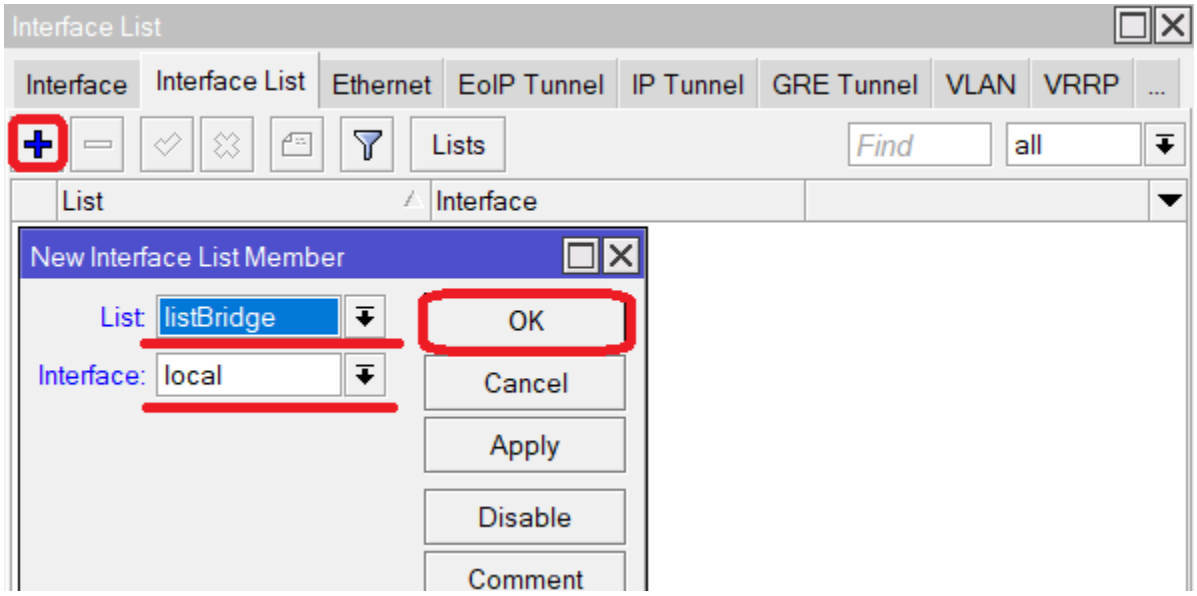
Do the same for Winbox MAC access

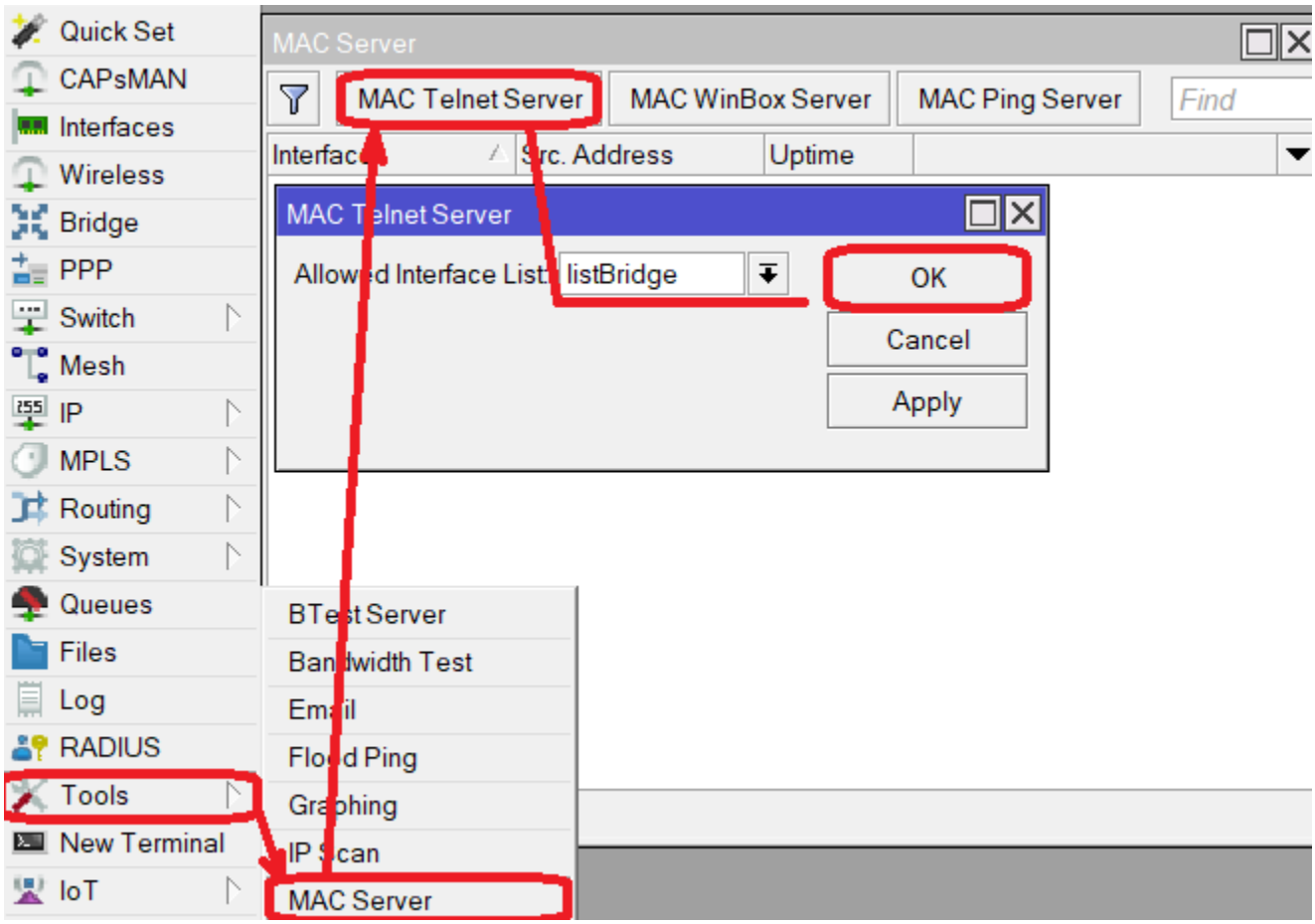
```
[admin@MikroTik] > tool mac-server mac-winbox set allowed-interface-list=listBridge
```

Winbox/Webfig actions:

- Open **Interfaces → Interface List → Lists** window and add a new list by clicking "+";
- Input the interface list name "listBridge" into the **Name** field and click **OK**;
- Go back to the **Interfaces → Interface List** section and click "+";
- Select "listBridge" from the dropdown **List** options and select "local" from the dropdown **Interface** options and click **OK**;
- Open **Tools -> Mac Server** window;
- Click on the **"MAC Telnet Server"** button, a new dialog will open;
- Select the newly created list "listBridge" from the dropdown list and click on the **OK** button to apply settings.


Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Tools
New Terminal
IoT
Dot1X
MetaROUTER
Partition
Interface List
Interface List
Ethernet
EoIP Tunnel
IP Tunnel
GRE Tunnel
VLAN
VRRP
Lists
Find
all
List
Interface
Name
contains all interfaces
contains dynamic interfaces
dynamic
New Interface List
Name:
listBridge
Include:
Exclude:
OK
Cancel
Apply
Comment
Copy
Remove


Interface List
Interface
Interface List
Ethernet
EoIP Tunnel
IP Tunnel
GRE Tunnel
VLAN
VRRP
Lists
Find
all
List
Interface
New Interface List Member
List:
listBridge
Interface:
local
OK
Cancel
Apply
Disable
Comment



Do the same in the **MAC Winbox Server** tab to block Mac Winbox connections from the internet.

Neighbor Discovery

MikroTik Neighbor discovery protocol is used to show and recognize other MikroTik routers in the network. Disable neighbor discovery on public interfaces:

```
/ip neighbor discovery-settings set discover-interface-list=listBridge
```

IP Connectivity Access

Besides the fact that the firewall protects your router from unauthorized access from outer networks, it is possible to restrict username access for the specific IP address

```
/user set 0 allowed-address=x.x.x.x/yy
```

x.x.x.x/yy - your IP or network subnet that is allowed to access your router.

IP connectivity on the public interface must be limited in the firewall. We will accept only ICMP(ping/traceroute), IP Winbox, and SSH access.

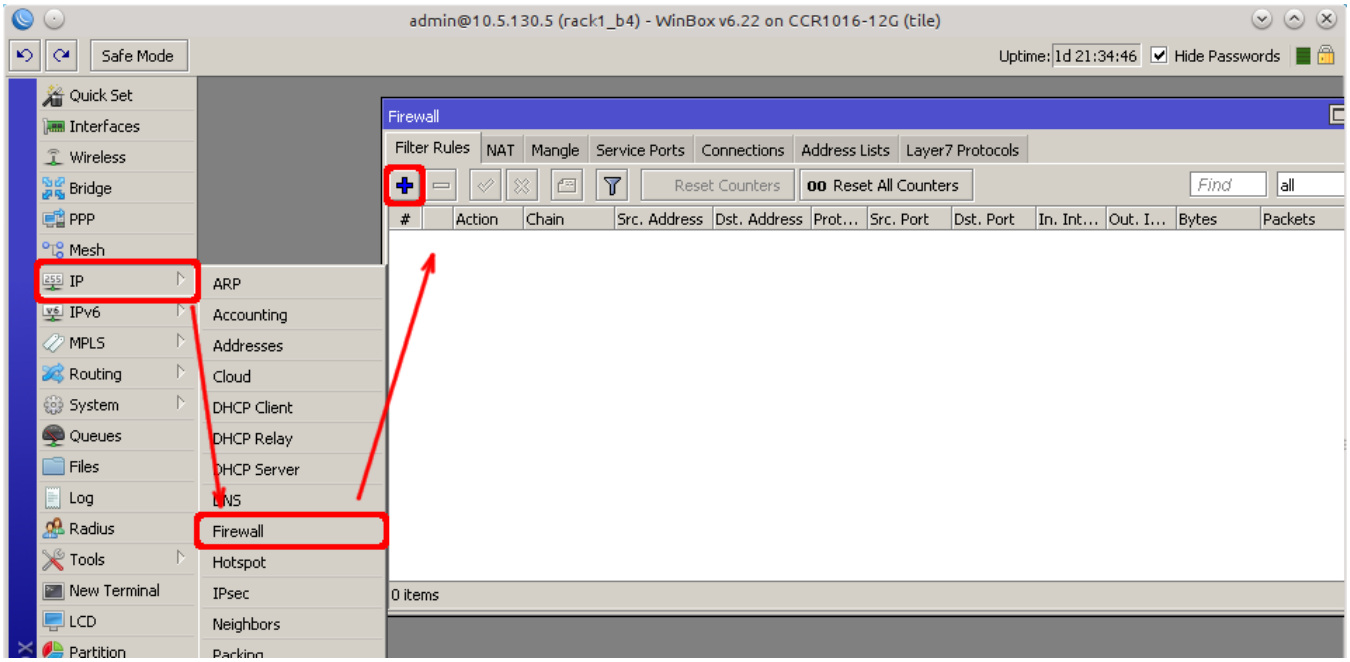
```
/ip firewall filter
  add chain=input connection-state=established,related action=accept comment="accept established,related";
  add chain=input connection-state=invalid action=drop;
  add chain=input in-interface=ether1 protocol=icmp action=accept comment="allow ICMP";
  add chain=input in-interface=ether1 protocol=tcp port=8291 action=accept comment="allow Winbox";
  add chain=input in-interface=ether1 protocol=tcp port=22 action=accept comment="allow SSH";
  add chain=input in-interface=ether1 action=drop comment="block everything else";
```

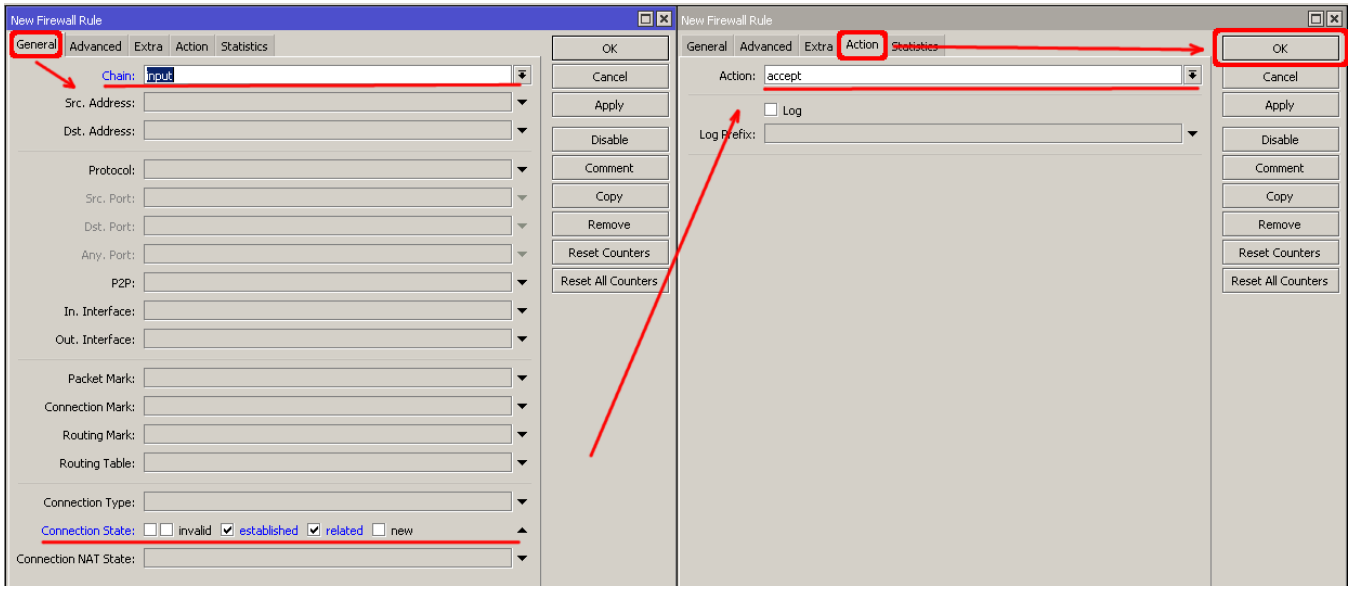
In case the public interface is PPPoE, then the in-interface should be set to "pppoe-out".

The first two rules accept packets from already established connections, so we assume those are OK to not overload the CPU. The third rule drops any packet that connection tracking thinks is invalid. After that, we set up typical accept rules for specific protocols.

If you are using Winbox/Webfig for configuration, here is an example of how to add an established/related rule:

- Open **IP -> Firewall** window, click on the **Filter rules** tab;
- Click on the **+** button, a new dialog will open;
- Select chain input, click on **Connection state,** and select checkboxes for established and related;
- Click on the **Action** tab and make sure action accept is selected;
- Click on the **OK** button to apply settings.





To add other rules click on **+** for each new rule and fill the same parameters as provided in the console example.

Administrative Services

Although the firewall protects the router from the public interface, you may still want to disable RouterOS services.

Most of RouterOS administrative tools are configured at the /ip service menu

Keep only secure ones,

```
/ip service disable telnet,ftp,www,api
```

Change default service ports, this will immediately stop most of the random SSH brute force login attempts:

```
/ip service set ssh port=2200
```

Additionally, each service can be secured by the allowed IP address or address range(the address service will reply to), although the preferred method is to block unwanted access in the firewall filter,because the firewall will not even allow to open socket

```
/ip service set winbox address=192.168.88.0/24
```

Other Services

A bandwidth server is used to test throughput between two MikroTik routers. Disable it in the production environment.

```
/tool bandwidth-server set enabled=no
```

A router might have DNS cache enabled, which decreases the resolving time for DNS requests from clients to remote servers. In case DNS cache is not required on your router or another router is used for such purposes, disable it.

```
/ip dns set allow-remote-requests=no
```

Some RouterBOARDs have an LCD module for informational purposes, set a pin or disable it.

```
/lcd set enabled=no
```

It is good practice to disable all unused interfaces on your router, to decrease the chances of unauthorized access to your router.

```
/interface print
/interface set x disabled=yes
```

Where "X" is the number of the unused interfaces.

You can enable stronger crypto for SSH, most newer programs use it, to turn on SSH strong crypto:

```
/ip ssh set strong-crypto=yes
```

The following services are disabled by default, nevertheless, it is better to make sure that none of them were enabled accidentally:

- MikroTik caching proxy,

```
/ip proxy set enabled=no
```

- MikroTik socks proxy,

```
/ip socks set enabled=no
```

- MikroTik UPNP service,

```
/ip upnp set enabled=no
```

- MikroTik dynamic name service or IP cloud,

```
/ip cloud set ddns-enabled=no update-time=no
```

NAT Configuration

At this point, the PC is not yet able to access the Internet, because locally used addresses are not routable over the Internet. Remote hosts simply do not know how to correctly reply to your local address.

The solution for this problem is to change the source address for outgoing packets to the router's public IP. This can be done with the NAT rule:

```
/ip firewall nat
  add chain=srcnat out-interface=ether1 action=masquerade
```

In case if a public interface is a pppoe, then the out-interface should be set to "pppoe-out".

Another benefit of such a setup is that NATed clients behind the router are not directly connected to the Internet, that way additional protection against attacks from outside mostly is not required.

Port Forwarding

Some client devices may need direct access to the internet over specific ports. For example, a client with an IP address 192.168.88.254 must be accessible by Remote desktop protocol (RDP).

After a quick search on Google, we can find out that RDP runs on TCP port 3389. Now we can add a destination NAT rule to redirect RDP to the client's PC.

```
/ip firewall nat
  add chain=dstnat protocol=tcp port=3389 in-interface=ether1 action=dst-nat to-address=192.168.88.254
```

If you have set up strict firewall rules then RDP protocol must be allowed in the firewall filter forward chain.

Setting up Wireless

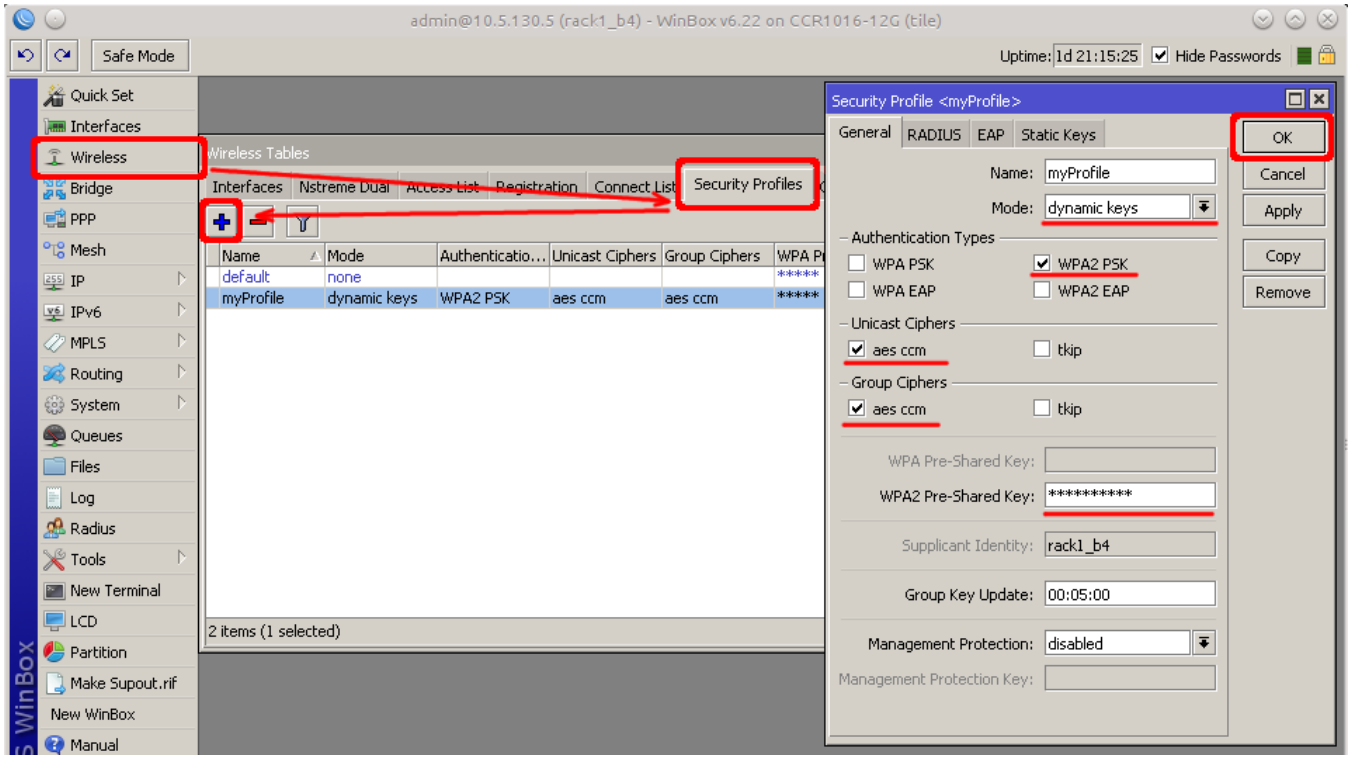
For ease of use bridged wireless setup will be made so that your wired hosts are in the same Ethernet broadcast domain as wireless clients.

The important part is to make sure that our wireless is protected, so the first step is the security profile.

Security profiles are configured from `/interface wireless security-profiles` menu in a terminal.

```
/interface wireless security-profiles
  add name=myProfile authentication-types=wpa2-psk mode=dynamic-keys wpa2-pre-shared-key=1234567890
```

in Winbox/Webfig click on **Wireless** to open wireless windows and choose the **Security Profile** tab.



If there are legacy devices that do not support WPA2 (like Windows XP), you may also want to allow WPA protocol.

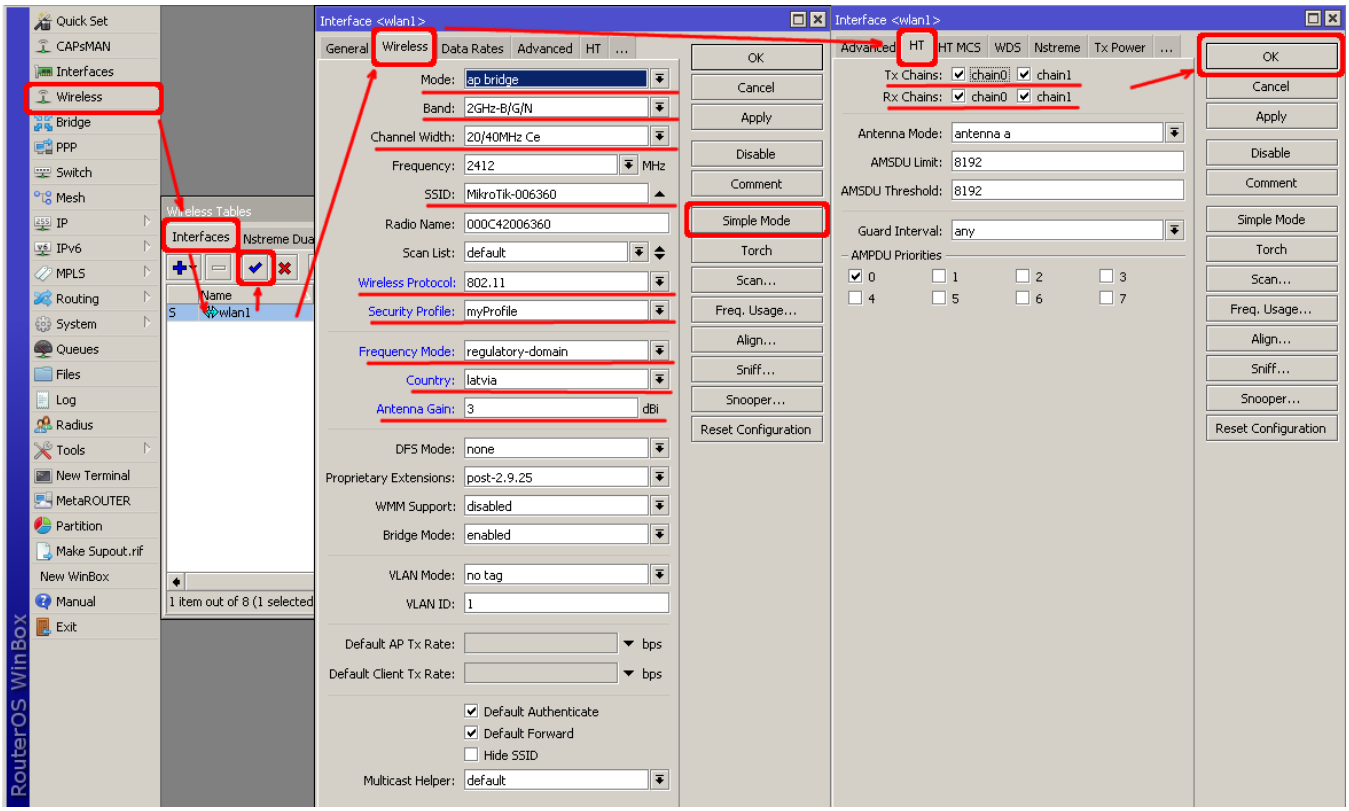
WPA and WPA2 pre-shared keys should not be the same.

Now when the security profile is ready we can enable the wireless interface and set the desired parameters

```
/interface wireless
  enable wlan1;
  set wlan1 band=2ghz-b/g/n channel-width=20/40mhz-Ce distance=indoors mode=ap-bridge ssid=MikroTik-006360
wireless-protocol=802.11 security-profile=myProfile frequency-mode=regulatory-domain set country=latvia
```

To do the same from Winbox/Webfig:

- Open the Wireless window, select wlan1 interface, and click on the *enable* button;
- Double-click on the wireless interface to open the configuration dialog;
- In the configuration dialog click on the **Wireless** tab and click the **Advanced mode** button on the right side. When you click on the button additional configuration parameters will appear and the description of the button will change to **Simple mode**;
- Choose parameters as shown in the screenshot, except for the country settings and SSID. You may want to also choose a different frequency and antenna gain;
- Next, click on the **HT** tab and make sure both chains are selected;
- Click on the **OK** button to apply settings.



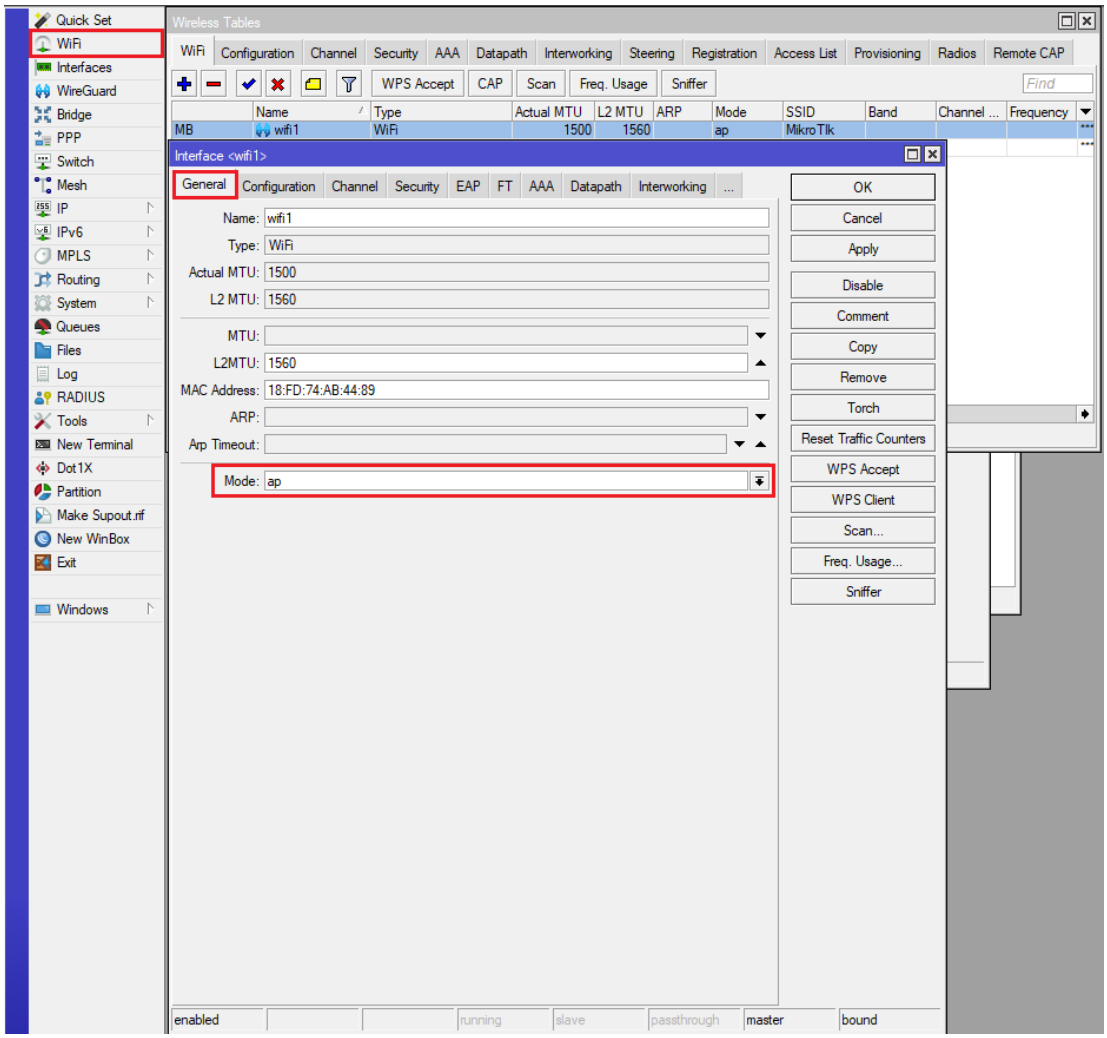
The last step is to add a wireless interface to a local bridge, otherwise, connected clients will not get an IP address:

```
/interface bridge port
  add interface=wlan1 bridge=local
```

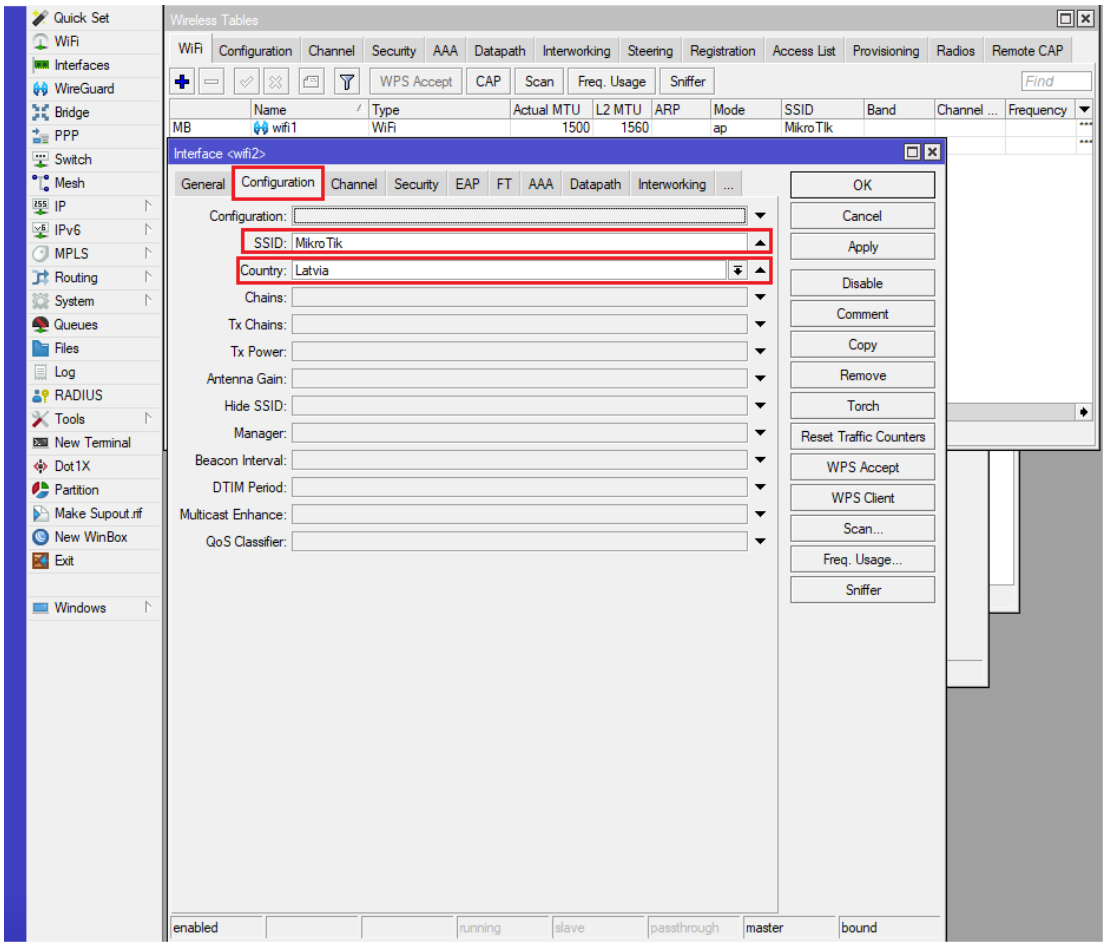
Now wireless should be able to connect to your access point, get an IP address, and access the internet.

RouterOS 7.13+ wireless setup

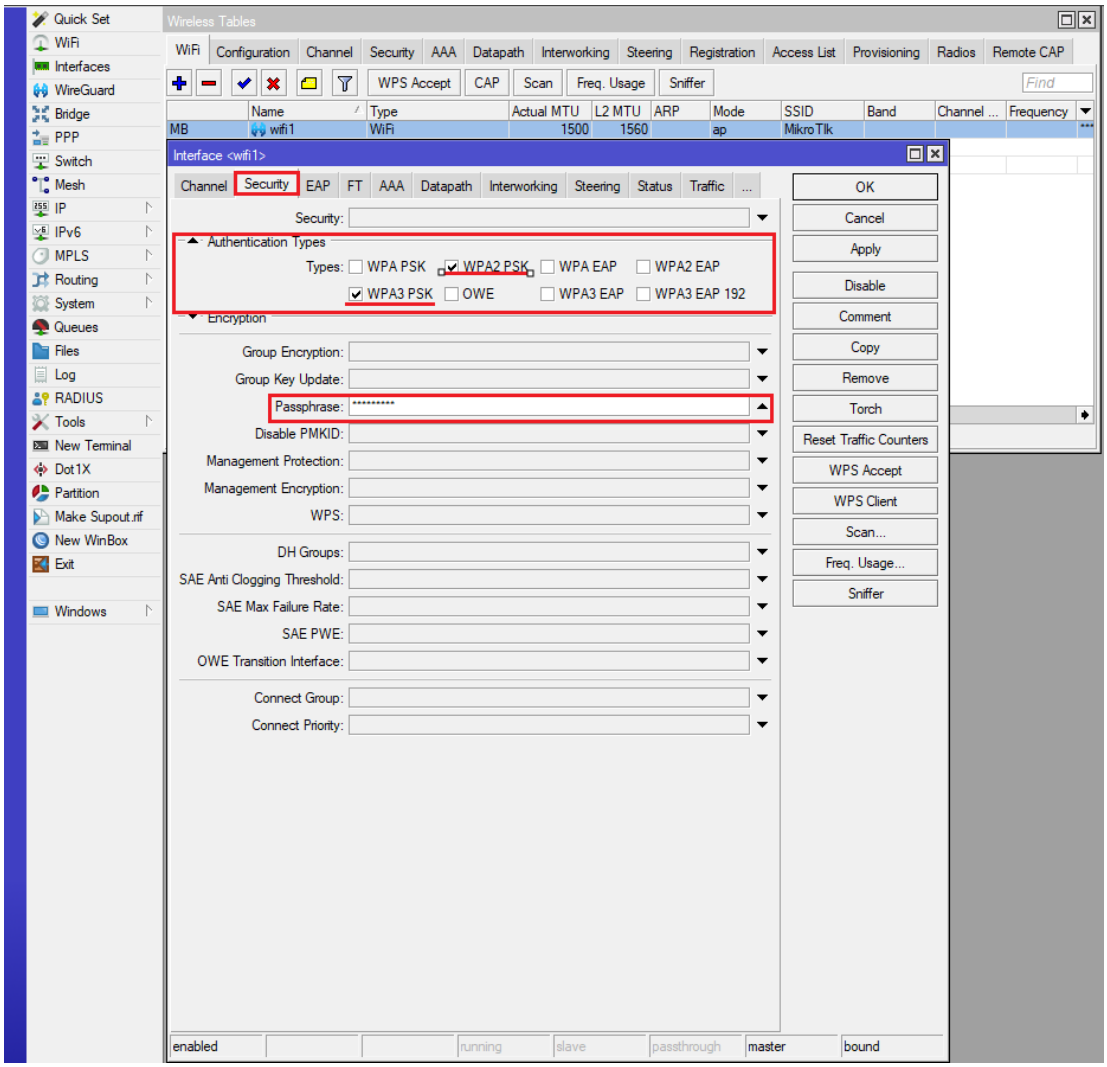
First enable the device as an Access Point, by navigating to the **General** tab and setting **Mode** to **AP**:



In the "Configuration tab" set the **Country** and **SSID**



Configure the wireless network`s security, we recommend using both WPA3 and WPA2 protocols, to ensure compatibility with older wireless clients, **authen tication-types** and **passphrase** should be set here:




```
interface/wifi/set wifi1 configuration.country=Latvia .mode=ap .ssid=MikroTik security.authentication-
types=wpa2-psk,wpa3-psk .passphrase=123456789
```

WPA2 is optional, some older devices may not support WPA3.

Protecting the Clients

Now it is time to add some protection for clients on our LAN. We will start with a basic set of rules.

```
/ip firewall filter
  add chain=forward action=fasttrack-connection connection-state=established,related comment="fast-track for
established,related";
  add chain=forward action=accept connection-state=established,related comment="accept established,related";
  add chain=forward action=drop connection-state=invalid
  add chain=forward action=drop connection-state=new connection-nat-state=!dstnat in-interface=ether1 comment="
drop access to clients behind NAT from WAN"
```

A ruleset is similar to input chain rules (accept established/related and drop invalid), except the first rule with `action=fasttrack-connection`. This rule allows established and related connections to bypass the firewall and significantly reduce CPU usage.

Another difference is the last rule which drops all new connection attempts from the WAN port to our LAN network (unless DstNat is used). Without this rule, if an attacker knows or guesses your local subnet, he/she can establish connections directly to local hosts and cause a security threat.

For more detailed examples on how to build firewalls will be discussed in the firewall section, or check directly Building Your First Firewall article.

Blocking Unwanted Websites

Sometimes you may want to block certain websites, for example, deny access to entertainment sites for employees, deny access to porn, and so on. This can be achieved by redirecting HTTP traffic to a proxy server and using an access-list to allow or deny certain websites.

First, we need to add a NAT rule to redirect HTTP to our proxy. We will use RouterOS built-in proxy server running on port 8080.

```
/ip firewall nat
  add chain=dst-nat protocol=tcp dst-port=80 src-address=192.168.88.0/24 action=redirect to-ports=8080
```

Enable web proxy and drop some websites:

```
/ip proxy set enabled=yes
/ip proxy access add dst-host=www.facebook.com action=deny
/ip proxy access add dst-host=*.youtube.* action=deny
/ip proxy access add dst-host=:vimeo action=deny
```

Using Winbox:

- On the left menu navigate to IP -> Web Proxy
- Web proxy settings dialog will appear.
- Check the "Enable" checkbox and click on the "Apply" button
- Then click on the "Access" button to open the "Web Proxy Access" dialog

- In the "Web Proxy Access" dialog click on "+" to add a new Web-proxy rule
- Enter the Dst hostname that you want to block, in this case, "www.facebook.com", and choose the action "deny"
- Then click on the "Ok" button to apply changes.
- Repeat the same to add other rules.

Troubleshooting

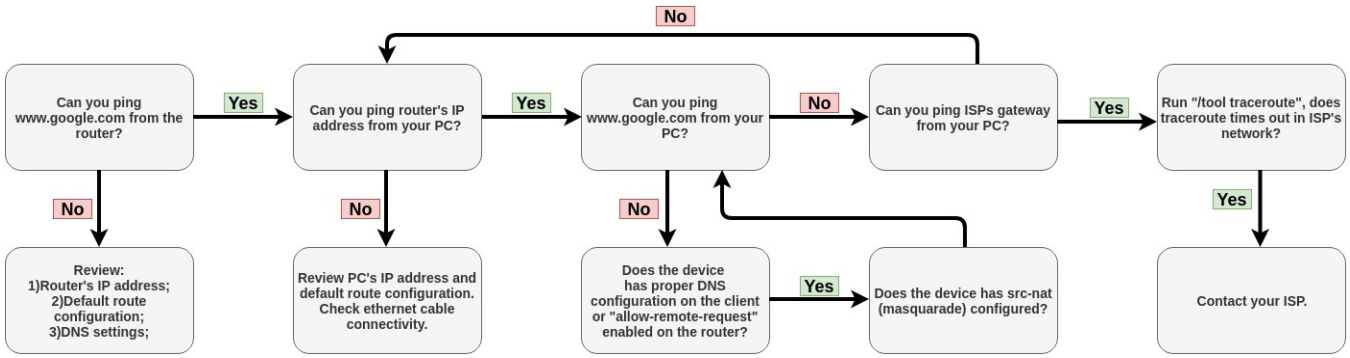
RouterOS has built-in various troubleshooting tools, like ping, traceroute, torch, packet sniffer, bandwidth test, etc.

We already used the ping tool in this article to verify internet connectivity.

Troubleshoot if ping fails

The problem with the ping tool is that it says only that the destination is **unreachable**, but no more detailed information is available. Let's overview the basic mistakes.

You cannot reach www.google.com from your computer which is connected to a MikroTik device:



If you are not sure how exactly to configure your gateway device, please reach out to MikroTik's official consultants for configuration support.

Upgrading and installation

Overview

MikroTik devices are preinstalled with RouterOS, so installation is usually not needed, except in the case where installing RouterOS on an x86 PC or virtual instance CHR. The upgrade procedure on already installed devices is straightforward.

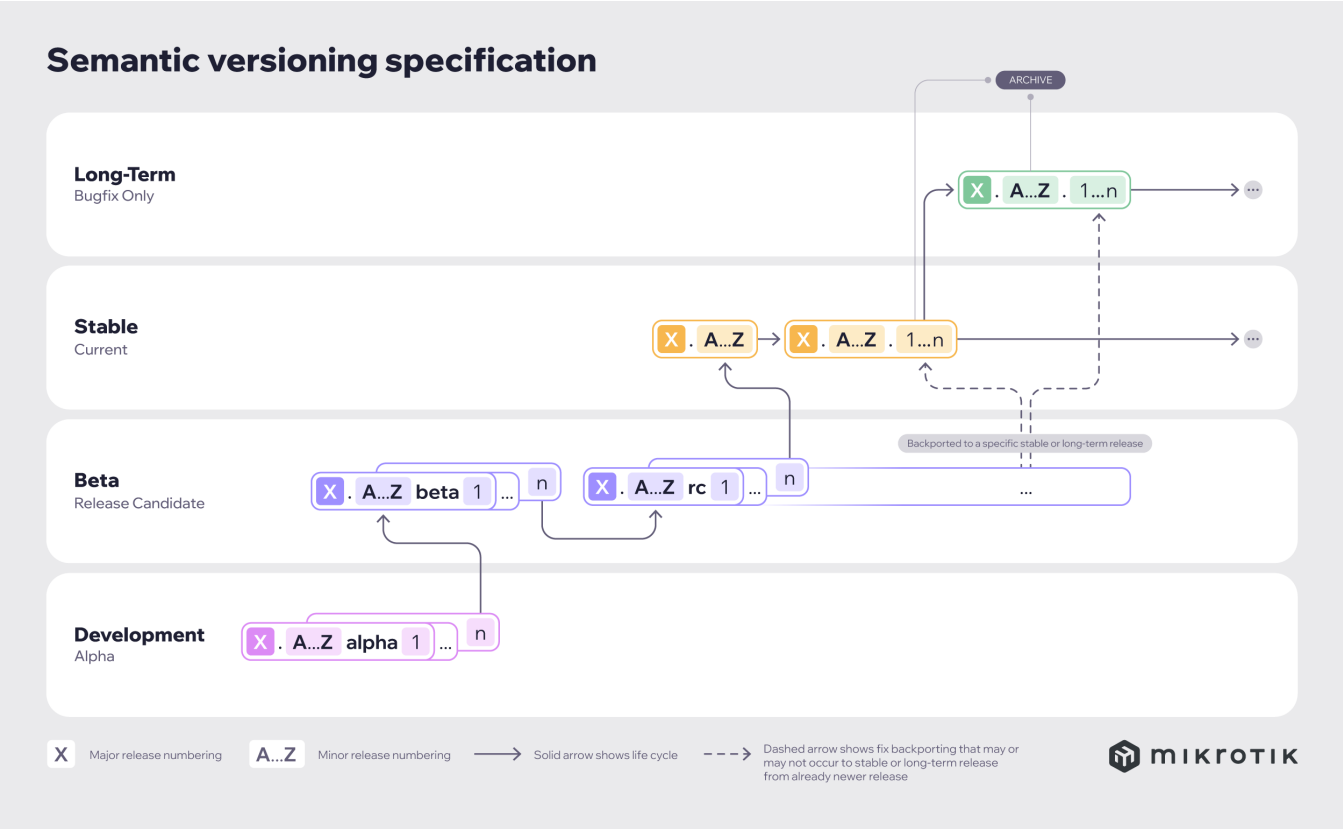
Upgrading

Version numbering

RouterOS versions are numbered sequentially when a period is used to separate sequences, it does *not* represent a decimal point, and the sequences do *not* have positional significance. An identifier of 2.5, for instance, is not "two and a half" or "halfway to version three", it is the fifth second-level revision of the second first-level revision. Therefore v5.2 is older than v5.18, which is newer.

RouterOS versions are released in several "release chains": Long term, Stable, Testing, and Development. When upgrading RouterOS, you can choose a release chain from which to install the new packages.

- **Long term**: Released rarely, and includes only the most critical fixes, upgrades within one number branch do not contain new features. When a **Stable** release has been out for a while and seems to be stable enough, it gets promoted into the long-term branch, replacing an older release, which is then moved to the archive. This consecutively adds new features.
- **Stable**: Released every few months, including all tested new features and fixes.
- **Testing**: Released every few weeks, only undergoes basic internal testing, and should not be used in production.
- **Development**: Released when necessary. Includes raw changes and is available for software enthusiasts for testing new features.

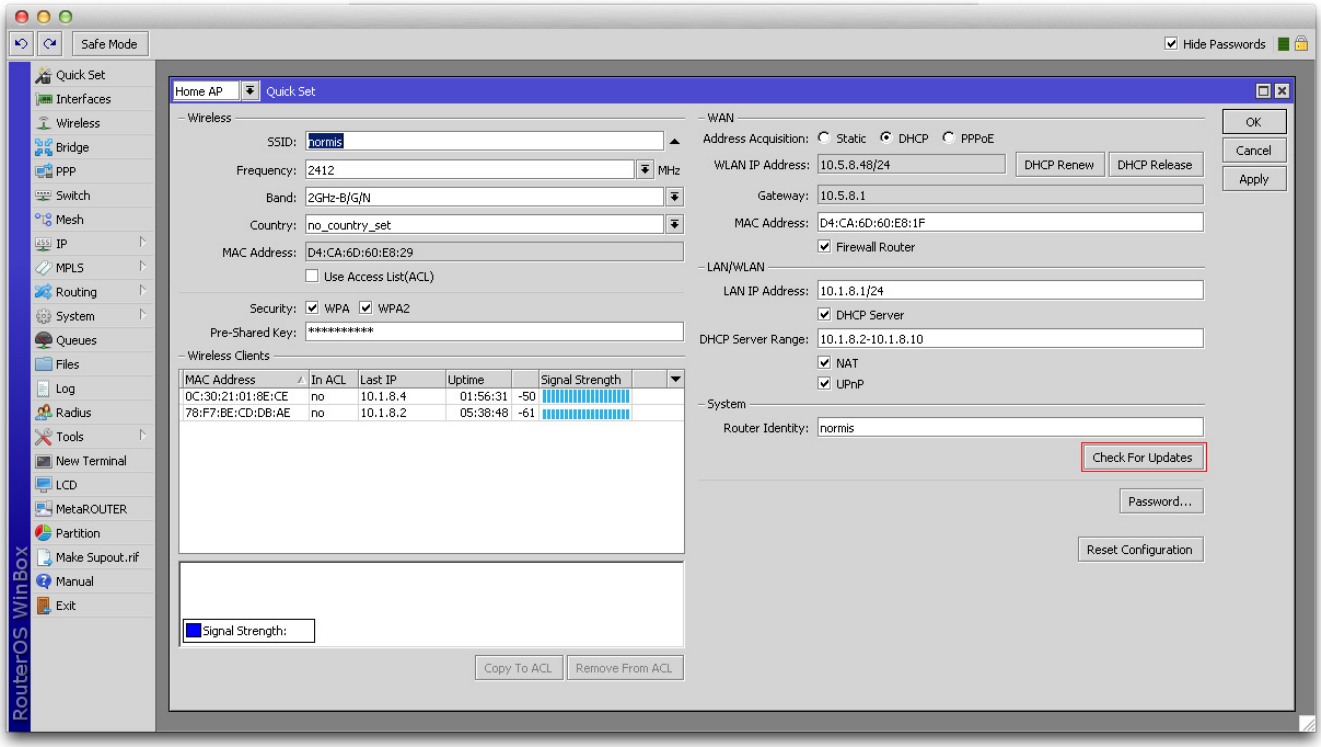


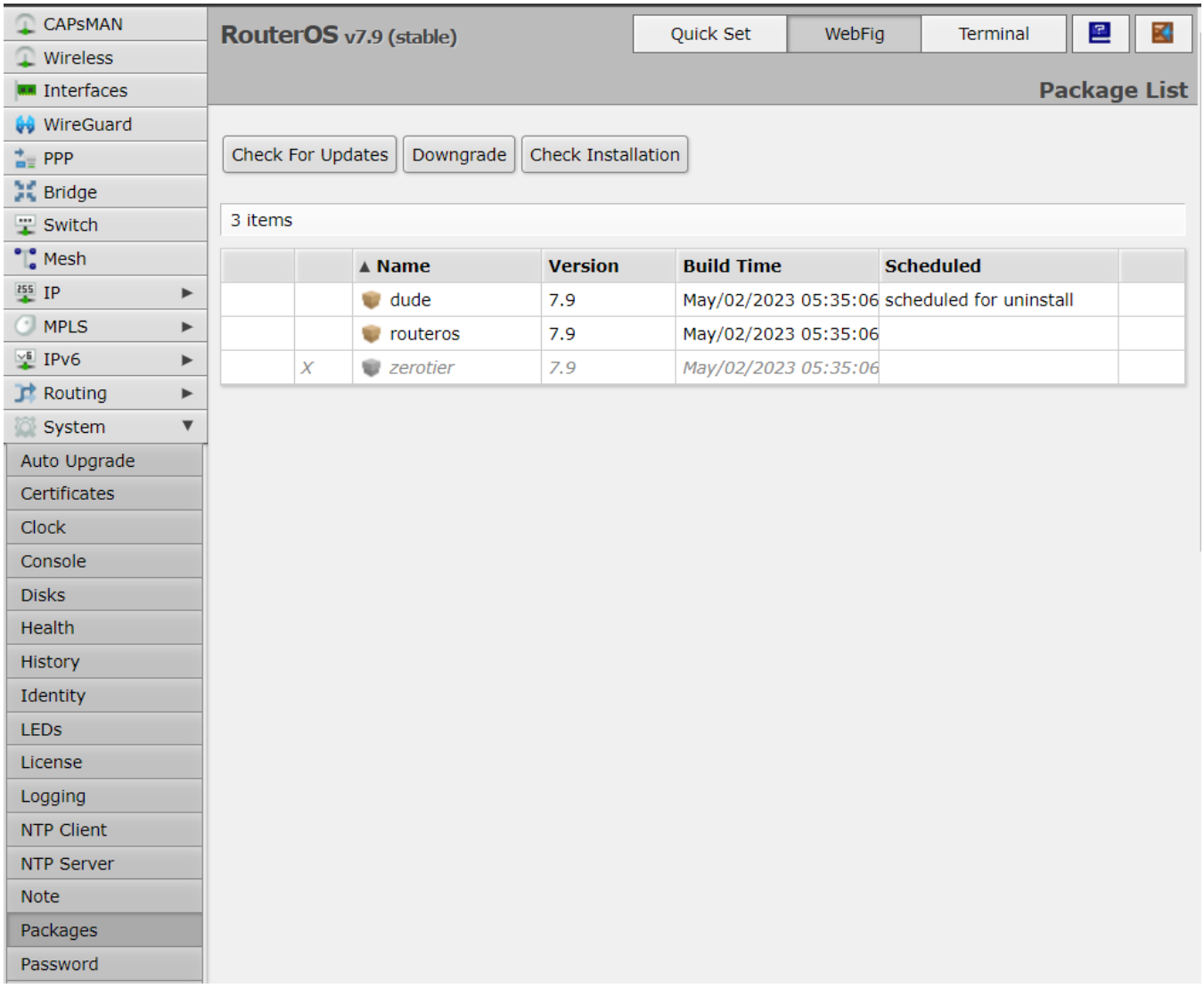
Standard upgrade

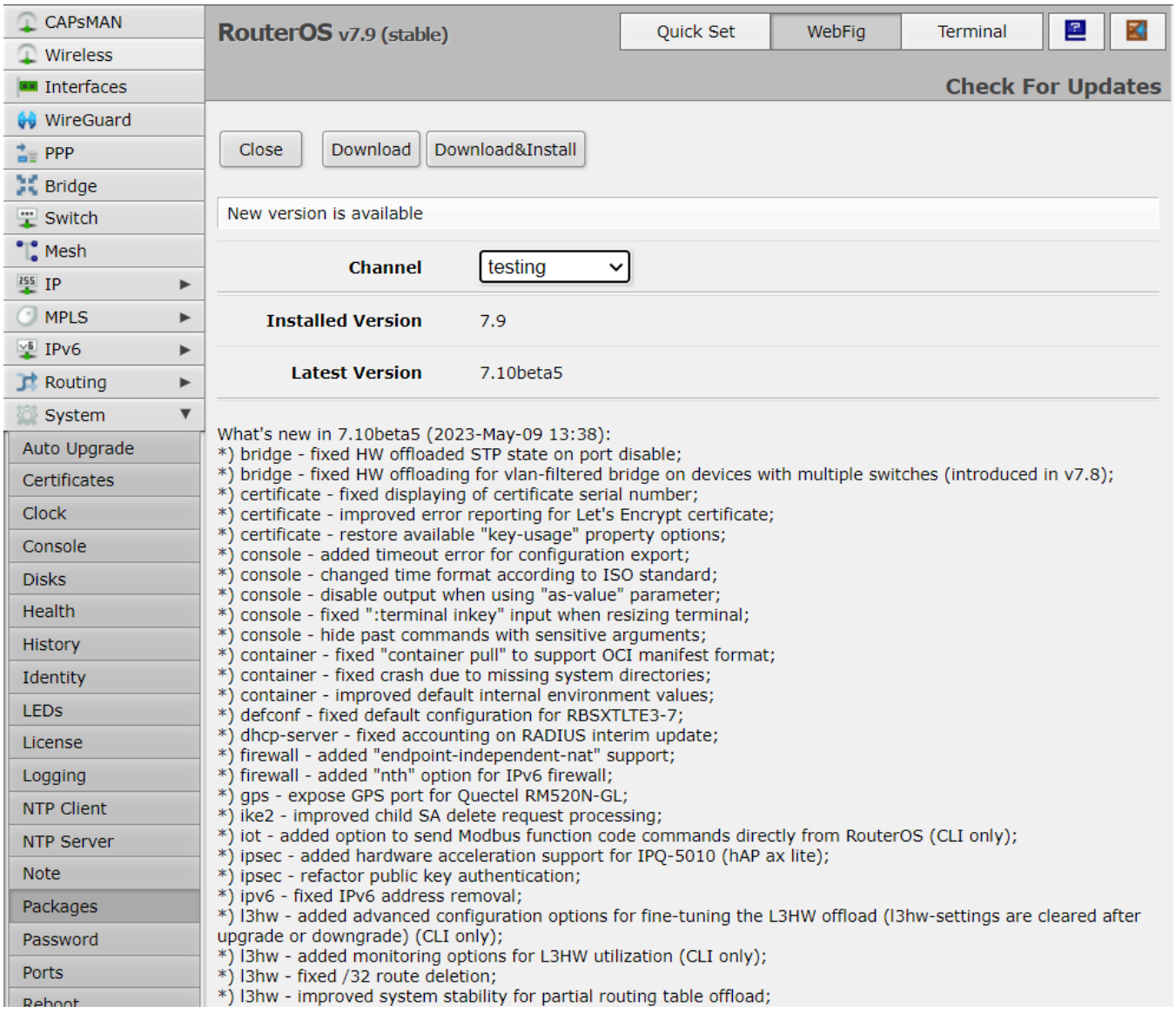
The package upgrade feature connects to the MikroTik download servers and checks if there is another RouterOS version for your device under the selected release channel. Can also be used for downgrading, if you, for example, are using stable release at the moment, but changed the release channel to the long-term.

After clicking the Upgrade button in QuickSet or in the Packages menu upgrade window will open with the current Changelog (if a newer version exists) and buttons to download and install the latest versions.

By clicking "Download & Install", downloads will start, and after a successful download will reboot to install the downloaded packages. Even if custom packages are installed, the downloader will take that into account and download all necessary packages.


Safe Mode
Hide Passwords
RouterOS WinBox
Quick Set
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
Radius
Tools
New Terminal
LCD
MetaROUTER
Partition
Make Supout.rif
Manual
Exit
Home AP
Quick Set
Wireless
SSID: normis
Frequency: 2412 MHz
Band: 2GHz-B/G/N
Country: no_country_set
MAC Address: D4:CA:6D:60:E8:29
Use Access List(ACL)
Security: WPA WPA2
Pre-Shared Key: **********
Wireless Clients
MAC Address | In ACL | Last IP | Uptime | Signal Strength
0C:30:21:01:8E:CE | no | 10.1.8.4 | 01:56:31 | -50
78:F7:BE:CD:DB:AE | no | 10.1.8.2 | 05:38:48 | -61
Signal Strength:
Copy To ACL
Remove From ACL
WAN
Address Acquisition: Static DHCP PPPoE
WLAN IP Address: 10.5.8.48/24
DHCP Renew
DHCP Release
Gateway: 10.5.8.1
MAC Address: D4:CA:6D:60:E8:1F
Firewall Router
LAN/WLAN
LAN IP Address: 10.1.8.1/24
DHCP Server
DHCP Server Range: 10.1.8.2-10.1.8.10
NAT
UPnP
System
Router Identity: normis
Check For Updates
Password...
Reset Configuration
OK
Cancel
Apply


CAPsMAN
Wireless
Interfaces
WireGuard
PPP
Bridge
Switch
Mesh
IP
MPLS
IPv6
Routing
System
Auto Upgrade
Certificates
Clock
Console
Disks
Health
History
Identity
LEDs
License
Logging
NTP Client
NTP Server
Note
Packages
Password
RouterOS v7.9 (stable)
Quick Set
WebFig
Terminal
Package List
Check For Updates
Downgrade
Check Installation
3 items
Name
Version
Build Time
Scheduled
dude
7.9
May/02/2023 05:35:06
scheduled for uninstall
routeros
7.9
May/02/2023 05:35:06
X
zerotier
7.9
May/02/2023 05:35:06


CAPsMAN
Wireless
Interfaces
WireGuard
PPP
Bridge
Switch
Mesh
IP
MPLS
IPv6
Routing
System
Auto Upgrade
Certificates
Clock
Console
Disks
Health
History
Identity
LEDs
License
Logging
NTP Client
NTP Server
Note
Packages
Password
Ports
Reboot
RouterOS v7.9 (stable)
Quick Set
WebFig
Terminal
Check For Updates
Close
Download
Download&Install
New version is available
Channel
testing
Installed Version
7.9
Latest Version
7.10beta5
What's new in 7.10beta5 (2023-May-09 13:38):
*) bridge - fixed HW offloaded STP state on port disable;
*) bridge - fixed HW offloading for vlan-filtered bridge on devices with multiple switches (introduced in v7.8);
*) certificate - fixed displaying of certificate serial number;
*) certificate - improved error reporting for Let's Encrypt certificate;
*) certificate - restore available "key-usage" property options;
*) console - added timeout error for configuration export;
*) console - changed time format according to ISO standard;
*) console - disable output when using "as-value" parameter;
*) console - fixed ":terminal inkey" input when resizing terminal;
*) console - hide past commands with sensitive arguments;
*) container - fixed "container pull" to support OCI manifest format;
*) container - fixed crash due to missing system directories;
*) container - improved default internal environment values;
*) defconf - fixed default configuration for RBSXTLTE3-7;
*) dhcp-server - fixed accounting on RADIUS interim update;
*) firewall - added "endpoint-independent-nat" support;
*) firewall - added "nth" option for IPv6 firewall;
*) gps - expose GPS port for Quectel RM520N-GL;
*) ike2 - improved child SA delete request processing;
*) iot - added option to send Modbus function code commands directly from RouterOS (CLI only);
*) ipsec - added hardware acceleration support for IPQ-5010 (hAP ax lite);
*) ipsec - refactor public key authentication;
*) ipv6 - fixed IPv6 address removal;
*) l3hw - added advanced configuration options for fine-tuning the L3HW offload (l3hw-settings are cleared after upgrade or downgrade) (CLI only);
*) l3hw - added monitoring options for L3HW utilization (CLI only);
*) l3hw - fixed /32 route deletion;
*) l3hw - improved system stability for partial routing table offload;

Manual upgrade

You can upgrade RouterOS in the following ways:

- Winbox – drag and drop files to the Files menu
- WebFig - upload files from the Files menu
- FTP - upload files to the root directory

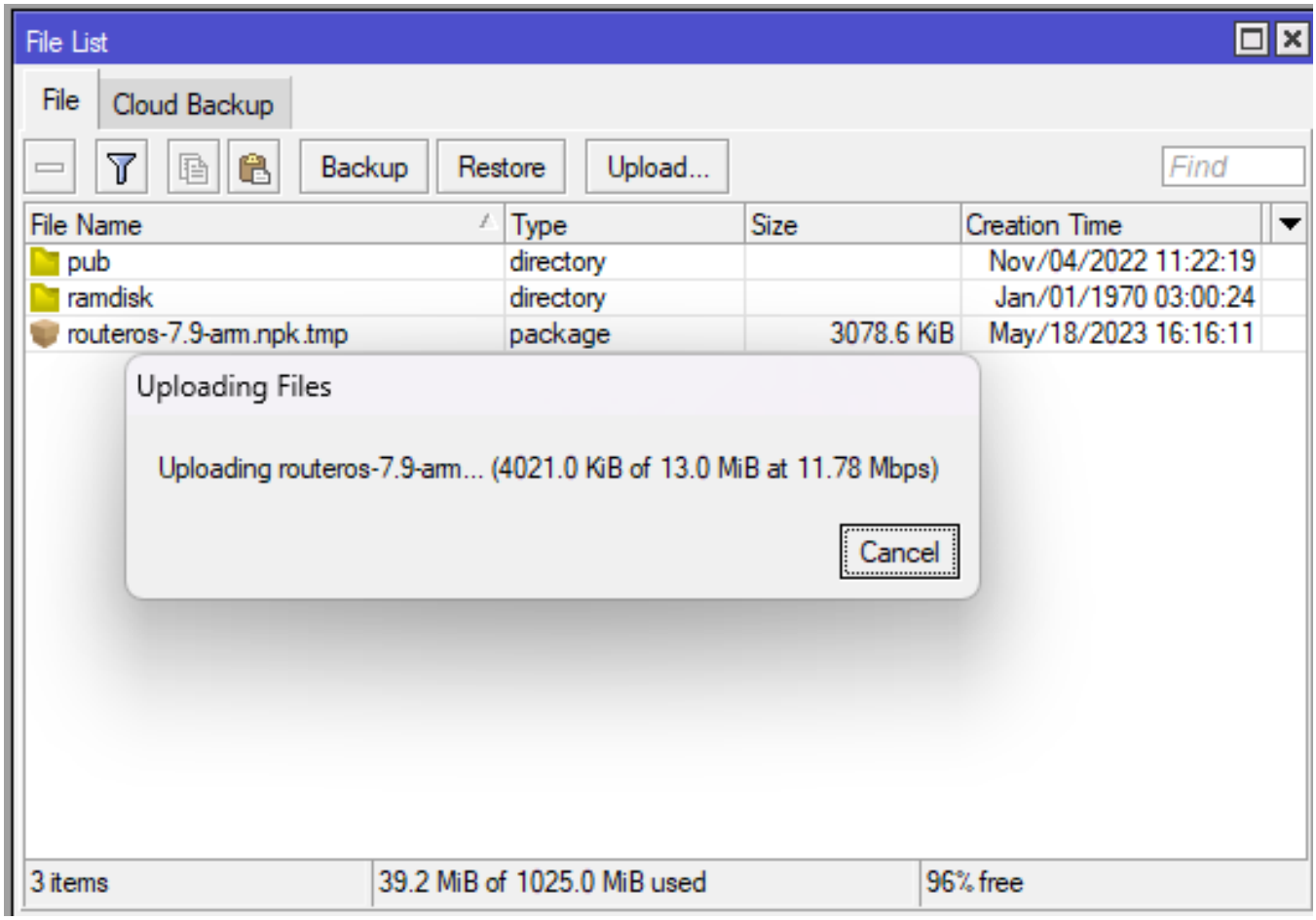
> ⚠ RouterOS cannot be upgraded through a serial cable. Only RouterBOOT is upgradeable using this method.

Manual upgrade process

- First step - visit www.mikrotik.com and head to the Software/download page, then choose the type of system you have the RouterOS installed on;
- Download the **routeros** ***(main)*** and extra packages that are installed on a device;
- Upload packages to a device using one of the previously mentioned methods:

Using Winbox

Choose your system type, and download the upgrade package. Connect to your router with Winbox, Select the downloaded file with your mouse, and drag it to the Files menu. If some files are already present, make sure to put the package in the root menu, not inside the hotspot folder! The upload will start.



After it finishes - reboot the device. The New version number will be seen in the Winbox Title and in the Packages menu

Using FTP

- Open your favorite SFTP program (in this case it is Filezilla), select the package, and upload it to your router (demo2.mt.lv is the address of my router in this example). note that in the image I'm uploading many packages, but in your case - you will have one file that contains them all
- if you wish, you can check if the file is successfully transferred onto the router (optional):

```
[admin@MikroTik] >/file print
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME                  TYPE       SIZE     CREATION-TIME
0  routeros-7.9-arm.npk  package    13.0MiB  may/18/2023 16:16:18
1  pub                   directory           nov/04/2022 11:22:19
2  ramdisk               directory           jan/01/1970 03:00:24
```

- reboot your router for the upgrade process to begin:

```
[admin@MikroTik] >/system reboot
Reboot, yes? [y/N]: y
```

- after the reboot, your router will be up to date, you can check it in this menu:

```
[admin@MikroTik] >/system package print
```

- if your router did not upgrade correctly, make sure you check the **log**

```
[admin@MikroTik] >/log print without-paging
```

RouterOS mass upgrade

You can upgrade multiple MikroTik routers within a few clicks. Let's have a look at a simple network with 3 routers (the same method works on networks with infinite numbers of routers),

RouterOS auto-upgrade

Sub-menu: `/system package update`

You can automate the upgrade process by running a script in the system scheduler. This script queries the MikroTik upgrade servers for new versions, if the response received says "New version is available", the script then issues the upgrade command:

```
[admin@MikroTik] >/system package update
check-for-updates once
:delay 3s;
:if ( [get status] = "New version is available") do={ install }
```

RouterOS can download software packages from a remote MikroTik router.

- Make one router as a network upgrade central point, that will update MikroTik RouterOS on other routers.
- Upload necessary RouterOS packages to this router (in the example, maps for RB751U and PowerPC for RB1100AHx2).
- Add the upgrade server router (192.168.100.1) information to a router that you want to update (192.168.100.253), required settings IP address /Username/Password
- Click on refresh to see available packages, download the newest packages, and reboot the router to finalize the upgrade.

License issues

When upgrading from older versions, there could be issues with your license key. Possible scenarios:

- When upgrading from RouterOS v2.8 or older, the system might complain about an expired upgrade time. To override this, use Netinstall to upgrade. Netinstall will ignore old license restrictions and will upgrade
- When upgrading to RouterOS v4 or newer, the system will ask you to update the license to a new format. To do this, ensure your Winbox PC (not the router) has a working internet connection without any restrictions to reach www.mikrotik.com and click "update license" in the license menu.

Suggestions

When using a RouterBOARD device, it is always suggested to upgrade its RouterBOOT bootloader after RouterOS is upgraded. To do this, issue the "*/system routerboard upgrade*" command in CLI, followed by a reboot. Alternatively, navigate to the GUI System → RouterBOARD menu and click the "Upgrade" button, then reboot the device.

Netinstall

NetInstall is the most commonly used installation tool. It runs on Windows machines or Linux with Wine (superuser permissions are required).

You can download the NetInstall utility on the https://www.mikrotik.com/download download section.

NetInstall is also used to re-install RouterOS in cases where the previous install failed, became damaged or access passwords were lost.

Your device must support booting from ethernet, and there must be a direct ethernet link from the NetInstall computer to the target device. All RouterBOARDs support PXE network booting, it must be enabled inside RouterOS "routerboard" menu if RouterOS is operable or in the bootloader settings. For this, you will need a serial cable.
Note: For RouterBOARD devices with no serial port, and no RouterOS access, the reset button can also start PXE booting mode. See your RouterBOARD manual PDF for details.

NetInstall can also directly install RouterOS on a disk (USB/CF/IDE/SATA) that is connected to the Netinstall Windows machine. After installation just move the disk to the Router machine and boot from it.

User Interface

The following options are available in the NetInstall window:

- **Routers/Drives** - list of PC drives and PXE booted routers. Select from the list on which drive or router you want to install RouterOS.
- **Net booting** - used to enable PXE booting over the network.
- **Install/Cancel** - after selecting the router and selecting the RouterOS packages below, use this to start the installation.
- **SoftID** - the SoftID that was generated on the router. Use this to purchase your key.
- **Key / Browse** - apply the purchased key here, or leave it blank to install a 24h trial.
- **Get key** - get the key from your mikrotik.com account directly.
- **Flashfig** - launch Flashfig - the mass config utility that works on brand new devices.
- **Keep old configuration** - involves downloading the configuration database from the router, reinstalling the router (including disk formatting), and uploading the configuration files back to it. However, it's important to note that this process solely applies to the configuration itself and does not impact the files, including databases like the User Manager database, Dude database, and others.
- **IP address / Netmask** - enter an IP address and netmask in CIDR notation to preconfigure the router.
- **Gateway** - default gateway to preconfigure in the router.
- **Baud rate** - default serial port baud rate to preconfigure in the router.
- **Configure script File** - a file that contains RouterOS CLI commands that directly configure the router (e.g. commands produced by export command). Used to apply default configuration.

Attention! Do not try to install RouterOS on your system drive. Action will format your hard drive and wipe out your existing OS.

CD Install

RouterOS Package Types

Information about RouterOS packages can be found here

Packages

Summary

In RouterOS v7, most of the features are combined in one ***routeros*** (system) package.

Installing the corresponding package can enable specific features (like container, dude).

Packages are provided only by MikroTik, and no 3rd parties are allowed to make them.

Acquiring packages

Packages can be downloaded from the MikroTik download page.

RouterOS packages

Starting from RouterOS 7.13, the ***routeros*** (system) package and one of the wireless packages are needed for the basic operation of a simple home router.

802.11ax WiFi APs require radio drivers, which are provided by the ***wifi-qcom*** package or (for RouterOS version before 7.13) the ***wifiwave2*** package.

Previous generation WiFi APs require a ***wireless*** package.

Other packages are optional and not required for a home router. Install them only if you are sure of their purpose.

System packages

Package	Description
routeros-arm (*arm*)	system package for arm devices
routeros-arm *(arm64)*	system package for arm64 devices
routeros-mipsbe (*mipsbe*)	system package for mipsbe devices
routeros-mmips (*mmips*)	system package for mmips devices
routeros-smips (*smips*)	system package for smips devices
routeros-tile (*tile*)	system package for tile devices
routeros-ppc (*ppc*)	system package for ppc devices
routeros (*x86, CHR*)	system package for x86 installations and CHR environment

Extra packages

To install extra packages, download the necessary package from the MikroTik download page, selecting the RouterOS v6 section based on your device's architecture found in the System/Resources menu. Extract the archive and upload the required package to your router using any convenient method, and proceed to reboot the router.

⚠ Certain packages, such as Container, require physical access to the router for installation.

Package	Description
calea (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	Data gathering tool for specific use due to "Communications Assistance for Law Enforcement Act" in the USA
container *(arm, arm64, x86, CHR)*	Container implementation of Linux containers, allows users to run containerized environments within RouterOS
dude (*arm, arm64, mmips, tile, x86, CHR*)	Dude tool that allows monitoring of network environment
gps (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	Global Positioning System devices support
iot (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	Enables: • MQTT • LoRa (for devices with LR8/9/2 miniPCie cards) • Bluetooth (for devices with Bluetooth chip) • GPIO (for devices with GPIO pins) • Modbus (for devices with RS485 port)
lora *(arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR)*	Dummy package for Lora support. LoRa package is not obligatory anymore and is left only for compatibility reasons. LoRa functionality is moved into iot package.
lte *(mipsbe)*	Required package only for SXT LTE (RBSXTLTE3-7), which contains drivers for the built-in LTE interface.
rose-storage (*arm, arm64, tile, x86, CHR*)	Additional enterprise data center functionality in RouterOS, support disk monitoring, improved formatting, RAIDs, rsync, iSCSI , NVMe over TCP, NFS, and improved SMB
tr069-client (*arm, arm64, mipsbe, mmips, smips, tile, ppc, x86, CHR*)	TR069 Client package
ups (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	APC ups management interface
user-manager (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	MikroTik User Manager server for controlling Hotspot and other service users.
wifiwave2 (*arm, arm64, mmips, tile, ppc, x86, CHR*)	For 7.12 and older versions: WifiWave2 package for managing compatible 802.11ax and 802.11ac wave 2 wireless interfaces and WifiWave2 CAPsMAN for central WifiWave2 device management. Mandatory for 802.11ax devices.
wifi-qcom (*arm, arm64*)	Mandatory driver package for 802.11ax interfaces. Introduced in 7.13. Wifi CAPsMAN support comes with the system package.
wifi-qcom-ac (*arm*)	Optional Wifi driver package for compatible 802.11ac interfaces. Introduced in 7.13.
wireless (*arm, arm64, mipsbe, mmips, tile, ppc, x86, CHR*)	Utilities and drivers for managing WiFi (up to 802.11ac) and 60GHz wireless interfaces. This package is bundled into RouterOS for versions up to 7.12. Starting with 7.13, it is a separate package. The **wireless** package conflicts with **wifi-qcom** and **wifi-qcom-ac** packages - they cannot be active at the same time.
zerotier *(arm, arm64)*	Enables ZeroTier functionality

Working with packages

Menu: */system package*

Commands executed in this menu will take place only on restart of the router. Until then, the user can freely schedule or revert set actions.

Command	Description
disable	schedule the package to be disabled after the next reboot. No features provided by the package will be accessible

downgrade	will prompt for the reboot. During the reboot process will try to downgrade the RouterOS to the oldest version possible by checking the packages that are uploaded to the router.
enable	schedule package to be enabled after the next reboot
uninstall	schedule package to be removed from the router. That will take place during the reboot.
unschedule	remove scheduled task for the package.
print	outputs information about the packages, like: version, package state, planned state changes, etc.

Menu: */system/check-installation*

The "Check installation" function ensures the integrity of the RouterOS system by verifying the readability and correct placement of files. Its primary purpose is to confirm the health and status of your NAND/Flash storage.

Examples

The upgrade process is described here.

List of available packages.

zerotier package is disabled and dude package is scheduled for uninstall.

```
/system package print
Flags: X - DISABLED
Columns: NAME, VERSION, SCHEDULED
#   NAME      VERSION  SCHEDULED
0   dude      7.9      scheduled for uninstall
1 X zerotier  7.9
2   routeros  7.9
```

Uninstall package

```
/system package uninstall dude; /system reboot;
Reboot, yes? [y/N]:
```

Disable package

```
/system package disable zerotier; /system reboot;
Reboot, yes? [y/N]:
```

Downgrade

```
/system package downgrade; /system reboot;
Reboot, yes? [y/N]:
```

Cancel uninstall or disable action

```
/system package unschedule zerotier;
/system package unschedule dude;
```

Configuration Management

Overview

This article describes a set of commands used for configuration management.

Configuration Undo and Redo

Any action done in the GUI or any command executed from the CLI is recorded in `/system history`. You can undo or redo any action by running undo or redo commands from the CLI or by clicking on Undo, and Redo buttons from the GUI.

A simple example to demonstrate the addition of the firewall rule and how to undo and redo the action:

```
[admin@v7_ccr_bgp] /ip/firewall/filter> add chain=forward action=drop

[admin@v7_ccr_bgp] /ip/firewall/filter> print
Flags: X - disabled, I - invalid; D - dynamic
0 X chain=input action=drop protocol=icmp src-address=10.155.101.1 log=no
log-prefix=""

1 chain=forward action=drop

[admin@v7_ccr_bgp] /ip/firewall/filter> /system/history/print
Flags: U - undoable, R - redoable, F - floating-undo
Columns: ACTION, BY, POLICy
ACTION BY POLIC
F filter rule added admin write
U --- write
[admin@v7_ccr_bgp] /ip/firewall/filter>
```

We have added a firewall rule and in `/system history` we can see all that was done.

Let's undo everything:

```
[admin@v7_ccr_bgp] /ip/firewall/filter> /undo
[admin@v7_ccr_bgp] /ip/firewall/filter> print
Flags: X - disabled, I - invalid; D - dynamic
0 X chain=input action=drop protocol=icmp src-address=10.155.101.1 log=no
log-prefix=""

[admin@v7_ccr_bgp] /ip/firewall/filter>
```

As you can see firewall rule disappeared.
Now redo the last change:

```
[admin@v7_ccr_bgp] /ip/firewall/filter> /redo
[admin@v7_ccr_bgp] /ip/firewall/filter> print
Flags: X - disabled, I - invalid; D - dynamic
0 X chain=input action=drop protocol=icmp src-address=10.155.101.1 log=no
log-prefix=""

1 chain=forward action=drop

[admin@v7_ccr_bgp] /ip/firewall/filter>
```

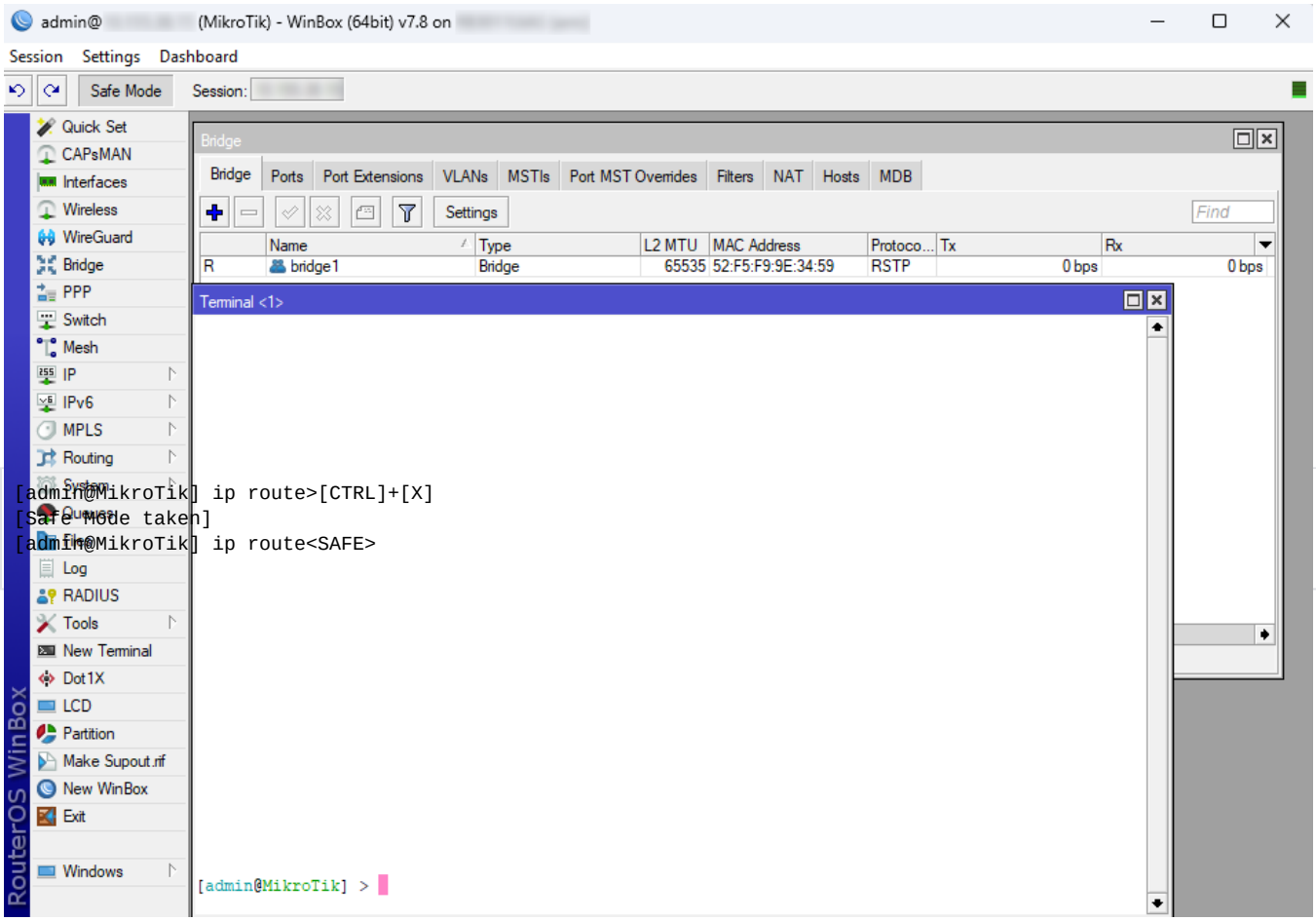
System history is capable of showing exact CLI commands that will be executed during "Undo" or "Redo" actions even if we perform the action from GUI, for example, detailed history output after adding TCP accept rule from WinBox:

```
[admin@v7_ccr_bgp] /system/history> print detail
Flags: U - undoable, R - redoable, F - floating-undo
 F redo=
      /ip firewall filter add action=accept chain=forward disabled=no log=no \
          log-prefix="" protocol=tcp
    undo=/ip firewall filter remove *4 action="filter rule added" by="admin"
    policy=write time=oct/10/2019 18:51:05

 F redo=/ip firewall filter add action=accept chain=forward
    undo=/ip firewall filter remove *3 action="filter rule added" by="admin"
    policy=write time=oct/10/2019 18:49:03


U redo="" undo="" action="---" by="" policy=write time=sep/27/2019 13:07:35
[admin@v7_ccr_bgp] /system/history>
```

Safe Mode

It is sometimes possible to change router configuration in a way that will make the router inaccessible (except from the local console). Usually, this is done by accident, but there is no way to undo the last change when the connection to the router is already cut. Safe mode can be used to minimize such risk.

The **"Safe Mode"** button in the Winbox GUI allows you to enter Safe Mode, while in the CLI, you can access it by either using the keyboard shortcut **F4** or pressing **[CTRL]+[X]**. To exit without saving the made changes in CLI, hit **[CTRL]+[D].**



Safe Mode taken is displayed and prompt changes to reflect that session is now in safe mode. All configuration changes that are made (also from other login sessions), while the router is in safe mode, are automatically undone if the safe mode session terminates abnormally. You can see all such changes that will be automatically undone and tagged with an **F** flag in the system history:

```
[admin@MikroTik] /ip/route>
[Safe Mode taken]
[admin@MikroTik] /ip/route<SAFE> add
[admin@MikroTik] /ip/route<SAFE> /system/history/print
Flags: U, F - FLOATING-UNDO
Columns:
 ACTION, BY, POLICY ACTION BY POLICY
F route 0.0.0.0/0 added admin write
```

Now, if the telnet connection (or WinBox terminal) is cut, then after a while (TCP timeout is **9** minutes) all changes that were made while in safe mode will be undone. Exiting session by **[Ctrl]+[D]** also undoes all safe mode changes, while **/quit** does not.

If another user tries to enter safe mode, he's given the following message:

```
[admin@MikroTik] >
Hijacking Safe Mode from someone - unroll/release/don't take it [u/r/d]:
```

- [u] - undoes all safe mode changes, and puts the current session in safe mode.
- [r] - keeps all current safe mode changes, and puts the current session in a safe mode. The previous owner of safe mode is notified about this:

```
[admin@MikroTik] ip firewall rule input
[Safe mode released by another user]
```

- [d] - leaves everything as-is.

If too many changes are made while in safe mode, and there's no room in history to hold them all (currently history keeps up to 100 most recent actions), then the session is automatically put out of the safe mode, and no changes are automatically undone. Thus, it is best to change the configuration in small steps, while in safe mode. Pressing **[Ctrl]+[X]** twice is an easy way to empty the safe mode action list.

System Backup and Restore

System backup is the way to completely clone router configuration in binary format.

More information about Backup and Restore is found here.

Configuration Export and Import

RouterOS allows exporting and importing parts of the configuration in plain text format. This method can be used to copy bits of configuration between different devices, for example, clone the whole firewall from one router to another.

An export command can be executed from each menu (resulting in configuration export only from this specific menu and all its sub-menus) or from the root menu for complete config export and is available for CLI only.

The Export command does not export system user passwords, installed certificates, SSH keys, Dude, or a User-manager database.

Installed certificates, Dude, and User-manager databases must be manually exported and imported into a new device.

System user passwords and user SSH keys can not be exported.

During config import, we suggest using the same RouterOS version used during config export to prevent cases when some of the commands do not exist in one or another RouterOS version.

Configuration Export

The following command parameters are accepted:

Property	Description
compact	Output only modified configuration, the default behavior
file	Export configuration to a specified file. When the file is not specified export output will be printed to the terminal
show-sensitive *(yes|no; Default: no)*. **RouterOS version 7 only** **hide-sensitive** *(yes|no; Default: no)*. **RouterOS version 6 only**	Show sensitive information, like passwords, keys, etc. Hide sensitive information, like passwords, keys, etc.
terse	With this parameter, the export command will output only configuration parameters, without defaults.
verbose	With this parameter, the export command will output whole configuration parameters and items including defaults.

For example, export configuration from `/ip address` the menu and save it to a file:

```
[admin@MikroTik] > /ip address print
Flags: X - disabled, I - invalid, D - dynamic
#   ADDRESS            NETWORK         BROADCAST       INTERFACE
0   10.1.0.172/24      10.1.0.0        10.1.0.255      bridge1
1   10.5.1.1/24        10.5.1.0        10.5.1.255      ether1
[admin@MikroTik] > /ip address export file=address
[admin@MikroTik] > /file print
# NAME                             TYPE         SIZE       CREATION-TIME
0  address.rsc                      script       315        dec/23/2003 13:21:48
[admin@MikroTik] >
```

By default, the export command writes only user-edited configuration, RouterOS defaults are omitted.

For example, the IPSec default policy will not be exported, and if we change one property then only our change will be exported:

```
[admin@rack1_b4] /ip ipsec policy> print
Flags: T - template, X - disabled, D - dynamic, I - inactive, * - default
0 T * group=default src-address=::/0 dst-address=::/0 protocol=all
      proposal=default template=yes
[admin@rack1_b4] /ip ipsec policy> export
# apr/02/1970 17:59:14 by RouterOS 6.22
# software id = DB0D-LK67
#
[admin@rack1_b4] /ip ipsec policy> set 0 protocol=gre
[admin@rack1_b4] /ip ipsec policy> export
# apr/02/1970 17:59:30 by RouterOS 6.22
# software id = DB0D-LK67
#
/ip ipsec policy
set 0 protocol=gre
```

Note:

The ***** flag, indicates that the entry is system default and cannot be removed manually.

Here is the list of all menus containing default system entries

Menu	Default Entry
/interface wireless security-profiles	default
/ppp profile	"default", "default-encryption"
/ip hotspot profile	default
/ip hotspot user profile	default
/ip ipsec policy	default
/ip ipsec policy group	default
/ip ipsec proposal	default
/ip ipsec mode-conf	read-only
/ip smb shares	pub
/ip smb users	guest
/ipv6 nd	any
/mpls interface	all
/routing bfd interface	all

/routing bgp instance	default
/routing ospf instance	default
/routing ospf area	backbone
/routing ospf-v3 instance	defailt
/routing ospf-v3 area	backbone
/snmp community	public
/tool mac-server mac-winbox	all
/tool mac-server	all
/system logging	"info", "error", "warning", "critical"
/system logging action	"memory", "disk", "echo", "remote"
/queue type	"default", "ethernet-default", "wireless-default", "synchronous-default", "hotspot-default", "only-hardware-queue", "multi-queue-ethernet-default", "default-small"

If some specific menu will not be able to respond to the export command, starting from the RouterOS v7.11, an error message will be printed out in the export command output after a timeout ("#error exporting "/xxx" (timeout)") and the process will move on to the next menu.

Starting from RouterOS 7.13, you can export parts of a specific menu. For instance, it is possible to export a specific address-list among multiple address-lists on your router.

```
[admin@MikroTik] > ip firewall address-list export where list=mylist
```

Configuration Import

Root menu command import allows running configuration script from the specified file. Script file (with extension ".rsc") can contain any console command including complex scripts.

For example, load saved configuration file

```
[admin@MikroTik] > import address.rsc
Opening script file address.rsc

Script file loaded and executed successfully
[admin@MikroTik] >
```

Import command allows to specify the following parameters:

Property	Description
from-line	Start executing the script from the specified line number
file-name	Name of the script (.rsc) file to be executed.
verbose	Reads each line from the file and executes individually, allowing to debug syntax or other errors more easily.

If the device has a default or existing configuration that requires replacement, it is necessary to initiate a configuration reset.

This involves applying a clean, empty configuration using the command /system/reset-configuration no-defaults=yes, followed by a device reboot.

Auto Import

It is also possible to **automatically** execute scripts after uploading to the router with FTP or SFTP. The script file must be named with the extension *.auto.rsc. Once the commands in the file are executed, a new *.auto.log file is created which contains import success or failure information.

".auto.rsc" in the filename is mandatory for a file to be automatically executed.

Import troubleshooting

Configuration parts to watch out for in exported .rsc files

Things that should be removed from export files that were created with "/export", before attempting import on a new device.

- Interface renaming conflicts with the default ethernet naming scheme.

```
/interface ethernet
set [ find default-name=ether5 ] auto-negotiation=no name=ether1-gateway
set [ find default-name=ether6 ] name=ether2
set [ find default-name=ether7 ] name=ether3
set [ find default-name=ether8 ] name=ether4
set [ find default-name=ether1 ] name=ether5
set [ find default-name=ether2 ] name=ether6
set [ find default-name=ether3 ] name=ether7
set [ find default-name=ether4 ] name=ether8
```

- In older versions "export" default entries might show with "add" instead of the "set" command. That should be edited before import to avoid errors.
- Check if the total number of physical interfaces count matches the new and old devices. If there are some missing that will end up in error during .rsc import.

In case of problematic import, attempt the following:

- Reset the configuration on that device.
- Run the import command again with the "verbose=yes" argument. It will also stop the import process on a problem that you already encountered, but will also show the place where the export failed. This way shows you the place where things need to be edited in the .rsc import file.

Startup delay

If your configuration relies on interfaces that might not yet have started up upon command execution, it is suggested to introduce delays or to monitor until all needed interfaces are available. This example script allows you to set how many interfaces you are expecting, and how long to wait until they become available:

```
{
:local i 0
#Number of interfaces. It is necessary to reconfigure this number for each device (/interface print count-only)
:local x 10
#Max time to wait
:local t 30
while ($i < $t && [:len [/interface find]] < $x) do={
:put $i
:set $i ($i + 1)
:delay 1
}
if ($i = $t) do={
:log warning message="Could not load all physical interfaces"
} else={
#Rest of your script
}
}
```

The above script will wait until there are 10 interfaces visible, or 30 seconds. If there are no 10 interfaces at this time, it will put a message in the log. Modify the variables according to your needs.

Configuration Reset

RouterOS allows resetting configuration with `/system reset-configuration` command

This command clears all configuration of the router and sets it to the factory defaults including the login name and password ('admin' with an empty password or, for some models, check user and wireless passwords on the sticker). For more details on the default configuration see the list.

After executing the configuration reset command, the router will reboot and load the default configuration. Starting from version 7.13, following the reset, a license prompt will be displayed with the option to view the end-user license agreement.

The backup file of the existing configuration is stored before reset. That way you can easily restore any previous configuration if the reset is done by mistake.

If the router was installed using Netinstall and had a script specified as the initial configuration, the reset command executes this script after purging the configuration. To stop it from doing so, you will have to reinstall the router.

It is possible to override the default reset behavior with the parameters below:

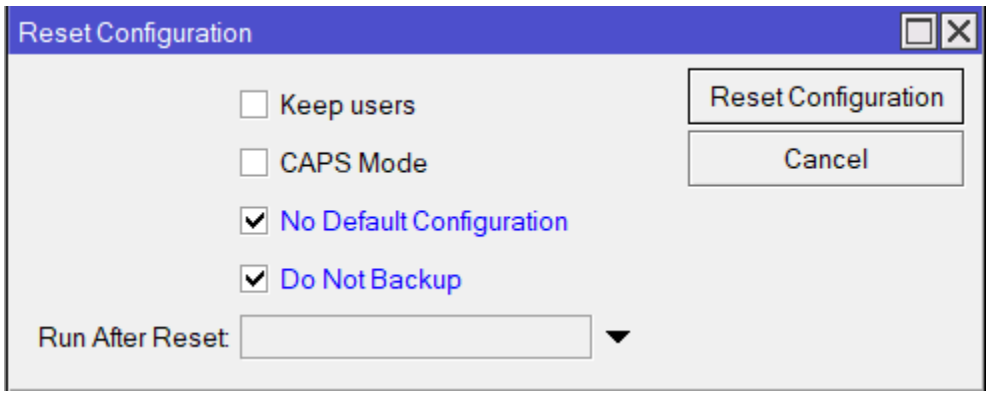
Property	Description
keep-users	Do not remove existing users from the configuration
no-defaults	Do not load the default configuration, just clear the configuration
skip-backup	Skip automatic backup file generation before reset
run-after-reset	Run specified .rsc file after reset. That way you can load your custom configuration.

If a specific .rsc file execution takes more than 2 minutes, a script will fail, and LOG will contain a "runtime limit exceeded" error.

For example hard reset configuration without loading default config and skipping backup file:

```
[admin@MikroTik] > /system reset-configuration no-defaults=yes skip-backup=yes
Dangerous! Reset anyway? [y/N]: y
```

And the same using Winbox:



Default configurations

All MikroTik devices come with some kind of default configuration. There are several different configurations depending on board type:

- CPE Router;
- LTE CPE AP router;
- AP Router (single or dual-band);
- PTP Bridge, W60G Bridge (AP or CPE);
- WISP Bridge (AP in ap_bridge mode);
- Switch;
- IP Only;
- CAP.

You can run the command `/system default-configuration print` to see the exact applied default configuration commands.

CPE Router

In this type of configuration, the router is configured as a wireless client device. WAN interface is a **Wireless** interface. WAN port has configured DHCP client, is protected by IP firewall and MAC discovery/connection is disabled.

List of routers using this type of configuration:

- RB 711,911,912,921,922 - with level3 license
- SXT
- QRT
- SEXTANT
- LHG
- LDF
- DISC
- Groove
- Metal

CPE RouterMode:

- * wireless interface connected to providers network (WAN port);
- * WAN port is protected by firewall and enabled DHCP client

wlan1 Configuration:

- mode: station;
- band: 2ghz-b/g/n;
- tx-chains: 0;1;
- rx-chains: 0;1;
- installation: outdoor;
- wpa2: no;
- ht-extension: 20/40mhz-XX;

LAN Configuration:

- IP address 192.168.88.1/24 is set on ether1 (LAN port)
- DHCP Server: enabled;
- DNS: enabled;

WAN (gateway) Configuration:

- gateway:wlan1 ;
- ip4 firewall: enabled;
- ip6 firewall: enabled;
- NAT: enabled;
- DHCP Client: enabled;

Login

- admin user protected by a password

Configuration **preview** :

.

LTE CPE AP router

This configuration type is applied to routers that have both LTE and wireless interfaces. LTE interface is considered a WAN port protected by a firewall and MAC discovery/connection disabled. The IP address on the WAN port is acquired automatically. Wireless is configured as an access point and bridged with all available Ethernet ports.

List of routers using this type of configuration:

- wAP LTE Kit
- SXT LTE
- LtAP 4G kit
- LtAP LTE kit
- Chateau

CPE RouterMode:

* wireless interface connected to providers network (WAN>

* WAN port is protected by firewall and enabled DHCP cli>

LAN Configuration:

- The IP address 192.168.188.1/24 is set on the bridge (LAN por>
- DHCP Server: enabled;
- DNS: enabled;

WAN (gateway) Configuration:

- gateway:lte1 ;
- ip4 firewall: enabled;
- ip6 firewall: enabled;
- NAT: enabled;

Login

- admin user protected by a password

Configuration **preview** :

.

AP Router

This type of configuration is applied to home access point routers to be used straight out of the box without additional configuration (except router passwords and wireless keys)

First Ethernet is always configured as a WAN port (protected by a firewall, enabled DHCP client, and disabled MAC connection/discovery). Other Ethernet ports and wireless interfaces are added to the local LAN bridge with 192.168.88.1/24 address set and configured DHCP server. In the case of dual-band routers, one wireless is configured as a 5 GHz access point and the other as a 2.4 GHz access point.

List of routers using this type of configuration:

- RB 450,751,850,951,953,2011,3011,4011
- hEX, PowerBox
- mAP
- wAP, wAP R (without LTE card)
- hAP
- cAP
- OmniTIK
- CRS series with wireless interface
- L009 series
- Audience
- Knot
- PWR

RouterMode:

* WAN port is protected by firewall and enabled DHCP cli>

* Wireless and Ethernet interfaces (except WAN port/s) are part of the LAN bridge

LAN Configuration:

- The IP address 192.168.88.1/24 is set on the bridge (LAN port)
- DHCP Server: enabled;
- DNS: enabled;

wlan1 Configuration:

- mode: ap-bridge;
- band: 2ghz-b/g/n;
- tx-chains: 0;1;
- rx-chains: 0;1;
- installation: indoor;
- wpa2: no;
- ht-extension: 20/40mhz-XX;

WAN (gateway) Configuration:

- ip4 firewall: enabled;
- ip6 firewall: enabled;
- NAT: enabled;
- DHCP Client: enabled;

Login

- admin user protected by a password

Configuration **preview** :

PTP Bridge, W60G Bridge:

Bridged Ethernet with a wireless interface. The default IP address 192.168.88.1/24 is set on the bridge interface. There are two possible options - CPE and AP. For CPE wireless interface is set in "station-bridge" mode, and for AP "bridge" mode is used. W60G Bridge - This configuration type is applied to routers that have a 60 GHz point-to-point link.

PTP Bridge:

* Wireless and LAN interfaces are bridged;

LAN Configuration:

Login

- admin user protected by a password

Configuration **preview** :

List of routers using this type of configuration:

- DynaDish - as CPE

W60G Bridge:

* W60G and LAN interfaces are bridged;

wlan60-1 Configuration:

- SSID: MikroTik;
- mode:station-bridge;
- password: no;
- The IP address 192.168.88.1/24 is set on the bridge

Login

- admin user protected by a password

Configuration **preview** :

List of routers using this type of configuration:

- Cube, Cube Pro
- nRAY, Dish
- Wireless Wire Dish, nRAY, Cube, Cube Pro
- *Wireless Wire kit*
- *wAP 60G - with level3 license*

WISP Bridge

The configuration is the same as PTP Bridge in AP mode, except that wireless mode is set to ap_bridge for PTMP setups. The router can be accessed directly using a MAC address. If the device is connected to the network with an enabled DHCP server, configured DHCP client configured on the bridge interface will get the IP address, that can be used to access the router.

List of routers using this type of configuration:

- RB 911,912,921,922 - with Level4 license
- Groove A, RB 711 A
- BaseBox, NetBox
- mANTBox, NetMetal
- wAP 60G AP - with level4 license
- LtAP
- CME

WISP Bridge:

* wireless and LAN interfaces are bridged;

wlan1 Configuration:

- mode: ap-bridge;
- band: 2ghz-b/g/n;
- tx-chains: 0;1;
- rx-chains: 0;1;
- installation: outdoor;
- wpa2: no;
- ht-extension: 20/40mhz-XX;

wlan2 Configuration:

- mode: ap-bridge;
- band: 5ghz-a/n/ac;
- tx-chains: 0;1;
- rx-chains: 0;1;
- installation: outdoor;
- wpa2: no;
- ht-extension: 20/40/80mhz-XXXX;

LAN Configuration:

- DHCP Client: enabled on bridge (LAN port);

Login

- admin user protected by a password

Configuration **preview** :

Switch

This configuration utilizes switch chip features to configure a basic switch. All Ethernet ports are added to switch group and default IP address 192.168.88.1 /24 is set on bridge interface.

List of routers using this type of configuration:

- FiberBox
- CRS without wireless interface

Switch mode:

- All interfaces switched;

Login

- admin user protected by a password

Configuration **preview** :

IP Only

When no specific configuration is found, IP address 192.168.88.1/24 is set on ether1, or combo1, or sfp1.

List of routers using this type of configuration:

- RB 411,433,435,493,800,M11,M33,1100
- CCR

LAN:

- IP on etherx: 192.168.88.1/24;

Login

- admin no password.

Configuration **preview** :

CAP

This type of configuration is used when a device needs to be used as a wireless client device controlled by CAPsMAN.

When CAP default configuration is loaded, ether1 is considered a management port with DHCP client configured. All other Ethernet interfaces are bridged and wlan1 is set to be managed by CAPsMAN.

To load CAP configuration refer to Reset Button manual.

Console

Overview

The console is used for accessing the MikroTik Router's configuration and management features using text terminals, either remotely using a serial port, telnet, SSH, or console screen within Winbox, or directly using a monitor and keyboard. The console is also used for writing scripts. This manual describes the general console operation principles. Please consult the Scripting Manual on some advanced console commands and on how to write scripts.

Hierarchy

The console allows the configuration of the router's settings using text commands. Since there are a lot of available commands, they are split into groups organized in a way of hierarchical menu levels. The name of a menu level reflects the configuration information accessible in the relevant section, eg. **/ip hotspot**.

Example

For example, you can issue the **/ip route print** command:

```
[admin@MikroTik] > ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 # DST-ADDRESS PREF-SRC G GATEWAY DIS INTE...
0 A S 0.0.0.0/0 r 10.0.3.1 1 bridge1
1 ADC 1.0.1.0/24 1.0.1.1 0 bridge1
2 ADC 1.0.2.0/24 1.0.2.1 0 ether3
3 ADC 10.0.3.0/24 10.0.3.144 0 bridge1
4 ADC 10.10.10.0/24 10.10.10.1 0 wlan1
[admin@MikroTik] >
```

Instead of typing /**ip route** path before each command, the path can be typed only once to move into this particular branch of the menu hierarchy. Thus, the example above could also be executed like this:

```
[admin@MikroTik] > ip route
[admin@MikroTik] ip route> print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
 B - blackhole, U - unreachable, P - prohibit #
DST-ADDRESS PREF-SRC G GATEWAY DIS INTE...
0 A S 0.0.0.0/0 r 10.0.3.1 1 bridge1
1 ADC 1.0.1.0/24 1.0.1.1 0 bridge1
2 ADC 1.0.2.0/24 1.0.2.1 0 ether3
3 ADC 10.0.3.0/24 10.0.3.144 0 bridge1
4 ADC 10.10.10.0/24 10.10.10.1 0 wlan1 [
admin@MikroTik] ip route>
```

Notice that the prompt changes to reflect where you are located in the menu hierarchy at the moment. To move to the top level again, type " **/** "

```
[admin@MikroTik] > ip route
[admin@MikroTik] ip route> /
[admin@MikroTik] >
```

To move up one command level, type " **..** "

```
[admin@MikroTik] ip route> ..
[admin@MikroTik] ip>
```

You can also use **/** and **..** to execute commands from other menu levels without changing the current level:

```
[admin@MikroTik] ip route> /ping 10.0.0.1
10.0.0.1 ping timeout
2 packets transmitted, 0 packets received, 100% packet loss
[admin@MikroTik] ip firewall nat> .. service-port print
Flags: X - disabled, I - invalid
# NAME PORTS
0 ftp 21
1 tftp 69
2 irc 6667
3 h323
4 sip
5 pptp
[admin@MikroTik] ip firewall nat>
```

Item Names and Numbers

Many of the command levels operate with arrays of items: interfaces, routes, users, etc. Such arrays are displayed in similarly looking lists. All items in the list have an item number followed by flags and parameter values.

To change the properties of an item, you have to use the **set** command and specify the name or number of the item.

Item Names

Some lists have items with specific names assigned to each of them. Examples are **interface** or **user** levels. There you can use item names instead of item numbers.

You do not have to use the **print** command before accessing items by their names, which, as opposed to numbers, are not assigned by the console internally, but are properties of the items. Thus, they would not change on their own. However, there are all kinds of obscure situations possible when several users are changing the router's configuration at the same time. Generally, item names are more "stable" than the numbers, and also more informative, so you should prefer them to numbers when writing console scripts.

Item Numbers

Item numbers are assigned by the print command and are not constant - two successive print commands may order items differently. But the results of the last print commands are memorized and, thus, once assigned, item numbers can be used even after **add**, **remove,** and **move** operations. Item numbers are assigned on a per-session basis, they will remain the same until you quit the console or until the next print command is executed. Also, numbers are assigned separately for every item list, so for example, the **ip address print** will not change the numbering of the interface list.

It is possible to use item numbers without running the **print** command. Numbers will be assigned just as if the **print** command was executed.

You can specify multiple items as targets for some commands. Almost everywhere, where you can write the number of item, you can also write a list of numbers.

```
[admin@MikroTik] > interface print
Flags: X - disabled, D - dynamic, R - running
# NAME TYPE MTU
0 R ether1 ether 1500
1 R ether2 ether 1500
2 R ether3 ether 1500
3 R ether4 ether 1500
[admin@MikroTik] > interface set 0,1,2 mtu=1460
[admin@MikroTik] > interface print
 Flags: X - disabled, D - dynamic, R - running
# NAME TYPE MTU
0 R ether1 ether 1460
1 R ether2 ether 1460
2 R ether3 ether 1460
3 R ether4 ether 1500
[admin@MikroTik] >
```

Warning: Do not use Item numbers in scripts, it is not a reliable way to edit items in the **scheduler. scripts**, etc. Instead, use the "find" command. More info here also look at scripting examples.

Quick Typing

Two features in the console help entering commands much quicker and easier - the **[Tab]** key completions, and abbreviations of command names. Completions work similarly to the bash shell in UNIX. If you press the **[Tab]** key after a part of a word, the console tries to find the command within the current context that begins with this word. If there is only one match, it is automatically appended, followed by a space:

*/inte***[Tab]**_ becomes **/interface _**

If there is more than one match, but they all have a common beginning, which is longer than that of what you have typed, then the word is completed to this common part, and no space is appended:

*/interface set e***[Tab]**_ becomes **/interface set ether_**

If you've typed just the common part, pressing the tab key once has no effect. However, pressing it for the second time shows all possible completions in compact form:

```
[admin@MikroTik] > interface set e[Tab]_
[admin@MikroTik] > interface set ether[Tab]_
[admin@MikroTik] > interface set ether[Tab]_
ether1 ether5
[admin@MikroTik] > interface set ether_
```

The **[Tab]** key can be used almost in any context where the console might have a clue about possible values - command names, argument names, arguments that have only several possible values (like names of items in some lists or name of protocol in firewall and NAT rules). You cannot complete numbers, IP addresses, and similar values.

Another way to press fewer keys while typing is to abbreviate command and argument names. You can type only the beginning of the command name, and, if it is not ambiguous, the console will accept it as a full name. So typing:

```
[admin@MikroTik] > pin 10.1 c 3 si 100
```

equals to:

```
[admin@MikroTik] > ping 10.0.0.1 count 3 size 100
```

It is possible to complete not only the beginning, but also any distinctive substring of a name: if there is no exact match, the console starts looking for words that have string being completed as first letters of a multiple-word name, or that simply contain letters of this string in the same order. If a single such word is found, it is completed at the cursor position. For example:

```
[admin@MikroTik] > interface x[TAB][TAB]_
dot1x  vxlan  export
[admin@MikroTik] > interface mt[TAB]_
[admin@MikroTik] > interface monitor-traffic _
```

General Commands

Some commands are common to nearly all menu levels, namely: **print**, **set**, **remove**, **add**, **find**, **get**, **export**, **enable**, **disable**, **comment**, and **move**. These commands have similar behavior throughout different menu levels.

- **add** - this command usually has all the same arguments as **set**, except the item number argument. It adds a new item with the values you have specified, usually at the end of the item list, in places where the order of items is relevant. There are some required properties that you have to supply, such as the interface for a new address, while other properties are set to defaults unless you explicitly specify them.
  - Common Parameters
    - *copy-from* - Copies an existing item. It takes the default values of a new item's properties from another item. If you do not want to make an exact copy, you can specify new values for some properties. When copying items that have names, you will usually have to give a new name to a copy
    - *place-before* - places a new item before an existing item with a specified position. Thus, you do not need to use the move command after adding an item to the list.
    - *disabled* - controls disabled/enabled state of the newly added item(-s)
    - *comment* - holds the description of a newly created item
  - Return Values
    - add command returns the internal number of item it has added
- **edit** - this command is associated with the **set** command. It can be used to edit values of properties that contain a large amount of text, such as scripts, but it works with all editable properties. Depending on the capabilities of the terminal, either a fullscreen editor or a single-line editor is launched to edit the value of the specified property. The edit field for console scripts is limited to 30 thousand characters.
- **find** - The find command has the same arguments as set, plus the flag arguments like *disabled* or *active* that take values *yes* or *no* depending on the value of the respective flag. To see all flags and their names, look at the top of the **print** command's output. The **find** command returns internal numbers of all items that have the same values of arguments as specified.
- **move** - changes the order of items in the list.
  - Parameters
    - first argument specifies the item(-s) being moved.
    - the second argument specifies the item before which to place all items being moved (they are placed at the end of the list if the second argument is omitted).
- **print** - shows all information that's accessible from a particular command level. Thus, **/system clock print** shows the system date and time, **/ip route print** shows all routes, etc. If there's a list of items in the current level and they are not read-only, i.e. you can change/remove them (an example of a read-only item list is /system history, which shows a history of executed actions), then print command also assigns numbers that are used by all commands that operate with items in this list.
  - Common Parameters
    - *from* - show only specified items, in the same order in which they are given.
    - *where* - show only items that match specified criteria. The syntax of *where* the property is similar to the **find** command.
    - *brief* - forces the print command to use tabular output form
    - *detail* - forces the print command to use property=value output form
    - *count-only* - shows the number of items
    - *file* - prints the contents of the specific submenu into a file on the router.
    - *interval* - updates the output from the *print* command for every interval of seconds.
    - *oid* - prints the OID value for properties that are accessible from SNMP
    - *without-paging* - prints the output without stopping after each screenful.
- **remove** - removes specified item(-s) from a list.
- **set** - allows you to change values of general parameters or item parameters. The set command has arguments with names corresponding to values you can change. Use "F1" or double [Tab] to see a list of all arguments. If there is a list of items in this command level, then "set" has one action argument that accepts the number of item (or list of numbers) you wish to set up. This command does not return anything.
- **reset** - reset parameters to default values

You can combine commands, here are two variants of the same command that will place a new firewall filter entry, by looking up the comment:

/ip firewall/filter/add chain=forward place-before=[find where comment=CommentX]
/ip/firewall/filter/add chain=forward place-before="CommentX"

Modes

The console line editor works either in multiline mode or in single-line mode. In multiline mode line editor displays the complete input line, even if it is longer than a single terminal line. It also uses a full-screen editor for editing large text values, such as scripts. In single-line mode, only one terminal line is used for line editing, and long lines are shown truncated around the cursor. A full-screen editor is not used in this mode.

The choice of modes depends on detected terminal capabilities.

List of keys

```
F1              Give the list of available commands
command F1      Give help on the command and list of arguments

[Tab]           Complete the command/word. If the input is ambiguous,
                a second [Tab] gives possible options

F3 or Ctrl-R    Search command history
F4 or Ctrl-X    Toggle safe mode
F5 or Ctrl-L    Repaint the screen
F7              Toggle hotlock mode
Ctrl-\          Split line
Home or Ctrl-A  Go to the beginning of the line
End or Ctrl-E   Go to the end of the line
Ctrl-C          Interrupt current action
Ctrl-D          Terminate session (on empty prompt)
Ctrl-K          Delete to the end of the line
Ctrl-U          Delete to the beginning of the line

/               Move up to base level
..              Move up one level
/command        Use command at the base level
```

up, **down** and **split** keys leave the cursor at the end of the line.

Built-in Help

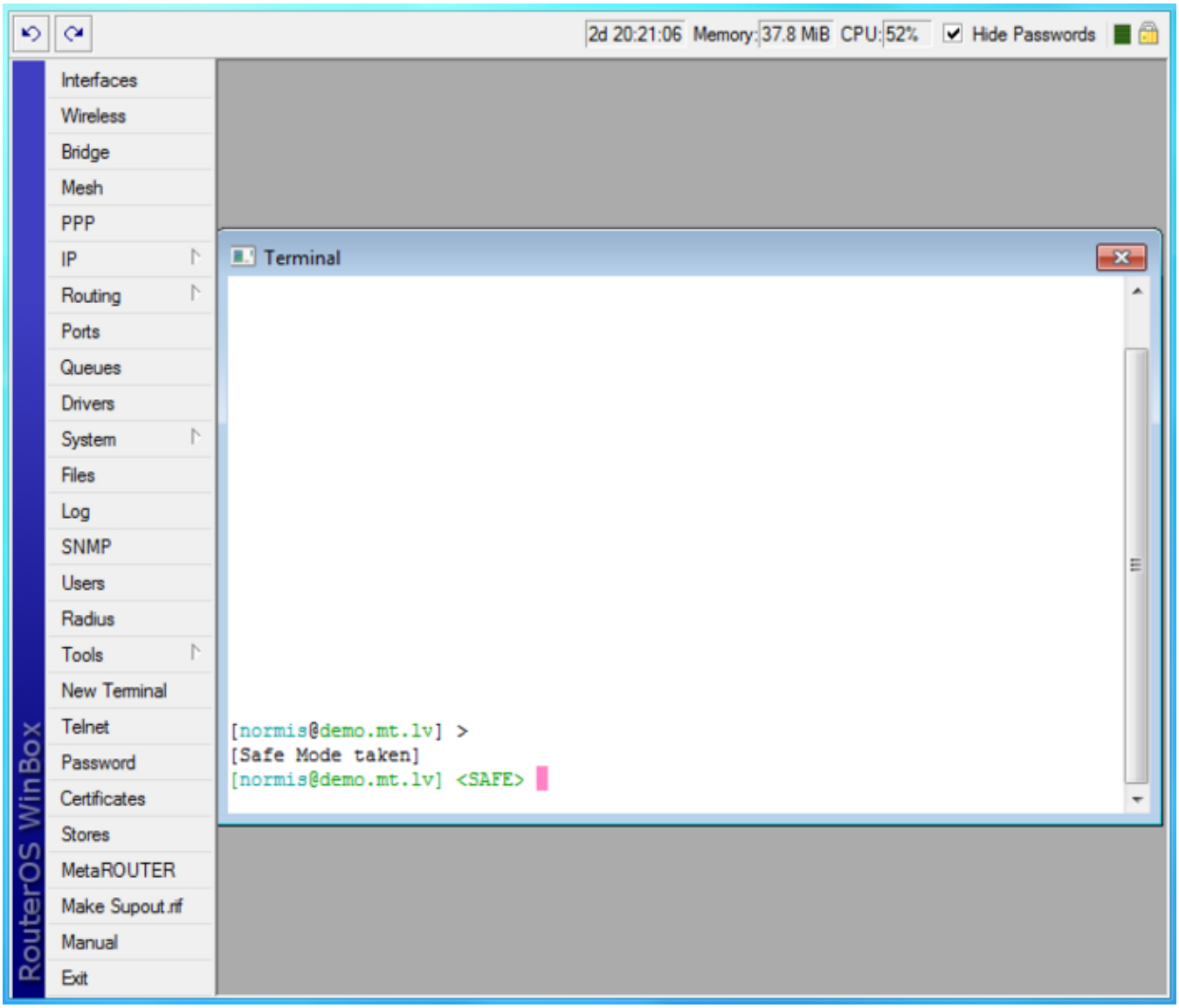
The console has built-in help. Press **F1** for general console usage Help. The general rule is that Help shows what you can type in a position where the **F1** w as pressed (similarly to pressing **[Tab]** key twice, but in verbose form and with explanations).

Safe Mode

It is sometimes possible to change router configuration in a way that will make the router inaccessible (except from the local console). Usually, this is done by accident, but there is no way to undo the last change when the connection to the router is already cut. Safe mode can be used to minimize such risk.

The **"Safe Mode"** button in the Winbox GUI allows you to enter Safe Mode, while in the CLI, you can access it by either using the keyboard shortcut **F4** or pressing **[CTRL]+[X]**. To exit without saving the made changes in CLI, hit **[CTRL]+[D].**

```
[admin@MikroTik] ip route>[CTRL]+[X]
[Safe Mode taken]
[admin@MikroTik] ip route<SAFE>
```



Message **Safe Mode taken** is displayed and prompt changes to reflect that session is now in safe mode. All configuration changes that are made (also from other login sessions), while the router is in safe mode, are automatically undone if the safe mode session terminates abnormally. You can see all such changes that will be automatically undone and tagged with an **F** flag in the system history:

```
[admin@MikroTik] /ip/route>
[Safe Mode taken]
[admin@MikroTik] /ip/route<SAFE> add
[admin@MikroTik] /ip/route<SAFE> /system/history/print
Flags: U, F - FLOATING-UNDO
Columns: ACTION, BY, POLICY
  ACTION                BY     POLICY
F route 0.0.0.0/0 added  admin  write
```

Now, if the telnet connection (or WinBox terminal) is cut, then after a while (TCP timeout is **9** minutes) all changes that were made while in safe mode will be undone. Exiting session by **[Ctrl]+[D]** also undoes all safe mode changes, while **/quit** does not.

If another user tries to enter safe mode, he's given the following message:

```
[admin@MikroTik] >
Hijacking Safe Mode from someone - unroll/release/don't take it [u/r/d]:
```

- [u] - undoes all safe mode changes, and puts the current session in safe mode.

- [r] - keeps all current safe mode changes, and puts the current session in a safe mode. The previous owner of safe mode is notified about this:

```
[admin@MikroTik] ip firewall rule input
[Safe mode released by another user]
```

- [d] - leaves everything as-is.

If too many changes are made while in safe mode, and there's no room in history to hold them all (currently history keeps up to 100 most recent actions), then the session is automatically put out of the safe mode, and no changes are automatically undone. Thus, it is best to change the configuration in small steps, while in safe mode. Pressing **[Ctrl]+[X]** twice is an easy way to empty the safe mode action list.

As "Safe Mode" operates within the user's session and stores configuration changes, it will be ignored for commands requiring a reboot, such as resetting configuration or restoring from a backup.

HotLock Mode

When HotLock mode is enabled commands will be auto-completed.

To enter/exit HotLock mode press **F7**.

```
[admin@MikroTik] /ip/address> [F7]
[admin@MikroTik] /ip/address>>
```

Double>>is an indication that HotLock mode is enabled. For example, if you type "`/in et`", it will be auto-completed to

```
[admin@MikroTik] /ip/address>> /interface/ethernet/
```

Reset Button

Introduction

The RouterOS password can only be reset by reinstalling the RouterOS or using the reset button (or jumper hole) in case the hardware is RouterBOARD. For X86(CHR) devices, only a complete reinstall will clear the password, along with any other configuration. For RouterBOARD devices, several methods exist, depending on the device model.

Reset From RouterOS

If you still have access to your router and want to recover its default configuration, then you can:

- Run the command **"/system reset-configuration"** from a command-line interface;
- Do it from the **System -> Reset Configuration** menu in the graphical user interface;

Using Reset Button

RouterBOARD devices are fitted with a reset button which has several functions:

- **Loading the backup RouterBOOT loader**
  Hold this button before applying power, and release it after three seconds since powering, to load the backup boot loader. This might be necessary if the device is not operating because of a failed RouterBOOT upgrade. When you have started the device with the backup loader, you can either set RouterOS to *force backup loader* in the RouterBOARD settings or have a chance to reinstall the failed RouterBOOT from a ".fwf" file (total of **3 seconds**)
- **Resetting the RouterOS configuration**
  Hold this button until the LED light starts flashing, and release the button to reset RouterOS configuration to default.
- **Enabling CAPs mode**
  To connect this device to a wireless network managed by CAPsMAN, keep holding the button for 5 more seconds, LED turns solid, release now to turn on CAPs mode. It is also possible to enable CAPs mode via the command line, to do so run the command **"/system reset-configuration caps-mode=yes"**;
- **Starting the RouterBOARD in Netinstall mode**
  Or keep holding the button for 5 more seconds until the LED turns off, then release it to make the RouterBOARD look for Netinstall servers. You can also simply keep the button pressed until the device shows up in the Netinstall program on Windows.

You can also do the previous three functions without loading the backup loader, simply push the button immediately after you apply power. You might need the assistance of another person to push the button and also plug in the power supply at the same time!

How to reset configuration

1) Unplug the device from power;

2) Press and hold the button right after applying power;

Note: hold the button until the LED will start flashing;

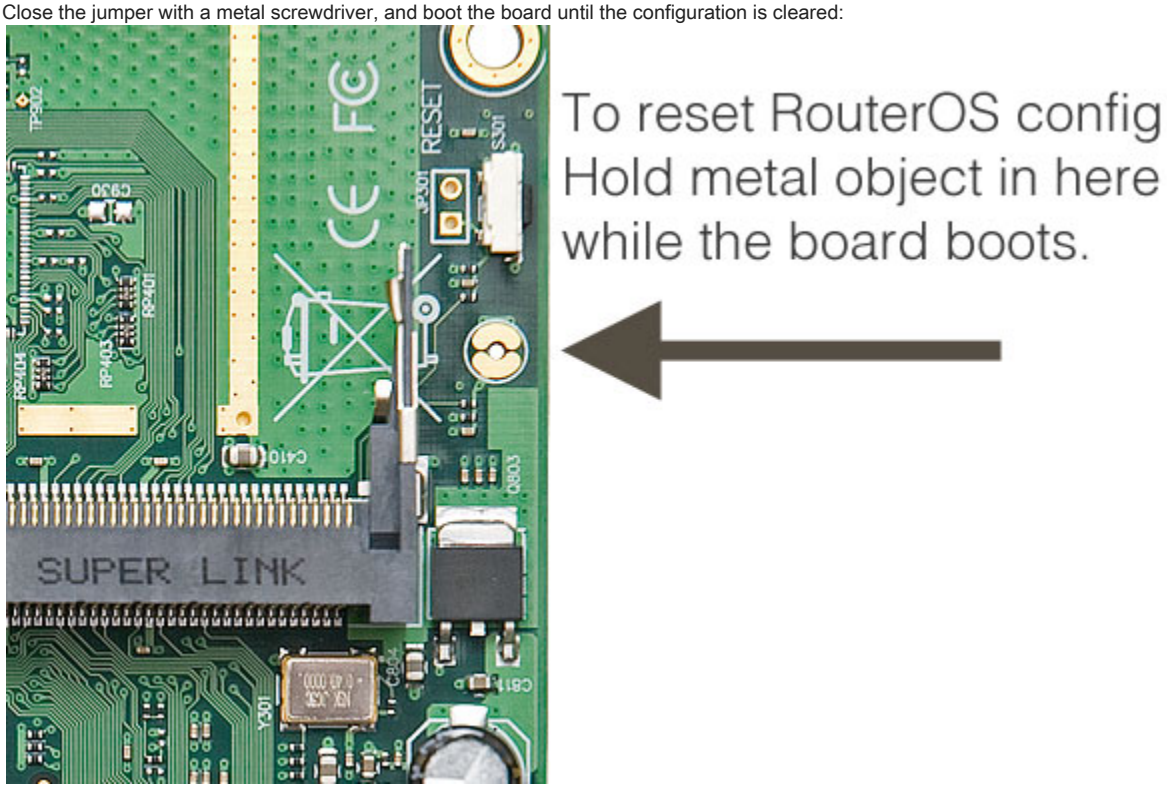
3) Release the button to clear the configuration;

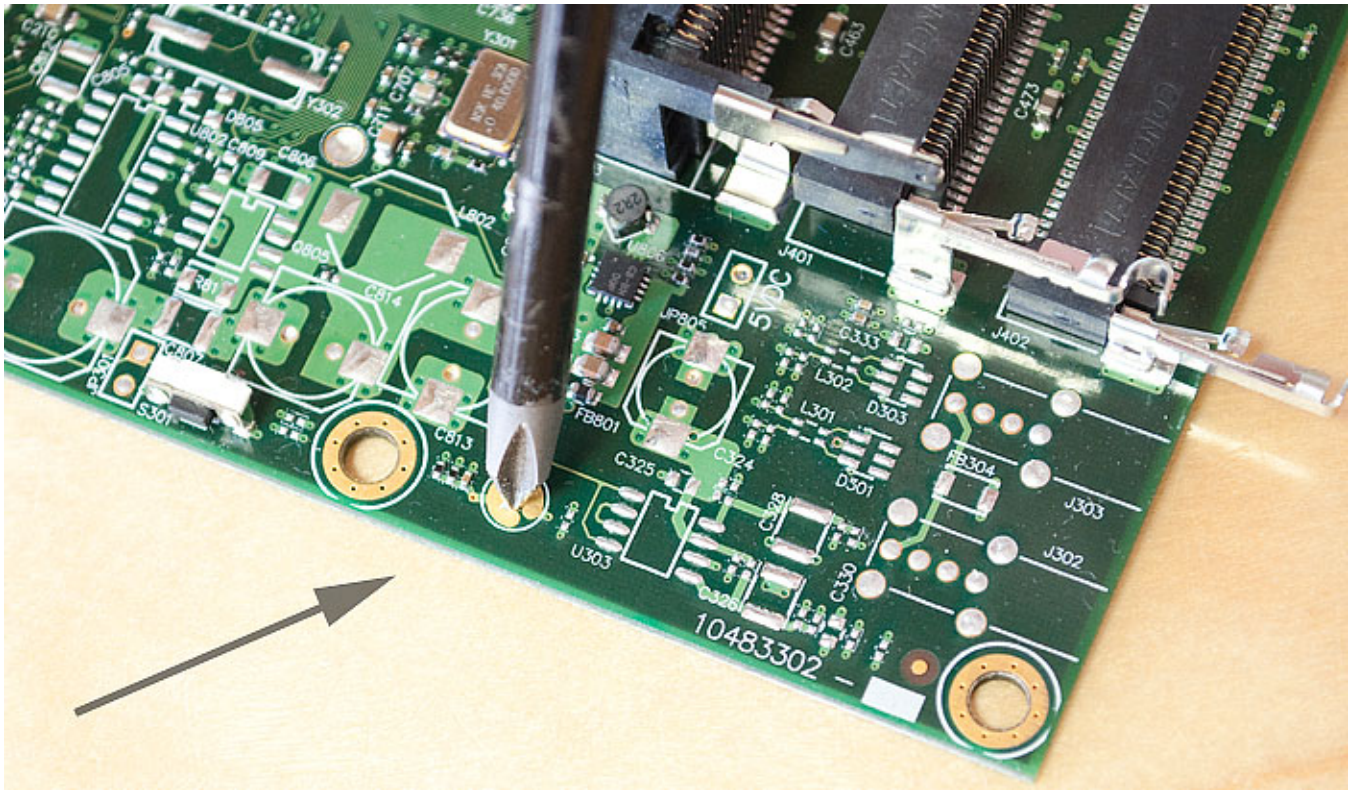
If you wait until the LED stops flashing, and only then release the button - this will instead launch Netinstall mode, to reinstall RouterOS.

Jumper hole reset

Older RouterBOARD models are also fitted with a reset jumper hole. Some devices might need an opening of the enclosure, RB750/RB951/RB751 have the jumper hole under one of the rubber feet of the enclosure.

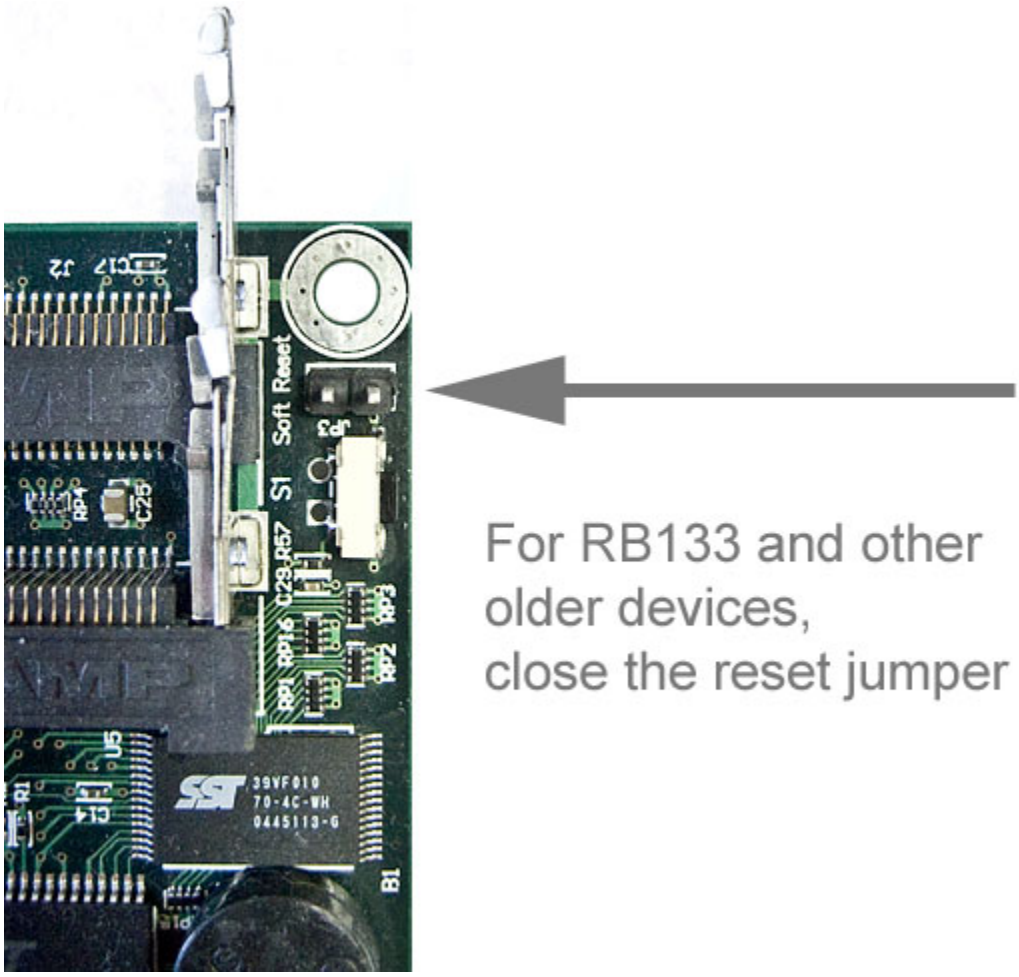
Close the jumper with a metal screwdriver, and boot the board until the configuration is cleared:





Jumper reset for older models

The **below** image shows the location of the Reset Jumper on older RouterBOARDs like RB133C:



Don't forget to remove the jumper after the configuration has been reset, or it will be reset every time you reboot!

Backup

Summary

The RouterOS backup feature allows cloning a router configuration in binary format, which can then be re-applied on the same device. The system's backup file also contains the device's MAC addresses, which are restored when the backup file is loaded.

We recommend restoring the backup on the same version of RouterOS.

If The Dude or User-manager or installed on the router, then the system backup will not contain configuration from these services, therefore, additional care should be taken to save configuration from these services. Use the provided tool mechanisms to save/export configuration if you want to save it.

System backups contain sensitive information about your device and its configuration, always consider encrypting the backup file and keeping the backup file in a safe place.

Saving a backup

Sub-menu: `/system backup save`

Property	Description
dont-encrypt (*yes* / *no*; Default: **no**)	Disable backup file encryption. Note that since RouterOS v6.43 without a provided password, the backup file is unencrypted.
encryption (*aes-sha256* / *rc4*; Default: **aes-sha256**)	The encryption algorithm to use for encrypting the backup file. Note that is not considered a secure encryption method and is only available for compatibility reasons with older RouterOS versions.
name (*string*; Default: **[identity]-[date]-[time].backup**)	The filename for the backup file.
password (*string*; Default:)	Password for the encrypted backup file. Note that since RouterOS v6.43 without a provided password, the backup file is unencrypted.

If a password is not provided in RouterOS versions older than v6.43, then the backup file will be encrypted with the current user's password, except if the *dont-encrypted* property is used or the current user's password is empty.

The backup file will be available under `/file` menu, which can be downloaded using FTP or using Winbox.

Loading a backup

Load units backup without password:

```
[admin@MikroTik] > system/backup/load name=auto-before-reset.backup password=""
```

Property	Description
name (*string*; Default:)	File name for the backup file.
password (*string*; Default:)	Password for the encrypted backup file.

Example

To save the router's configuration to file test and a password:

```
[admin@MikroTik] > /system backup save name=test password=<YOUR_PASSWORD>
Configuration backup saved
[admin@MikroTik] > /system backup
```

To see the files stored on the router:

```
[admin@MikroTik] > /file print
# NAME TYPE SIZE CREATION-TIME
0 test.backup backup 12567 sep/08/2018 21:07:50
[admin@MikroTik] >
```

To load the saved backup file test:

```
[admin@MikroTik] > /system backup load name=test
password: <YOUR_PASSWORD>
Restore and reboot? [y/N]: y
Restoring system configuration
System configuration restored, rebooting now
```

Cloud backup

Since RouterOS v6.44 it is possible to securely store your device's backup file on MikroTik's Cloud servers, read more about this feature on the IP/Cloud page.

Netinstall

Introduction

Netinstall is a tool for installing and reinstalling MikroTik devices running RouterOS. Always try using Netinstall if you suspect that your device is not working properly. The tool is available for Windows (with a graphical interface) and for Linux (as a command line tool).

In short, the Netinstall procedure goes like this: Connect your PC directly to the **boot** port (Usually Ether1, the port labeled BOOT or as otherwise indicated in the product manual) of the device you will be reinstalling. Turn on the device while holding the **reset** button until it shows up in the Netinstall tool.

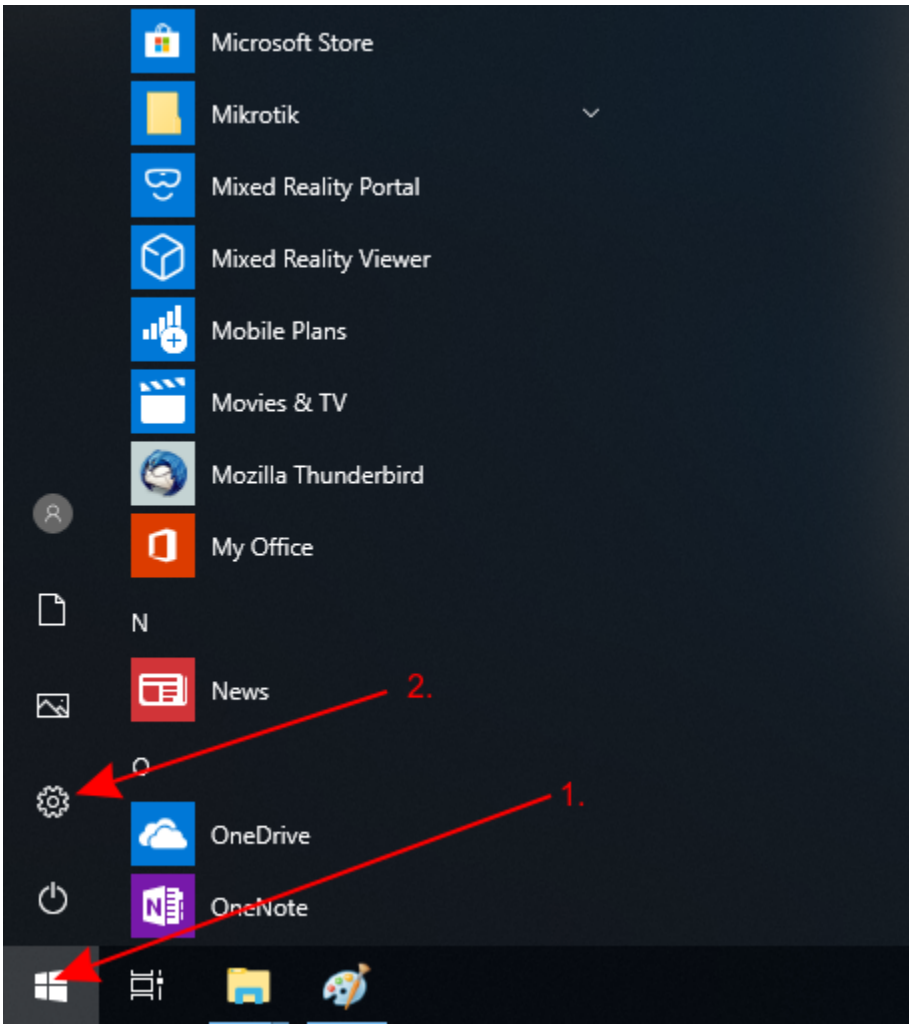
Careful. Netinstall re-formats the system's drive, all configuration and saved files will be lost. Netinstall does not erase the RouterOS license key, nor does it reset RouterBOOT related settings, for example, CPU frequency is not changed after reinstalling the device.

Instructions for Windows

- Download **Netinstall** from the downloads page. If you are not sure which version you need, then you can always select the version that is marked as **Current** (stable);
- Download the RouterOS **Main package** from the downloads page;

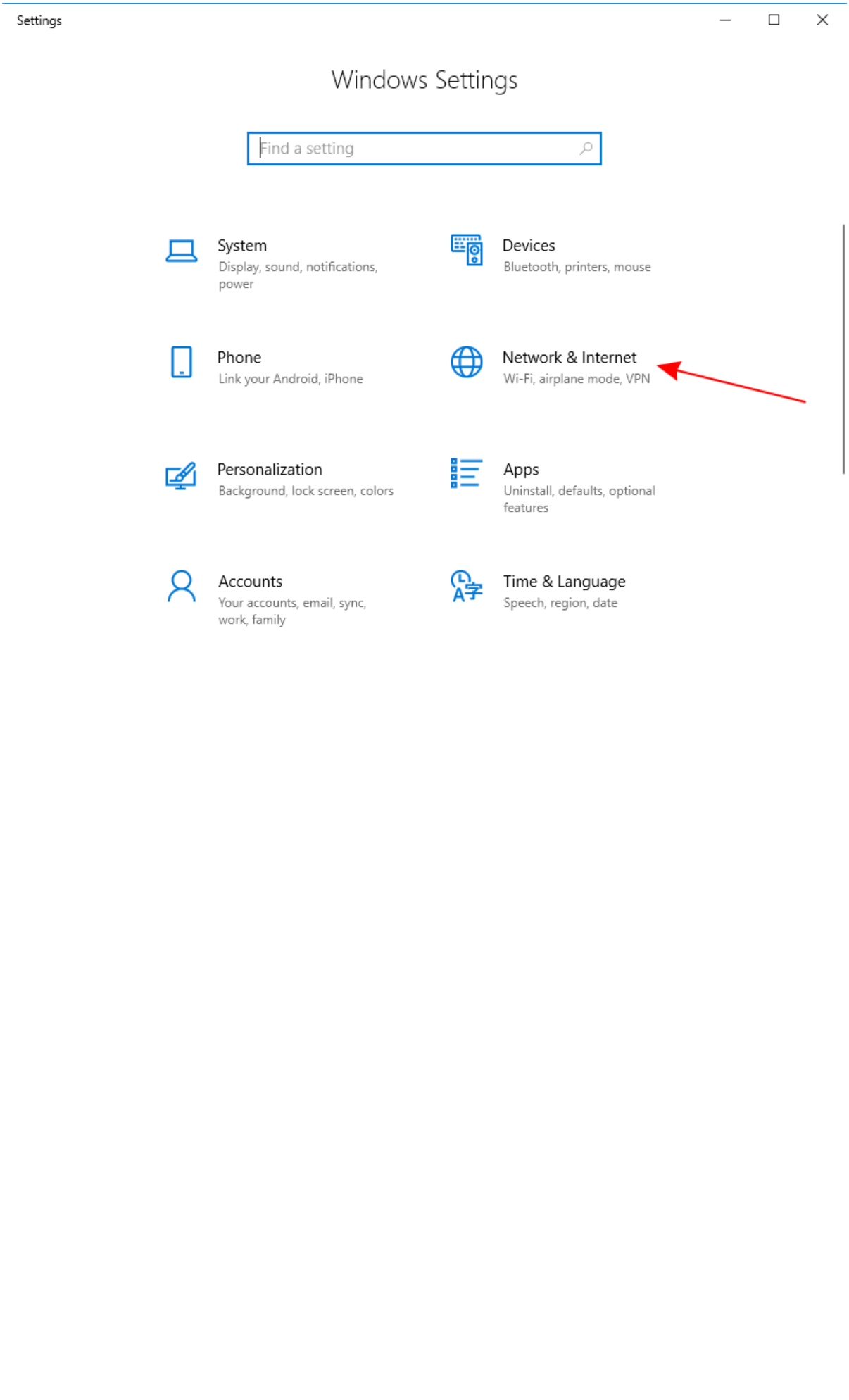
  You must choose a RouterOS version. You can always select the version that is marked as **Current**. You must also select the architecture (ARM, MIPS, SMIPS, TILE, etc...), but if you are not sure, then you can download the RouterOS package for **ALL** architect ures, Netinstall will choose the right architecture for you.

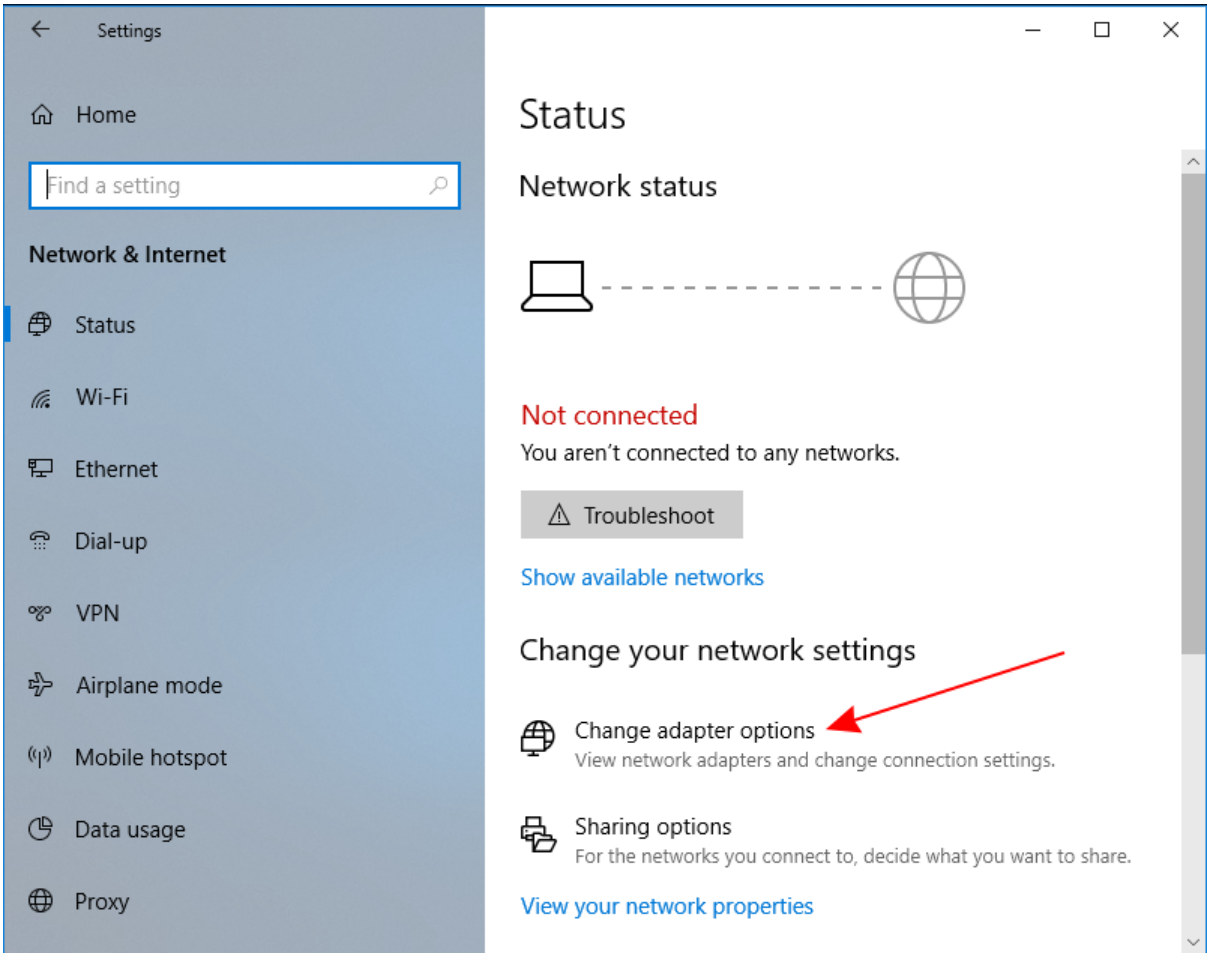
- Disable all of your computer network interfaces (WiFi, Ethernet, LTE, or any other type of connection) besides the one which will be used for installation! Netinstall will only work on one active interface on your computer, it is highly recommended that you disable any other network interfaces in order to be sure that Netinstall will select the right network interface.
- Configure a static IP address for your Ethernet interface, open **Start,** and select **Settings**:



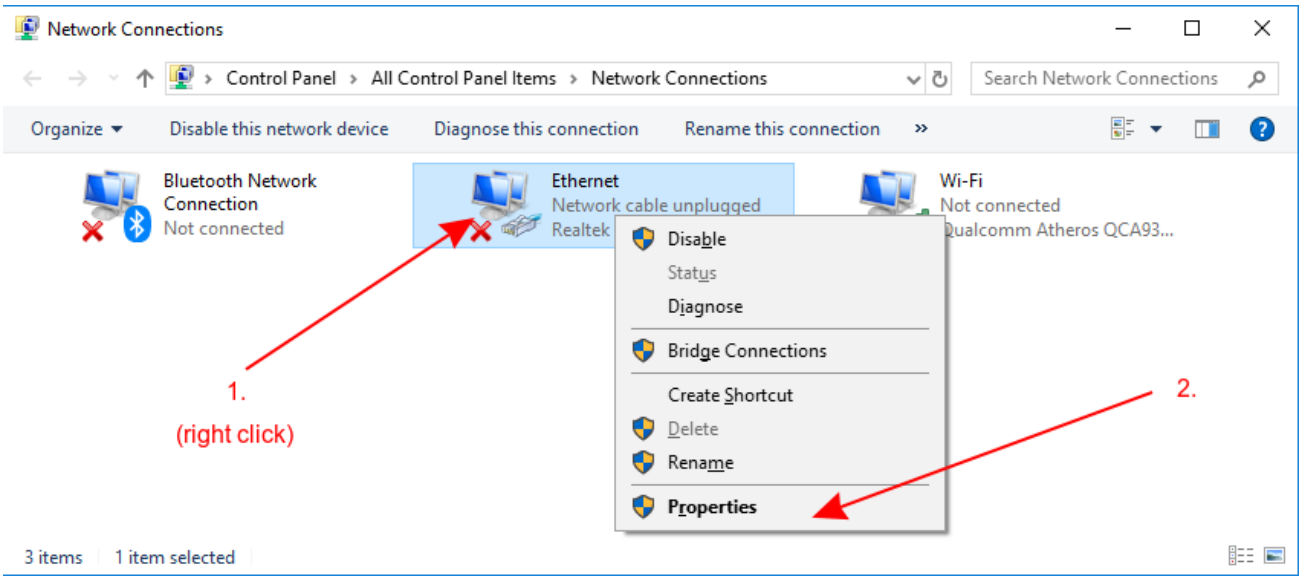
Netinstall can run also on a local network, in such case you could skip setting a static IP address, but it is highly recommended that you set a static IP address if you are not familiar with Netinstall.

- Open **Network & Internet** and select **Change adapter options**

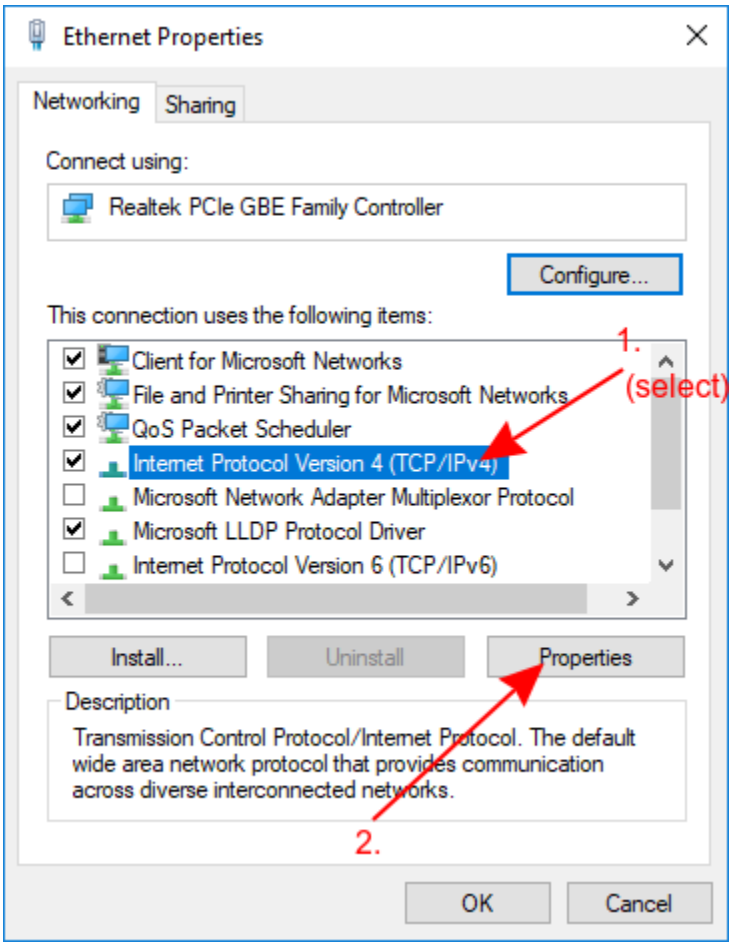

Settings
Windows Settings
Find a setting
System
Display, sound, notifications, power
Devices
Bluetooth, printers, mouse
Phone
Link your Android, iPhone
Network & Internet
Wi-Fi, airplane mode, VPN
Personalization
Background, lock screen, colors
Apps
Uninstall, defaults, optional features
Accounts
Your accounts, email, sync, work, family
Time & Language
Speech, region, date



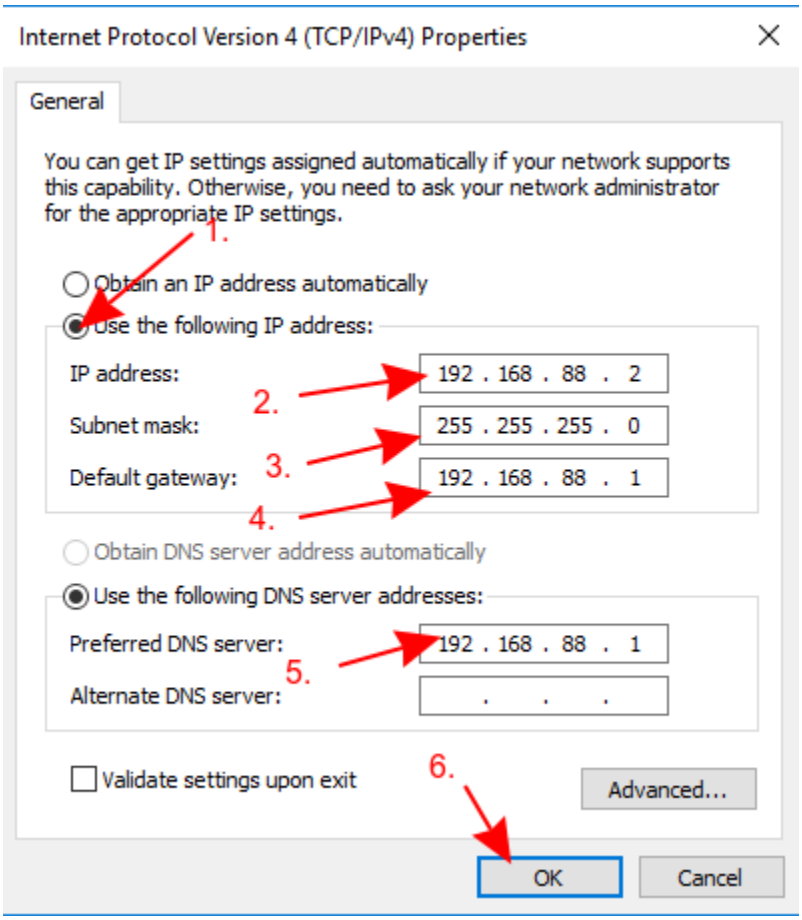
- Right-click on your Ethernet interface and select **Properties**



- Select **Internet Protocol Version 4 (TCP/IPv4)** and click **Properties**

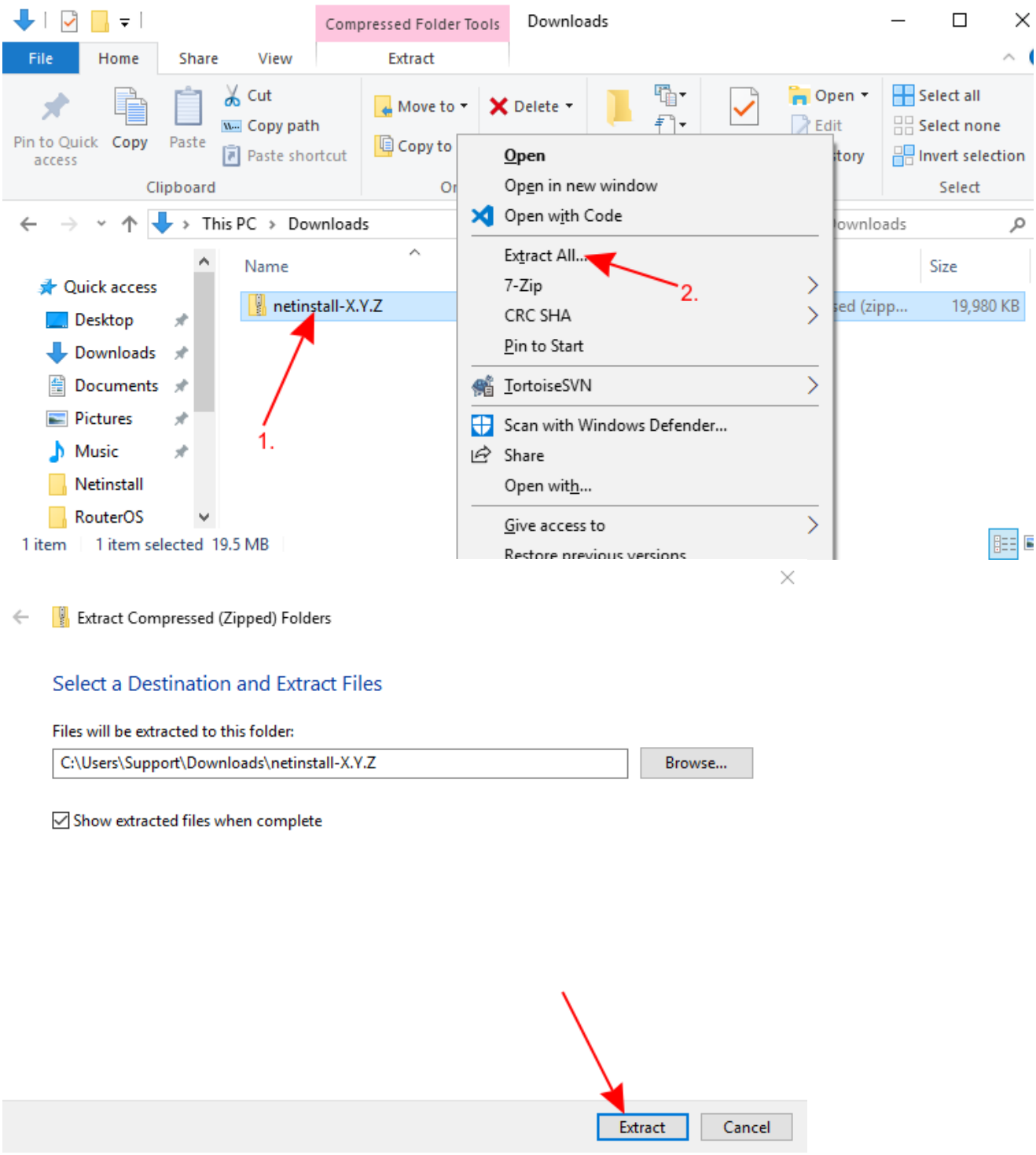


- Check Use the following IP address and fill out the fields as shown in the image below

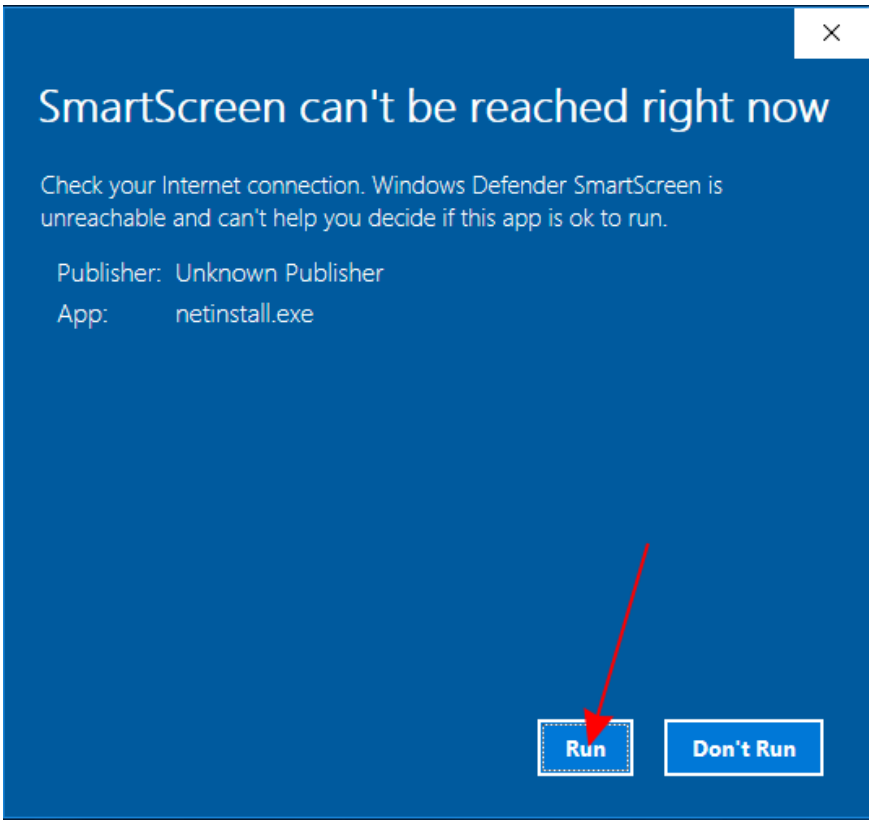


If you have a working router, then you can use it and skip the setting up a static IP part of this tutorial, but it requires you to know your LAN address since you will need to specify an unused IP address in your network for the network boot server. For this reason, it is recommended to apply a static IP address and follow this guide precisely, if you are not sure how to get these parameters out of your network.

- Open your Downloads folder (or wherever you saved the downloaded files) and extract the Netinstall .zip file to a convenient place

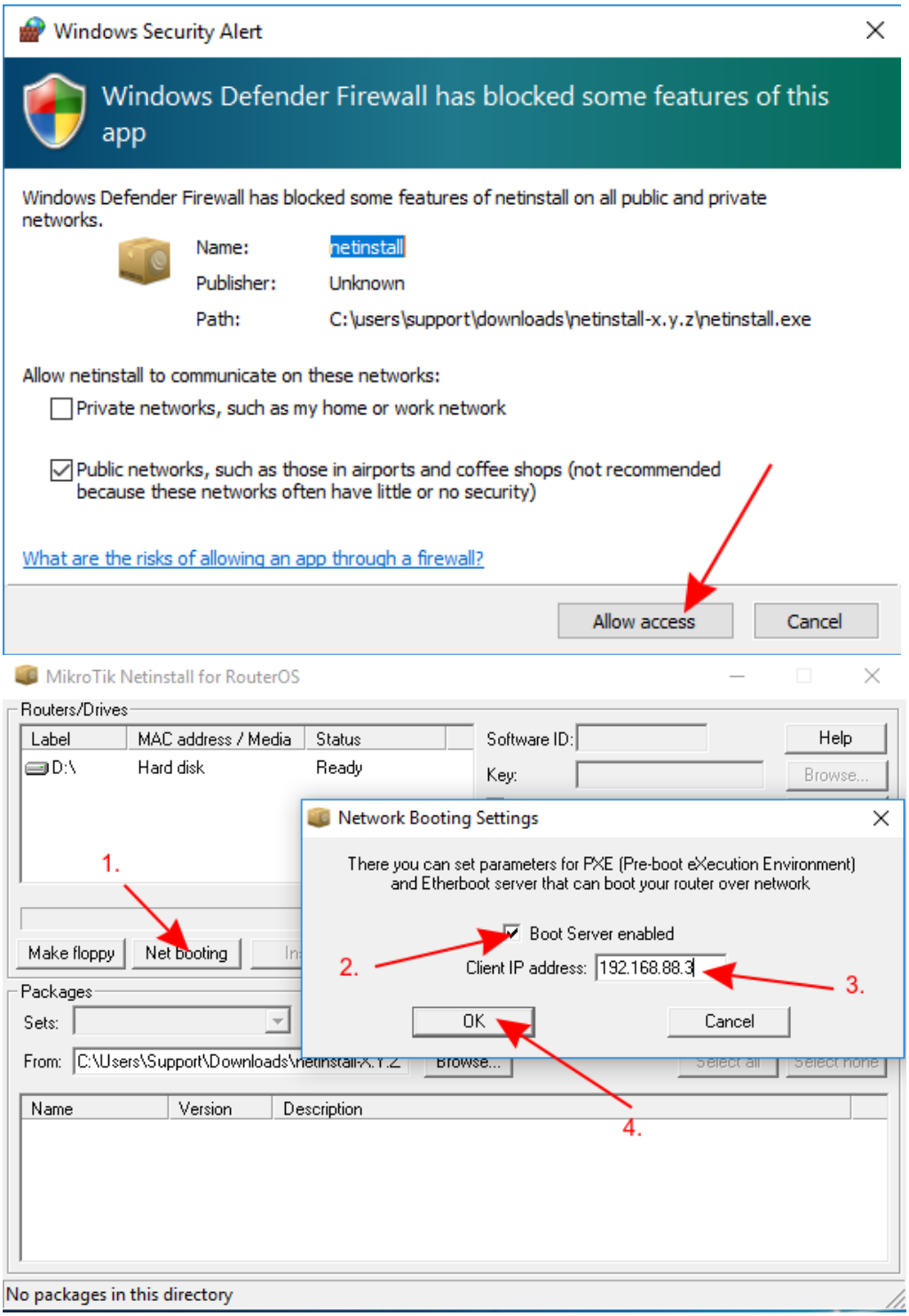


- Make sure that the Ethernet interface is running and launch Netinstall.exe. If you followed the guide precisely, then you should not have any Internet connection on your computer, Windows 10 wants to verify all apps that it runs, but will not be able to do it since lack of an Internet connection, for this reason, a warning might pop up, you should click **Run**.



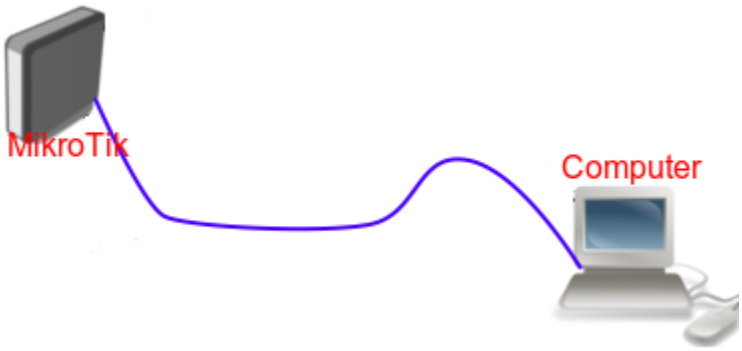
Netinstall requires administrator rights, there should be a window asking for permissions to run Netinstall, you must accept these permissions in order for Netinstall to work properly.

- Allow access for Netinstall in **Public** networks and configure **Net booting** settings and fill out the required fields as shown in the image below



The Client IP address must be unique! Don't use an existing IP address in your network, this also means that you should not use the computer's IP address as well. Use a completely different IP address from the same subnet.

- Connect your device to your computer using an ethernet cable directly (without any other devices in-between), plug the Ethernet cable into your device's Etherboot port.
- MikroTik devices are able to use Netinstall from their **first** port (Ether1), or from the port marked with "**BOOT**".



Some computers have a network interface (especially USB Ethernet adapters) that tend to create an extra link flap, which is enough for Netinstall to fail to detect a device that is in Etherboot mode. In such a case you can use a switch between your device and your computer or a router in bridge mode to prevent this issue.

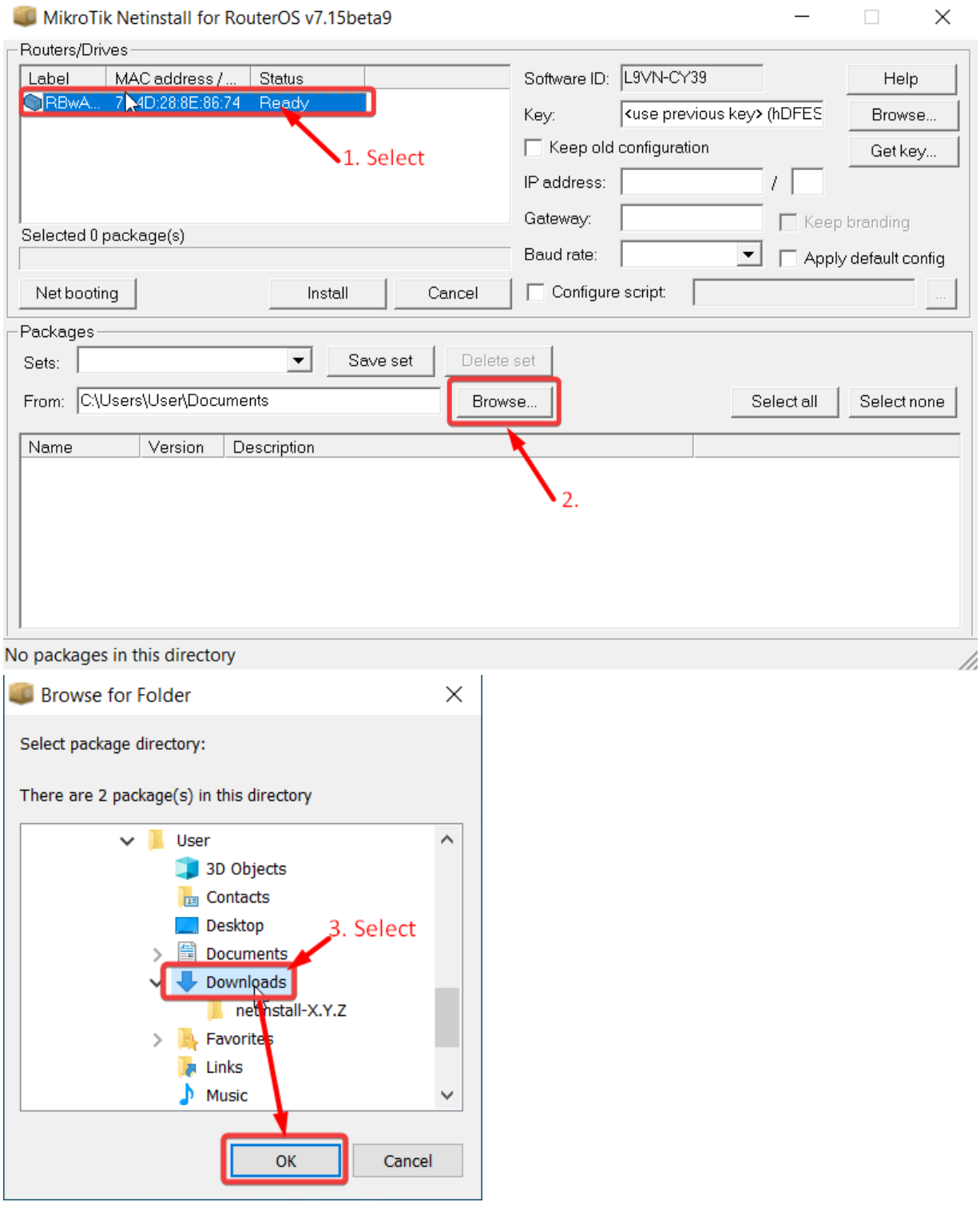
Netinstall uses bootp packets, which are using the same port numbers as DHCP packets. If you're using a switch between your PC and the device to be Netinstalled, ensure that the ports in the bridge are not blocked by other network devices.

If you have dhcp-snooping enabled, make sure to enable "trusted" on the bridge ports facing the Netinstall PC.

- Power up your device and put it into etherboot mode

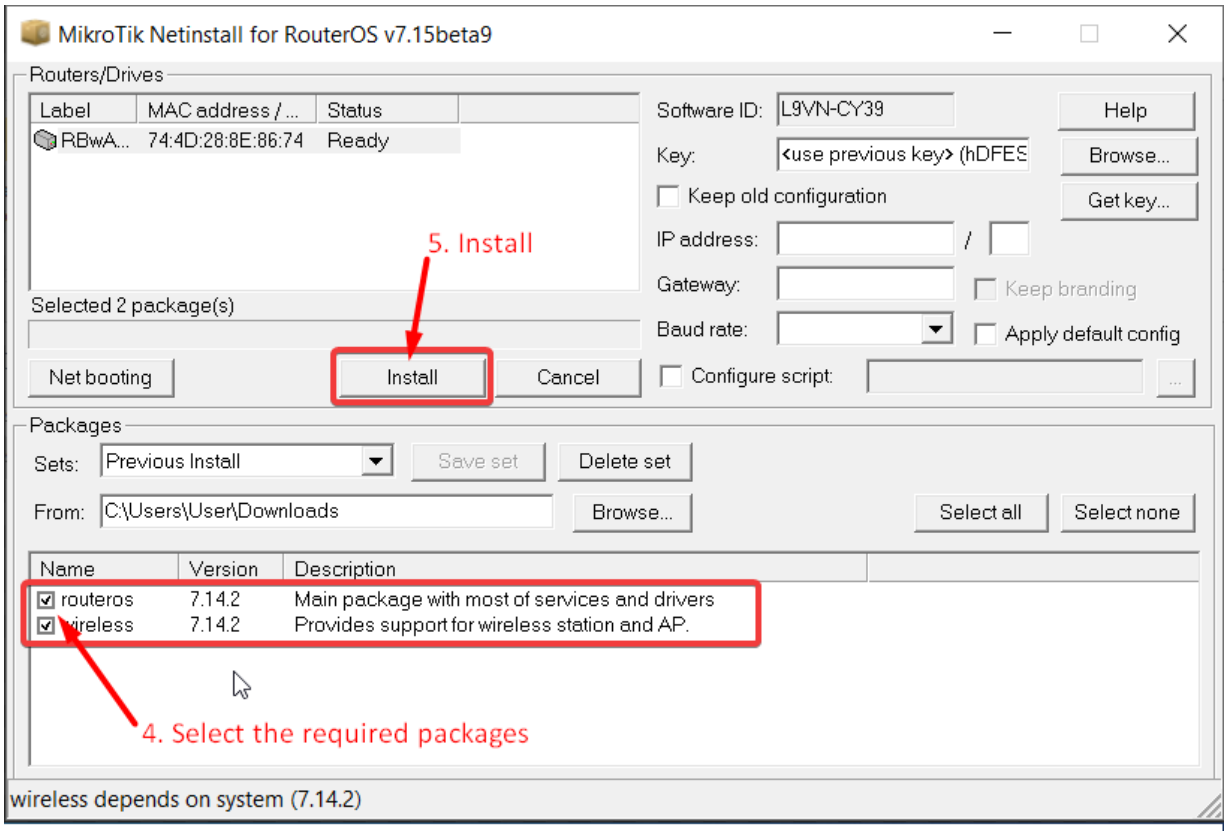
There are multiple ways how to put your device into Etherboot mode. Make sure you read the Etherboot manual before trying to put the device into this mode. Methods vary between different MikroTik devices.

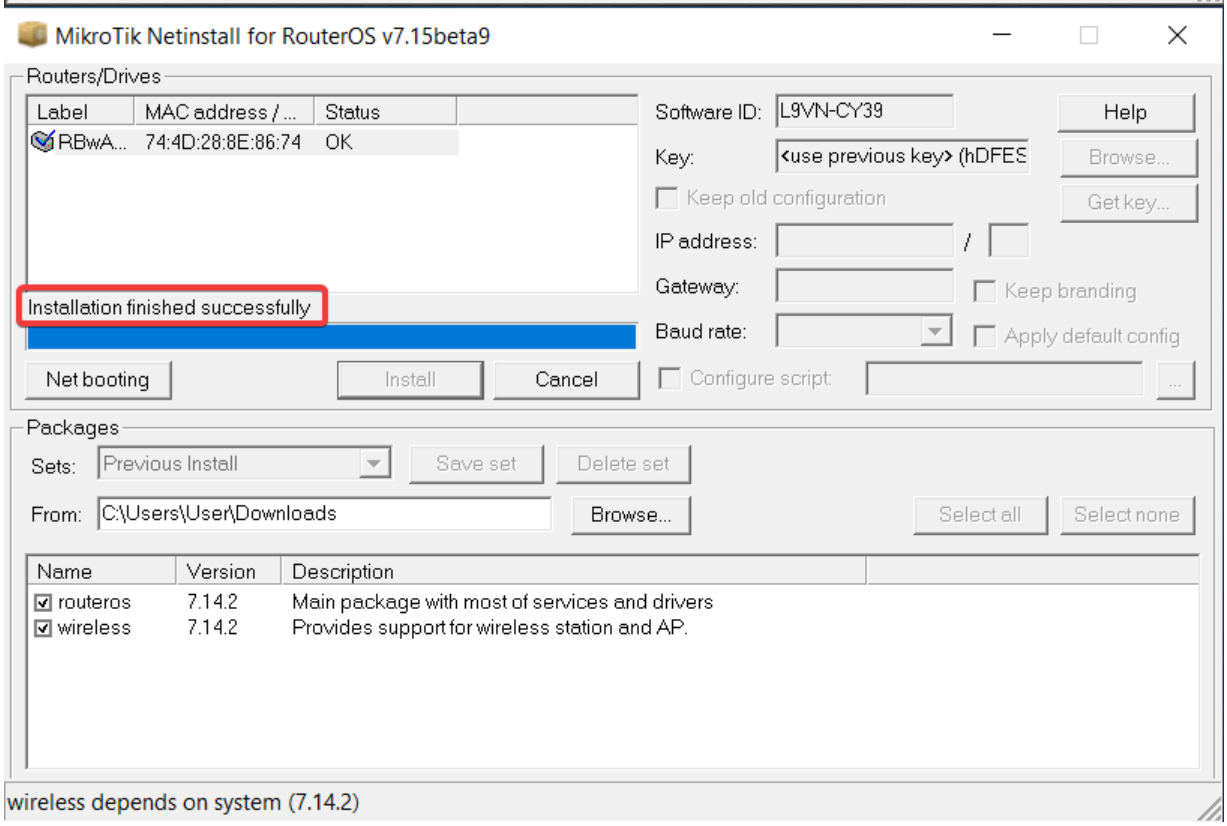
- Wait for the device to show up in Netinstall, then select it and click **Browse.** Navigate to your **Downloads** folder (or wherever you saved your RouterOS packages) and press **OK.**



- Select your desired RouterOS packages and press **Install.** Wait for the installation to finish and press "**Reboot**" (Devices without serial console have to be rebooted manually).

> ⚠ If you have downloaded RouterOS packages for multiple architectures, Netinstall will only display the appropriate architecture packages for your device after you have selected it. Unsupported packages will not appear in this window once a device is selected.



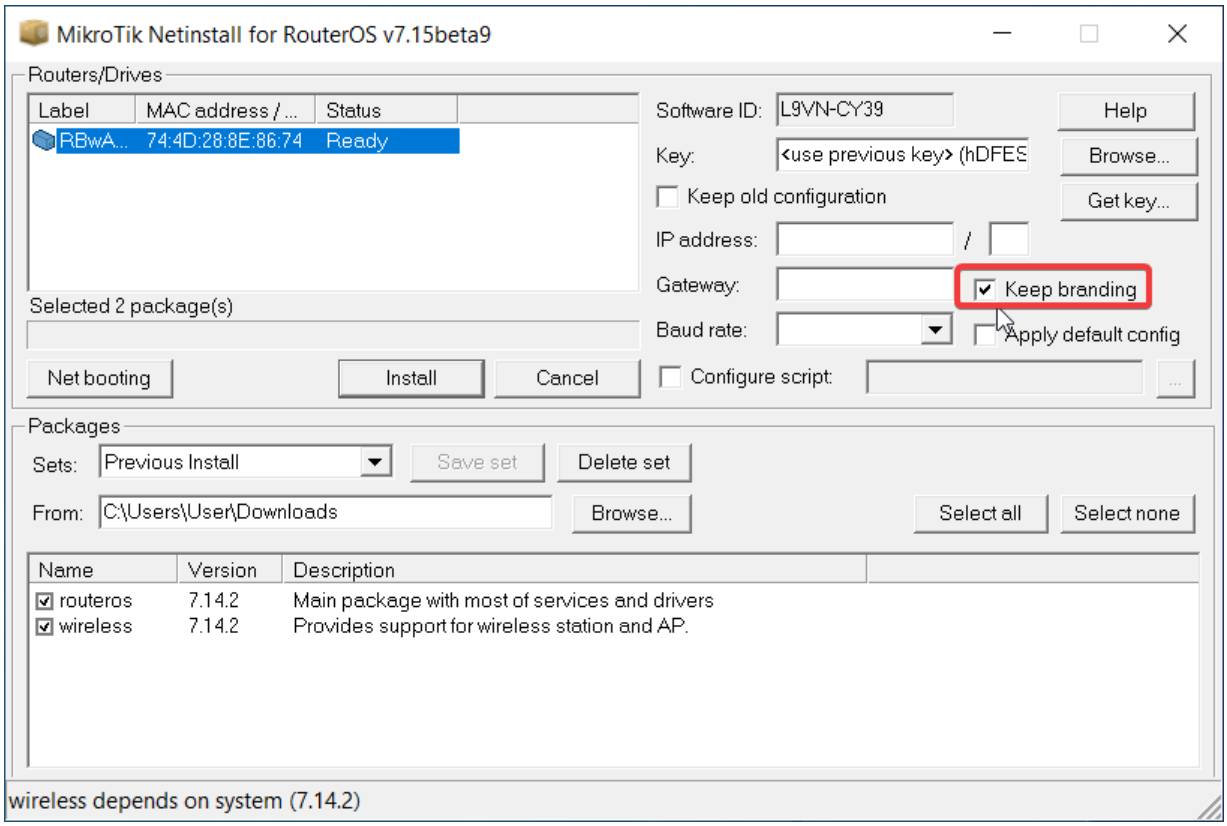


If the installation does not start (progress bar is not moving or no status is shown), then you can try closing the Netinstall application and opening it up again or try to put the device into **Etherboot** mode again. If you are still unable to get Netinstall working, then you should try using it on a different computer since there might be an operating system's issue that is preventing Netinstall from working properly.

- You are done! Remove the device from power, remove the Ethernet cable, place the device back in your network and your device should be running properly now!

After using Netinstall the device will be reset to defaults (unless you specified not to apply default configuration). Some devices are not accessible through **ether1** port with the default configuration for security reasons. Read more about Default configuration.

Option **"Keep branding"** allows you to retain the device's already installed branding package without reinstalling it using Netinstall.



The **"Keep old configuration"** process involves downloading the configuration database from the router, reinstalling the router (including disk formatting), and uploading the configuration files back to it. However, it's important to note that this process solely applies to the configuration itself and does not impact the files, including databases like the User Manager database, Dude database, and others.

When using the **Configure script** option, it is suggested to introduce a delay before configuration execution.

Instructions for Linux

The Linux version is a command line tool, which offers nearly the same parameters as the Windows counterpart.

Download the tool from our download page (links not literal):

```
wget https://download.mikrotik.com/routeros/[VERSION]/netinstall-[VERSION].tar.gz
```

Extract it:

```
tar -xzf netinstall-[VERSION].tar.gz
```

Run the tool:

```
./netinstall-cli -a 192.168.0.1 routeros-arm64-[VERSION].npk
```

The tool requires privileged access and must be run as root, use sudo.

The available parameters are as follows:

Parameter	Meaning
-r	When the reinstallation process is performed, the configuration is reset, and for devices that have it, the default configuration will be applied. (optional)
-e	When the reinstallation process is performed reset device to empty configuration.
-b	Option to discard the currently installed branding package from the device, otherwise it will be reinstalled together with RouterOS
-k keyfile	Provides the device with a license key in .KEY format. (optional)
-s userscript	Pre-configures the device with the provided configuration (text file in .RSC format). This configuration also takes place of the default configuration. The script can access factory passwords with read-only variables $defconfPassword and $defconfWifiPassword (starting from RouterOS 7.10beta8). (optional)
-a IP	Uses a specific IP address that the Netinstall server will assign to the device. Mandatory, but can be auto-assigned if interface parameter used.
PACKAGE	Specify a list of RouterOS.NPK format packages that Netinstall will try to install on the device. (mandatory) **The system package must be listed first.**
-i	Allows you to specify an interface. (optional)

Note

If the "-r" or "e-" parameter is not specified, *netinstall-cli* will reinstall RouterOS and will keep the current configuration by downloading current configuration database from the router, reinstalling the router (including disk formatting), and uploading the configuration back to it, the same as Netinstall **"Keep old configuration"** option. However, it's important to note that this process solely applies to the configuration itself and does not impact the files, including databases like the User Manager database, Dude database, and others.

First make sure you have set the IP on your computer's interface:

```
admin@ubuntu:~$ sudo ifconfig <interface> 192.168.88.2/24
```

Then run the Netinstall version 6 (an example that resets the configuration upon reinstallation procedure):

```
admin@ubuntu:~$ sudo ./netinstall -r -a 192.168.88.3 routeros-mipsbe-6.48.1.npk
Using server IP: 192.168.88.2
Starting PXE server
Waiting for RouterBOARD...
PXE client: 01:23:45:67:89:10
Sending image: mips
Discovered RouterBOARD...
Formatting...
Sending package routeros-mipsbe-6.48.1.npk ...
Ready for reboot...
Sent reboot command
```

Or run the Netinstall version 7 (an example that applies an empty configuration and discards the branding during the reinstallation procedure):

```
admin@ubuntu:~$ sudo ./netinstall-cli -e -b -i enx1234567ee890 -a 192.168.88.3 routeros-7.14.2-arm.npk wireless-7.14.2-arm.npk
Version: 7.15beta9(2024-03-27 20:41:15)
Will apply empty config
Will remove branding
Using Interface: enx1234567ee890
Wait for Link-UP on 'enx1234567ee890'. OK
Using Client IP: 192.168.88.3
Using Server IP: 192.168.88.10
Starting PXE server
Waiting for RouterBOARD...
client: 74:4D:28:8E:86:74
Detected client architecture: arm
Sending and starting Netinstall boot image ...
Installed branding package detected
Discovered RouterBOARD... 74:4D:28:8E:86:74
Formatting...
Sending package routeros-7.14.2-arm.npk ...
Sending package wireless-7.14.2-arm.npk ...
Sending empty config ...
Ready for reboot...
Sent reboot command
```

Etherboot

Etherboot mode is a special state for a MikroTik device that allows you to reinstall your device using Netinstall. There are two types of booters available for use: the regular booter and the backup booter. It's essential to verify both options.

- To use the Regular booter press Ctrl+E to enter etherboot mode using the serial console or press the Reset button after a 1-2 second delay from when you power it on.
- To employ the backup booter, power OFF the device. Press the Reset button and power on your device (wait until the "USR" led is blinking then stable "On", and when the "USR" led is "Off" - release the Reset button) - the device is booting in bootp mode to reinstall RouterOS using Netinstall.

Reset button

The **Reset** can be found on all MikroTik devices, this button can be used to put the device into Etherboot mode. An easy way to put a device into Etherboot mode using the **Reset** button is by powering off the device, hold the **Reset** button, power on the device while holding the **Reset** button and keep holding it until the device shows up in your **Netinstalll** window.

If you have set up a Protected bootloader, then the reset button's behavior is changed. Make sure you remember the settings you used to set up the Protected bootloader, otherwise you will not be able to use Eterboot mode and will not be able to reset your device.

RouterOS

If your device is able to boot up and you are able to log in, then you can easily put the device into Etherboot mode. To do so, just connect to your device and execute the following command:

```
/system routerboard settings set boot-device=try-ethernet-once-then-nand
```

After that either reboot the device or do a power cycle on the device. Next time the device will boot up, then it will first try going into Etherboot mode. Note that after the first boot up, the device will not try going into Etherboot mode and will boot directly from NAND or from the storage type the device is using.

Serial console

Some devices come with a serial console that can be used to put the device into Etherboot mode. To do so, make sure you configure your computer's serial console. The required parameters for all MikroTik devices (except for RouterBOARD 230 series) are as following:

```
115200bit/s, 8 data bits, 1 stop bit, no parity, flow control=none by default.
```

For RouterBOARD 230 series devices the parameters are as following:

```
9600bit/s, 8 data bits, 1 stop bit, no parity, hardware (RTS/CTS) flow control by default.
```

Make sure you are using a proper null modem cable, you can find the proper pinout here. When the device is booting up, keep pressing **CTRL+E** on your keyboard until the device shows that it is **trying bootp protocol**:

```
RouterBOOT booter 7.14.2

CRS328-4C-20S-4S+

CPU frequency: 800 MHz
  Memory size: 512 MiB
 Storage size:  16 MiB

Press Ctrl+E to enter etherboot mode
Press any key within 2 seconds to enter setup
trying bootp protocol.... OK
Got IP address: 192.168.88.3
resolved mac address 84:69:93:9E:E6:49
transfer started ............................. transfer ok, time=2.00s
```

At this point your device is in Etherboot mode, now the device should show up in your Netinstall window.

Supout.rif

What is supout.rif file?

The support file is used for debugging MikroTik RouterOS and to solve the support questions faster. All MikroTik Router information is saved in a binary file, which is stored in the router and can be downloaded from the router using FTP or WinBox. If required, then you can generate the file on the "/flash" folder on devices with FLASH type memory or external storage drive, by specifying the full path to the file "name=flash/supout.rif". You can view the contents of this file in your Mikrotik account, simply click on "Supout.rif viewer" located in the left column and upload the file.

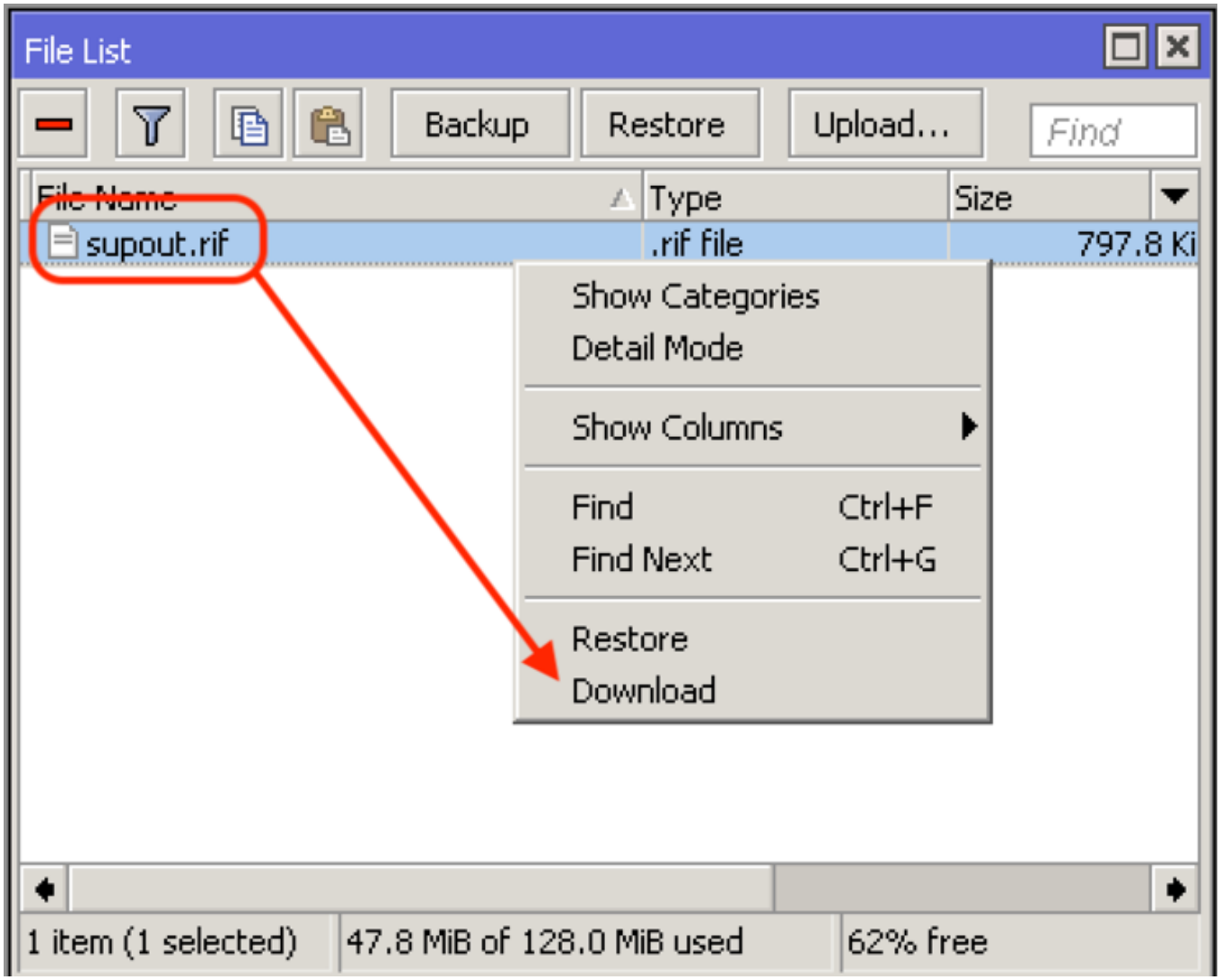
This file contains all your router's configuration, logs, and some other details that will help MikroTik Support to solve your issue. The file does not contain sensitive information or router passwords.

Creating a Support Output file

Winbox

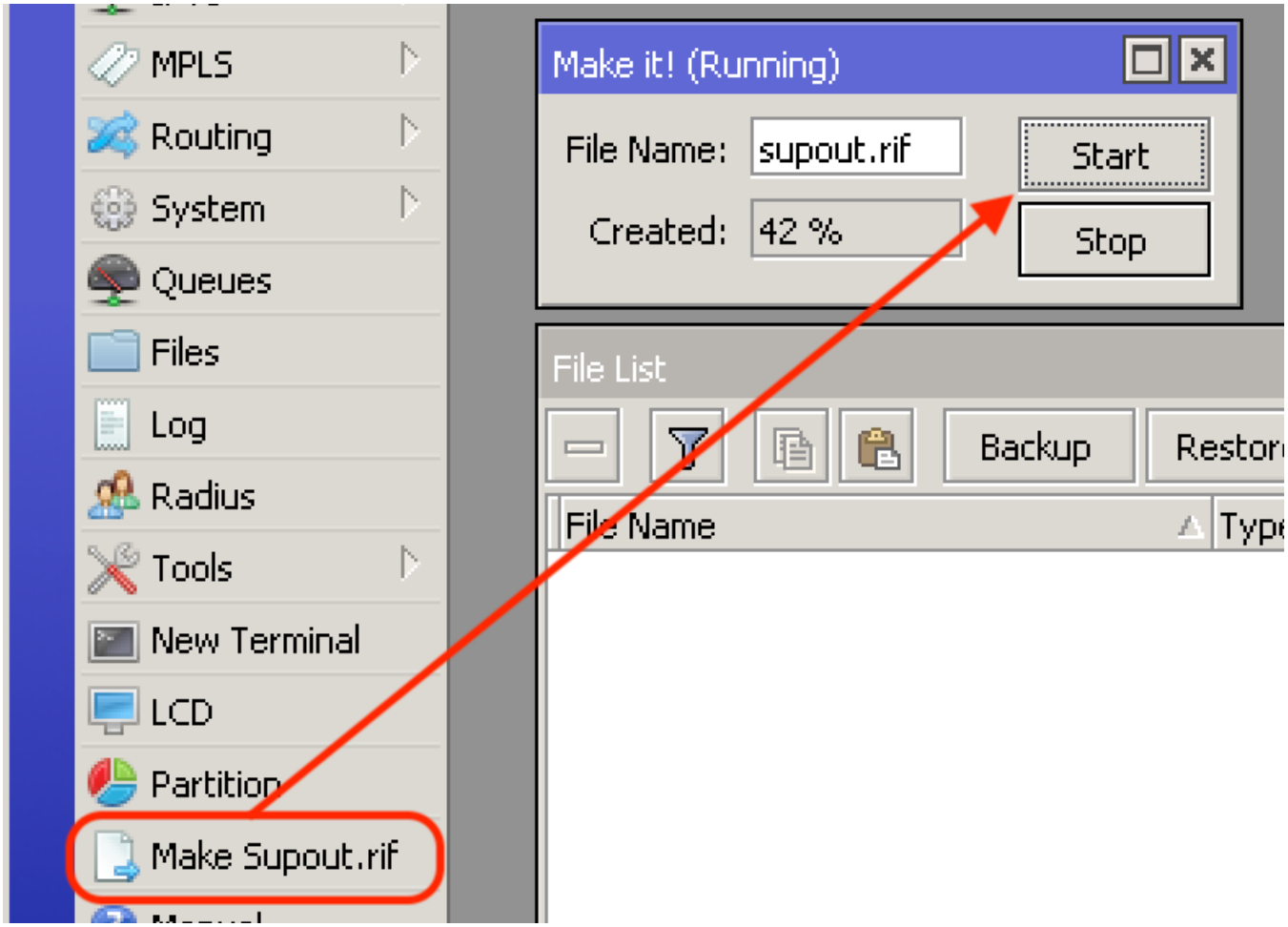
To generate this file in Winbox, click on "Make Supout.rif".

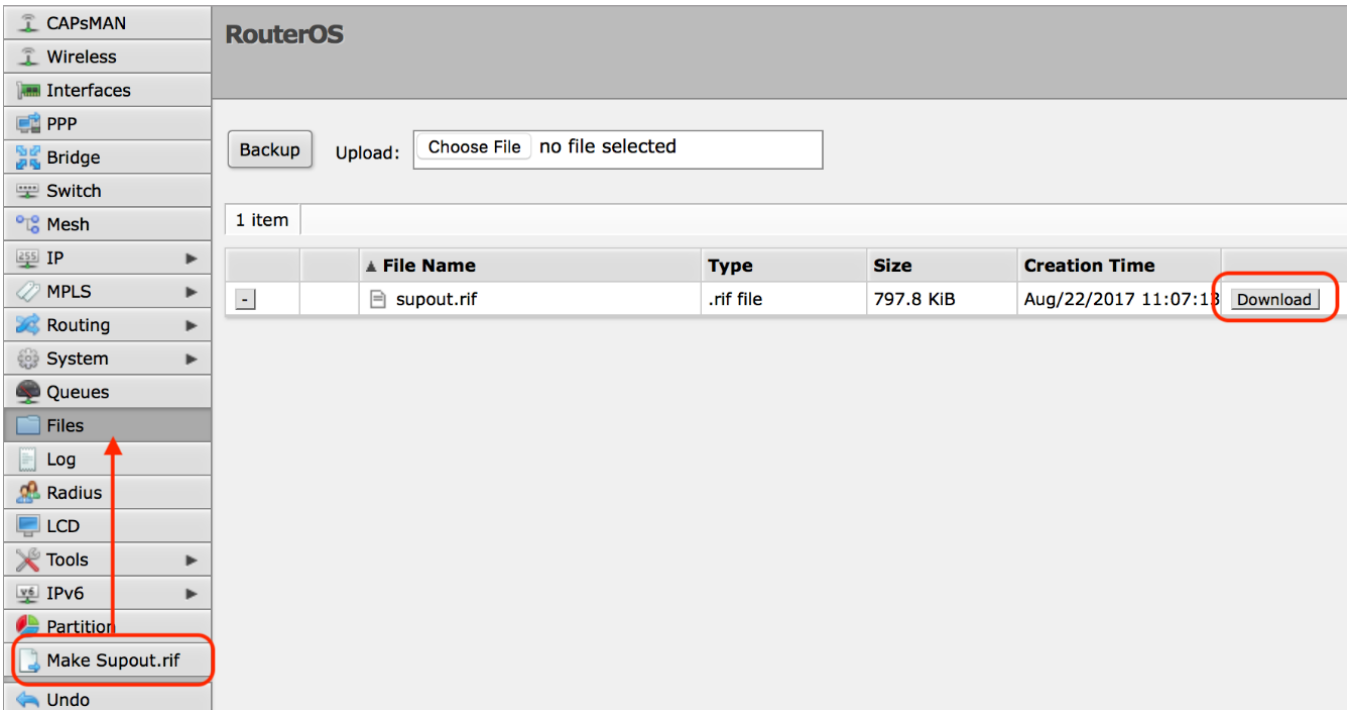
To save the file to your computer, right mouse click on the file and select "Download" to get support output file or simply drag the file to your desktop.



Webfig

To generate this file in Webfig, click on "Make Supout.rif" and then "Download" to get it on your computer.





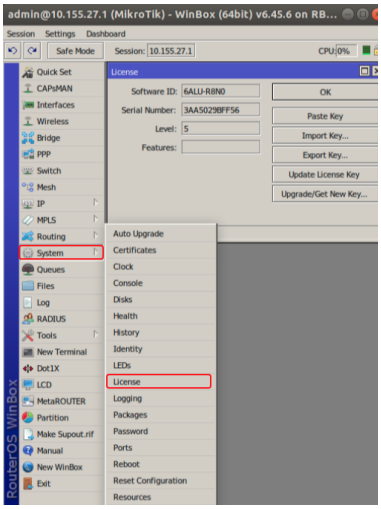
Console

To generate this file, please type in the command line:

```
/system sup-output name=supout.rif
```

RouterOS license keys

Overview



MikroTik hardware routers that run RouterOS come preinstalled with a RouterOS license, if you have purchased a RouterOS based device, nothing must be done regarding the license.

For X86 systems (i.e. PC devices), you need to obtain a license key.

The license key is a block of symbols that needs to be copied from your mikrotik.com account, or from the email you received in, and then it can be pasted into the router. You can paste the key anywhere in the terminal, or by clicking "Paste key" in WinBox License menu. A reboot is required for the key to take effect.

⚠ RouterOS licensing scheme is based on Software-ID / System-ID where:

- RouterBOARD software-id is bound to storage media (HDD, NAND).
- x86 software-id is bound to MBR
- CHR system-id is bound to MBR and UUID

Before the license purchase it is recommended to check if the software ID does not change on reboot. (Software ID may change on defective HDD, on HDD where RAID controllers are used but not properly configured etc.)

Licensing information can be read from CLI system console:

```
[admin@RB1100] > /system license print
    software-id: "43NU-NLT9"
         nlevel: 6
       features:
[admin@RB1100] >
```

or from equivalent WinBox, WebFig menu.

License Levels

After installation RouterOS runs in **trial mode**. You have 24 hours to register for Level1 (Free demo) or purchase a Level 4,5 or 6 license and paste a valid key.

Level 3 is a wireless station (client or CPE) only license. *For x86 PCs, Level3 is not available for purchase individually.*

Level 2 was a transitional license from old legacy (pre 2.8) license format. These licenses are not available any more, if you have this kind of license, it will work, but to upgrade it - you will have to purchase a new license.

The difference between license levels is shown in the table below.

Level number	0 (Trial mode)	1 (Free Demo)	3 (WISP CPE)	4 (WISP)	5 (WISP)	6 (Controller)
Price	no key	registration required	not for sale	$45	$95	$250
Wireless AP mode (PtMP)	24h trial	-	no	yes	yes	yes
PPPoE tunnels	24h trial	1	200	200	500	unlimited
PPTP tunnels	24h trial	1	200	200	500	unlimited
L2TP tunnels	24h trial	1	200	200	500	unlimited
OVPN tunnels	24h trial	1	200	200	unlimited	unlimited
EoIP tunnels	24h trial	1	unlimited	unlimited	unlimited	unlimited
VLAN interfaces	24h trial	1	unlimited	unlimited	unlimited	unlimited

Queue rules	24h trial	1	unlimited	unlimited	unlimited	unlimited
HotSpot active users	24h trial	1	1	200	500	unlimited
User manager active sessions	24h trial	1	10	20	50	Unlimited
Bonding interfaces	24h trial	1	unlimited	unlimited	unlimited	unlimited

All Licenses:

- never expire (a running and licensed router can be used indefinitely)
- can use unlimited number of interfaces
- are for one installation each
- offer unlimited software upgrades (exception - demo license does not allow ROS version upgrade (started from 7.8))

CHR License Levels

License levels described until now do not apply to Cloud Hosted Routers (CHRs). CHR is a RouterOS version intended for running as a virtual machine. It has its own 4 license levels as well as trial where you can test any of the paid license levels for 60 days.

60-day free trial license is available for all paid license levels. To get the free trial license, you have to have an account on MikroTik.com as all license management is done there.

Perpetual is a lifetime license (buy once, use forever). It is possible to transfer a perpetual license to another CHR instance. A running CHR instance will indicate the time when it has to access the account server to renew it's license. If the CHR instance will not be able to renew the license it will behave as if the trial period has ran out and will not allow an upgrade of RouterOS to a newer version.

After licensing a running trial system, you **must** manually run the `/system license renew` command from the CHR to make it active. Otherwise the system will not know you have licensed it in your account. If you do not do this before the system deadline time, the trial will end and you will have to do a complete fresh CHR installation, request a new trial and then license it with the license you had obtained.

License	Speed limit	Price	Description
Free	1Mbit	FREE	The free license level allows CHR to run indefinitely. It is limited to 1Mbps upload per interface. All the rest of the features provided by CHR are available without restrictions. To use this, all you have to do is download disk image file from our download page and create a virtual guest.
P1	1Gbit	$45	P1 (perpetual-1) license level allows CHR to run indefinitely. It is limited to 1Gbps upload per interface. All the rest of the features provided by CHR are available without restrictions. It is possible to upgrade p1 to p10 or p-unlimited After the upgrade is purchased the former license will become available for later use on your account.
P10	10Gbit	$95	P10 (perpetual-10) license level allows CHR to run indefinitely. It is limited to 10Gbps upload per interface. All the rest of the features provided by CHR are available without restrictions. It is possible to upgrade p10 to p-unlimited After the upgrade is purchased the former license will become available for later use on your account.
P-Unlimited	Unlimited	$250	The p-unlimited (perpetual-unlimited) license level allows CHR to run indefinitely. It is the highest tier license and it has no enforced limitations.
60-day Trial		FREE	In addition to the limited Free installation, you can also test the increased speed of P1/P10/PU licenses with a 60 trial. You will have to have an account registered on MikroTik.com. Then you can request the desired license level for trial from your router that will assign your router ID to your account and enable a purchase of the license from your account. All the paid license equivalents are available for trial. A trial period is 60 days from the day of acquisition, after this time passes, your license menu will start to show "Limited upgrades", which means that RouterOS can no longer be upgraded. **Note** that if you plan to purchase the selected license, you must do it ***before*** 60 days trial ends. If your trial has ended, and there are no purchases within 2 months, the device will no longer appear in your MikroTik account. You will have to make a new CHR installation to make a purchase within the required time frame. To request a trial license, you must run the command "`/system license renew`" from the CHR device command line. You will be asked for the username and password of your mikrotik.com account.

Warning: If you plan to use multiple virtual systems of the same kind, it may be possible that the next machine has the same SystemID as the original one. This can happen on certain cloud providers, such as Linode. To avoid this, after your first boot, run the command "`/system license generate-new-id`" before you request a trial license. Note that this feature must be used only while CHR is running on free type of RouterOS license. If you have already obtained paid or trial license, do not use regenerate feature since you will not be able to update your current key any more

To use multiple virtual machines, download the disk image from our webpage, and make as many copies, as you need virtual machines. Then make new virtual machine system from each virtual disk image.

Make sure to make copies of the Disk Image before you run or register the downloaded file.

Replacement Key

It is a special key which is issued by the MikroTik support team if you accidentally lose the license on a x86 instance running RouterOS, and the MikroTik Support employee decides that it is not directly your fault. It costs 10$ and has the same features as the key that you lost.

Note that before issuing such key, MikroTik Support can ask you to prove that the old drive has failed, in some cases, this means sending us the dead drive.

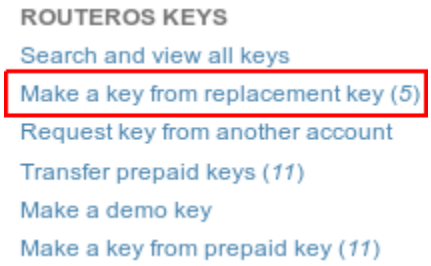
Replacement key request

1) Go to your account management in mikrotik.com and fill the "support contact form" or write a direct e-mail to support@mikrotik.com

- Please provide detailed information about why replacement key is required

2) Send required info to MikroTik support department.

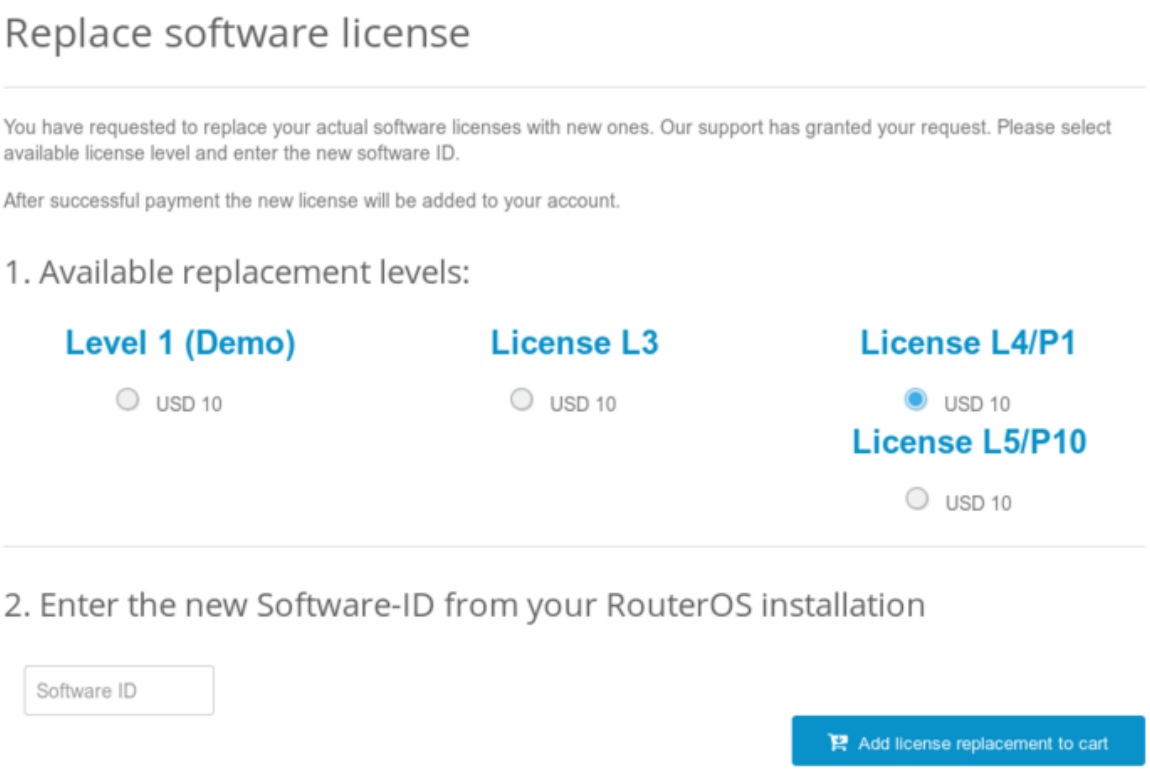
3) Re-check your account after support staff has confirmed that replacement key has been added to your account. Select the section "Make a key from replacement key"



4) Select the appropriate license level on which you wish to perform the replacement

5) Enter the new "software-ID"

6) Proceed to checkout by pressing "Add license replacement to cart" and finish the payment



7) An e-mail will be sent to your profile containing the new license.

- You can also find the newly generated key in the section "Search and view all keys" under the folder "Purchased YYYY" where "YYYY" is the current Year

> ⚠ *We may issue only one replacement key per one original key, using replacement key procedure twice for one key will not be possible. In cases like this new key for this RouterOS device must be purchased.*

Obtaining Licenses and Working With Them

Where can I buy a RouterOS license key?

MikroTik devices come preinstalled with a license and no purchase is needed.

To obtain a higher level license, or to obtain a license for a x86 PC installation, you must register an account on our webpage, and in there, use the option "Purchase a RouterOS license key".

If I have purchased my key elsewhere

You must contact the company who sold you the license, they will provide the support.

If I have a license and want to put it on another account?

You can give access to keys with the help of Virtual Folders

The only kind of licenses, that could be transferred to another Account is a prepaid key, which is purchased or is got from MUM. Prepaid keys got as a gift from the Training are not transferable.
To transfer purchased prepaid key navigate to "Transfer prepaid keys" in the section "ROUTEROS KEYS" on your MikroTik Account.

If I have lost a license on my device?

If for some reason you have lost license from your router, upgrade router to the latest RouterOS version available and use "Request license key" in your mikrotik.com account. Use soft-id and serial number available under System/License menu in RouterOS when requesting license. Apply received license or contact support@mikrotik.com if request feature do not work.

> ⚠ *If the license was lost due to repairs and they were not done via the distributor under warranty, you will have to purchase a new RouterOS license for the full price!*

Using the License

Can I Format or Re-Flash the drive?

Formatting, and Re-Imaging the drive with non-MikroTik tools (like DD and Fdisk) will destroy your license! Be very careful and contact MikroTik support before doing this. It is not recommended, as MikroTik support might deny your request for a replacement license. For this use MikroTik provided tools - Netinstall or CD-install that are freely available from our download page.

How many computers can I use the License on?

The RouterOS license can be used only in one system, at the same time. The License is bound to the HDD it is installed on, but you have the ability to move the HDD to another computer system. You cannot move the License to another HDD, neither can you format or overwrite the HDD with the RouterOS license. It will be erased from the drive, and you will have to get a new one. If you accidentally removed your license, contact the support team for help.

Can I temporary use the HDD for something else, other than RouterOS?

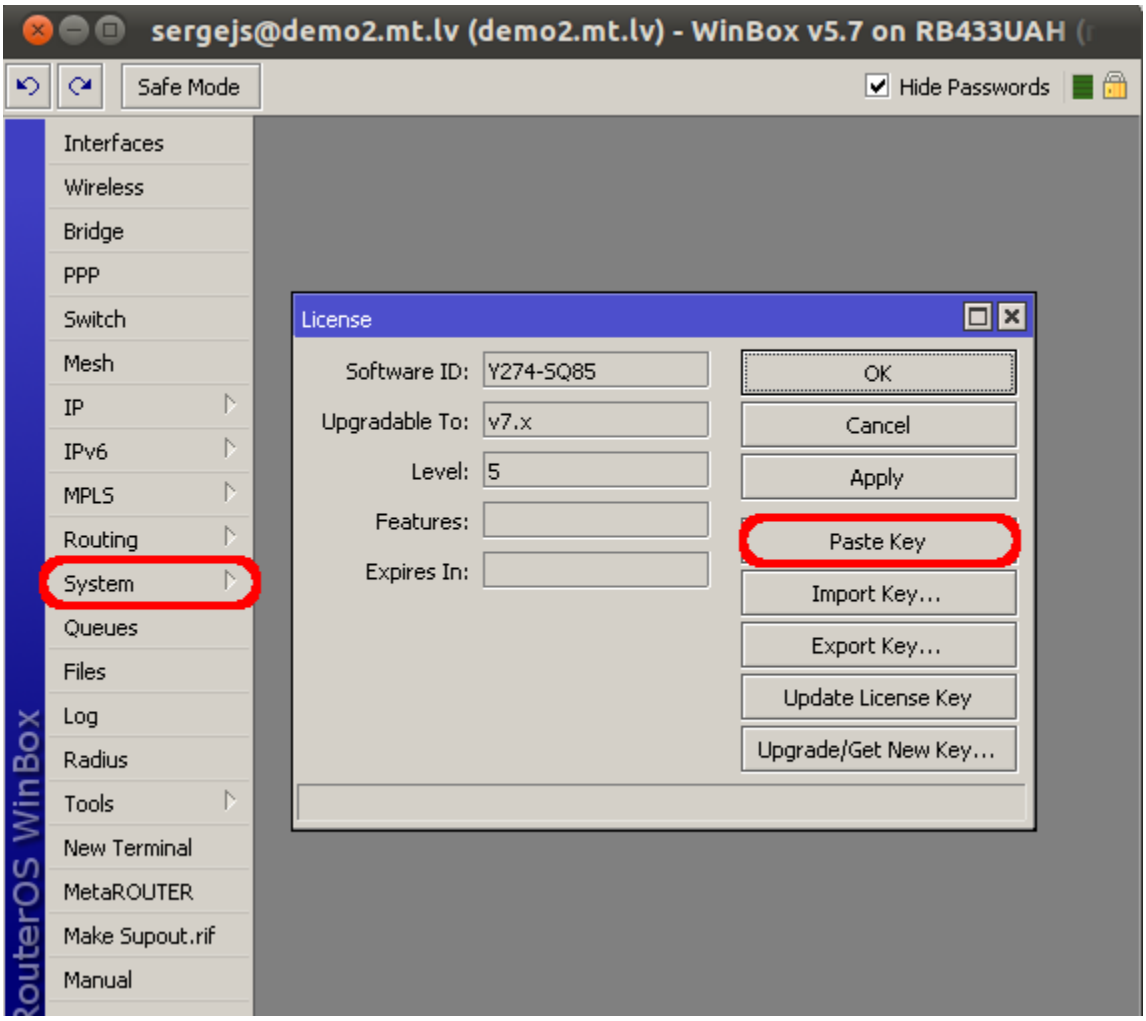
As stated above, no.

Can I move the license to another HDD ?

If your current HDD drive is destroyed, or can no longer be used, it is possible to transfer the license to another HDD. You will have to request a replacement key (see below) which will cost 10$

Must I type the whole key into the router?

No, simply copy it and paste in the menu **System** --> **License**,



Can I install another OS on my drive and then install RouterOS again later?

No, because if you use formatting, partitioning utilities or tools that do something to the MBR, you will lose the license and you will have to make a new one. This process is not free (see Replacement Key above)

I lost my RouterBOARD, can you give me the license to use on another system?

MikroTik hardware comes with an embedded license. You cannot move this license to a new system in any way, this includes any upgrades applied to the MikroTik router while it was still working.

Licenses Purchased from Resellers

The keys that you purchase from other vendors and resellers are not in your account. Your mikrotik.com account only contains licenses purchased from MikroTik directly. However, you can use the "Request key" link in your account, to get the key into your account for reference, or for some upgrades (if available).

I am not using the software, can you terminate my license?

The licenses are stand alone keys and MikroTik does not have any remote control over your devices. Therefore, we are unable to verify if you use your license or not. This is why MikroTik cannot terminate any issued licenses.

Cloud Hosted Router, CHR

Cloud Hosted Router (CHR) is a RouterOS version intended for running as a virtual machine. It supports the x86 64-bit architecture and can be used on most of the popular hypervisors such as VMWare, Hyper-V, VirtualBox, KVM, and others. CHR has full RouterOS features enabled by default but has a different licensing model than other RouterOS versions.

System Requirements

- Package version: RouterOS v6.34 or newer
- Host CPU: 64-bit with virtualization support
- RAM: 256MB or more (Max: 128GB)
- Disk: 128MB disk space for the CHR virtual hard drive (Max: 16GB)

The minimum required RAM depends on interface count and CPU count. You can get an approximate number by using the following formula:

- RouterOS v6 - RAM = 128 + [8 × (CPU_COUNT) × (INTERFACE_COUNT - 1)]
- RouterOS v7 - RAM = 512 + [8 × (CPU_COUNT) × (INTERFACE_COUNT - 1)]

Note: We recommend allocating at least 1024MiB of RAM for CHR instances.

CHR has been tested on the following platforms:

- VirtualBox 6 on Linux and OS X
- VMWare Fusion 7 and 8 on OS X
- VMWare ESXi 6.5
- Qemu 2.4.0.1 on Linux and OS X
- Hyper-V on Windows Server 2008r2, 2012 and Windows 10 *(Only Generation 1 Hyper-V virtual machine is supported at the moment)*
- Xen Server 7.1

Warning: Hypervisors that provide paravirtualization are not supported.

Usable Network and Disk interfaces on various hypervisors:

- ESX:
  - Network: vmxnet3, E1000
  - Disk: IDE, VMware paravirtual SCSI, LSI Logic SAS, LSI Logic Parallel
- Hyper-V:
  - Network: Network adapter, Legacy Network adapter
  - Disk: IDE, SCSI
- Qemu/KVM:
  - Network: Virtio, E1000, vmxnet3 (optional)
  - Disk: IDE, Sata, Virtio
- VirtualBox
  - Network: E1000, rtl8193
  - Disk: IDE, Sata, SCSI, SAS

Note: SCSI controller Hyper-V and ESX are usable just for secondary disks, system image must be used with IDE controller!

Warning: We do not recommend using the E1000 network interface if better synthetic interface options are available on a specific Hypervisor!

How to Install a virtual RouterOS system with CHR images

We provide 4 different virtual disk images to choose from. Note that they are only disk images, and you can't simply run them.

- RAW disk image (.img file)
- VMWare disk image (.vmdk file)
- Hyper-V disk image (.vhdx file)
- VirtualBox disk image (.vdi file)

Steps to install CHR

1. Download the virtual disk image for your hypervisor
2. Create a guest virtual machine
3. Use the previously downloaded image file as a virtual disk drive
4. Start the guest CHR virtual machine
5. Log in to your new CHR. The default user is 'admin', without a password

Please note that running CHR systems can be cloned and copied, but the copy will be aware of the previous trial period, so you cannot extend your trial time by making a copy of your CHR. However, you are allowed to license both systems individually. To make a new trial system, you need to make a fresh installation and reconfigure RouterOS.

Installing CHR guides

- VMWare Fusion / Workstation and ESXi 6.5
- VirtualBox
- Hyper-V
- Amazon Web Services (AWS)
- Hetzner Cloud
- Linode
- Google Compute Engine
- ProxMox

CHR Licensing

The CHR has 4 license levels:

- ***free***
- ***p1*** *perpetual-1* ($45)
- ***p10*** *perpetual-10* ($95)
- ***p-unlimited*** *perpetual-unlimited* ($250)

The 60-day free trial license is available for all paid license levels. To get the free trial license, you have to have an account on MikroTik.com as all license management is done there.

Perpetual is a lifetime license (buy once, use forever). It is possible to transfer a perpetual license to another CHR instance. A running CHR instance will indicate the time when it has to access the account server to renew its license. If the CHR instance will not be able to renew the license it will behave as if the trial period has run out and will not allow an upgrade of RouterOS to a newer version.

After licensing a running trial system, you **must** manually run the */system license renew* function from the CHR to make it active. Otherwise, the system will not know you have licensed it in your account. If you do not do this before the system deadline time, the trial will end and you will have to do a complete fresh CHR installation, request a new trial, and then license it with the license you had obtained.

License	Speed limit	Price
Free	1Mbit	FREE
P1	1Gbit	$45
P10	10Gbit	$95
P-Unlimited	Unlimited	$250

Paid licenses

p1

p1 (perpetual-1) license level allows CHR to run indefinitely. It is limited to 1Gbps upload per interface. All the rest of the features provided by CHR are available without restrictions. It is possible to upgrade *p1* to *p10* or *p-unlimited (new license level can be purchased by standard price).* After the upgrade is purchased the former license will become available for later use on your account.

p10

p10 (perpetual-10) license level allows CHR to run indefinitely. It is limited to 10Gbps upload per interface. All the rest of the features provided by CHR are available without restrictions. It is possible to upgrade *p10* to *p-unlimited* After the upgrade is purchased the former license will become available for later use on your account.

p-unlimited

The *p-unlimited* (perpetual-unlimited) license level allows CHR to run indefinitely. It is the highest-tier license and it has no enforced limitations.

Free licenses

There are several options to use and try CHR free of charge.

free

The *free* license level allows CHR to run indefinitely. It is limited to 1Mbps upload per interface. All the rest of the features provided by CHR are available without restrictions. To use this, all you have to do is download the disk image file from our download page and create a virtual guest.

60-day trial

In addition to the limited Free installation, you can also test the increased speed of P1/P10/PU licenses with a 60 trial.

You will have to have an account registered on MikroTik.com. Then you can request the desired license level for trial from your router that will assign your router ID to your account and enable the purchase of the license from your account. All the paid license equivalents are available for trial. A trial period is 60 days from the day of acquisition after this time passes, your license menu will start to show "Limited upgrades", which means that RouterOS can no longer be upgraded.

If you plan to purchase the selected license, you must do it within 60 days of the trial end date. If your trial ends, and there are no purchases within 2 months after it ended, the device will no longer appear in your MikroTik account. You will have to make a new CHR installation to make a purchase within the required time frame.

To request a trial license, you must run the command "**/system license renew**" from the CHR device command line. You will be asked for the username and password of your mikrotik.com account.

If you plan to use multiple virtual systems of the same kind, it may be possible that the next machine has the same system ID as the original one. This can happen on certain cloud providers, such as Linode. To avoid this, after your first boot, run the command "/system license generate-new-id" **before** you request a trial license. Note that this feature must be used only while CHR is running on a free type of RouterOS license. If you have already obtained a paid or trial license, do not use the regenerate feature since you will not be able to update your current key anymore

Getting the License

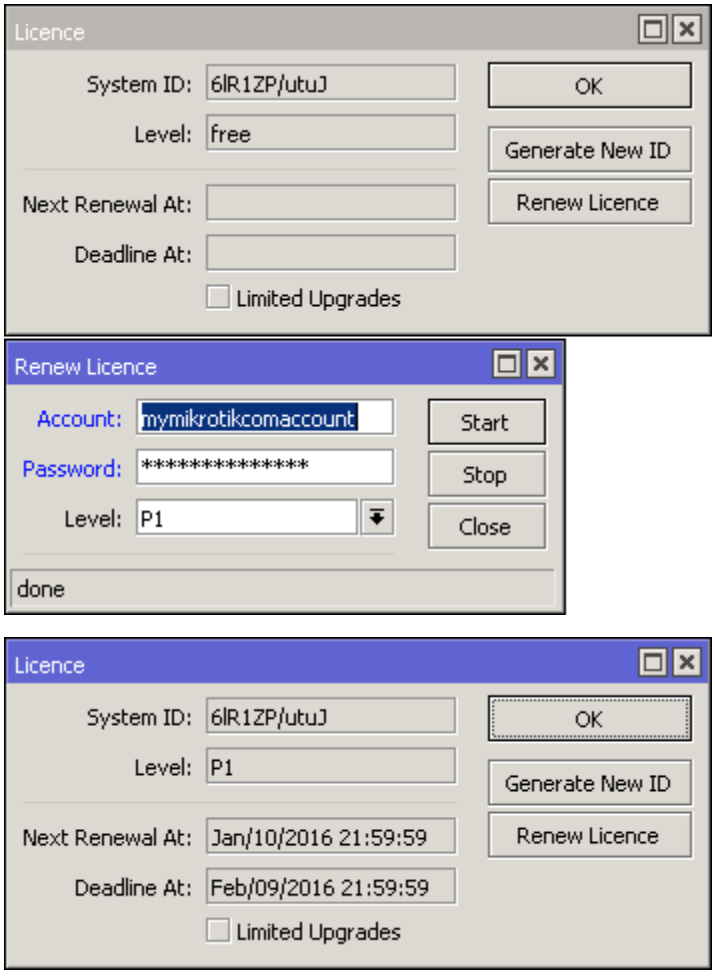
After the initial setup, a CHR instance will have a *free* license assigned. From there, it is possible to upgrade the license to a higher tier. Once you have a trial license all the work with the license is done on the account server where it is possible to upgrade the license to a higher tier unless it is *p-unlimited* already.

Upgrade from Free to p1 license level or higher

Initial upgrade from the *free* tier to anything higher than that incurs CHR instance registration on the account server. To do that you have to enter your Mikro Tik.com username and password and the desired license level you want to acquire. As a result, a CHR ID number will be assigned to your account on the account server and a 60-day trial created for that ID. There are 2 ways to obtain a license - using WinBox or RouterOS command-line interface:

Upgrade license level using WinBox

(System -> License menu):



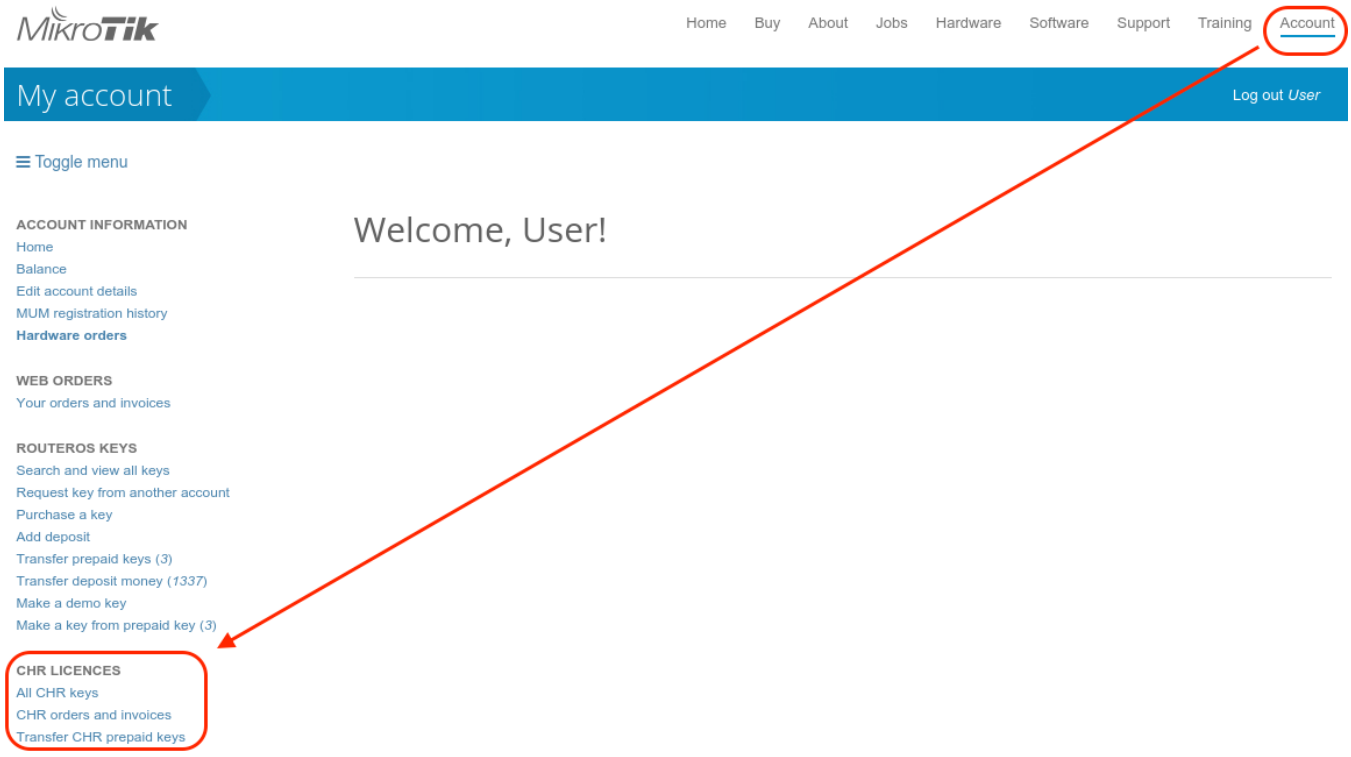
Upgrade license level using the command-line interface:

```
[admin@MikroTik] > /system license print
  system-id: 6lR1ZP/utuJ
      level: free

[admin@MikroTik] > /system license renew
account: mymikrotikcomaccount
password: ********************
level: p1
  status: done

[admin@MikroTik] > /system license print
        system-id: 6lR1ZP/utuJ
            level: p1
  next-renewal-at: jan/10/2016 21:59:59
      deadline-at: feb/09/2016 21:59:59
```

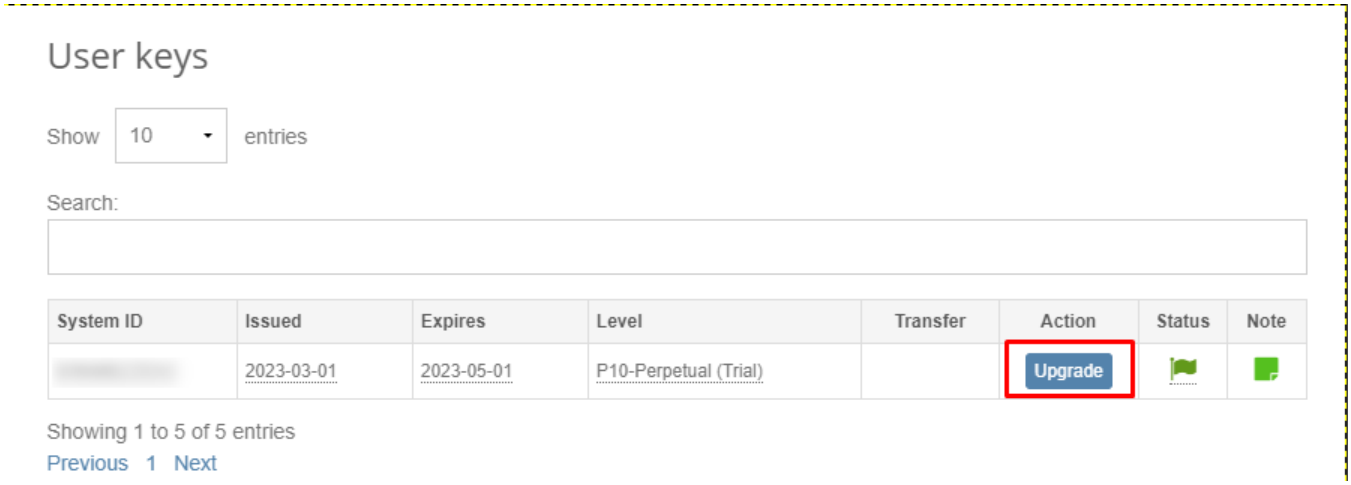
Payment:

To acquire a higher-level trial, set up a new CHR instance, renew the license, and select the desired level.

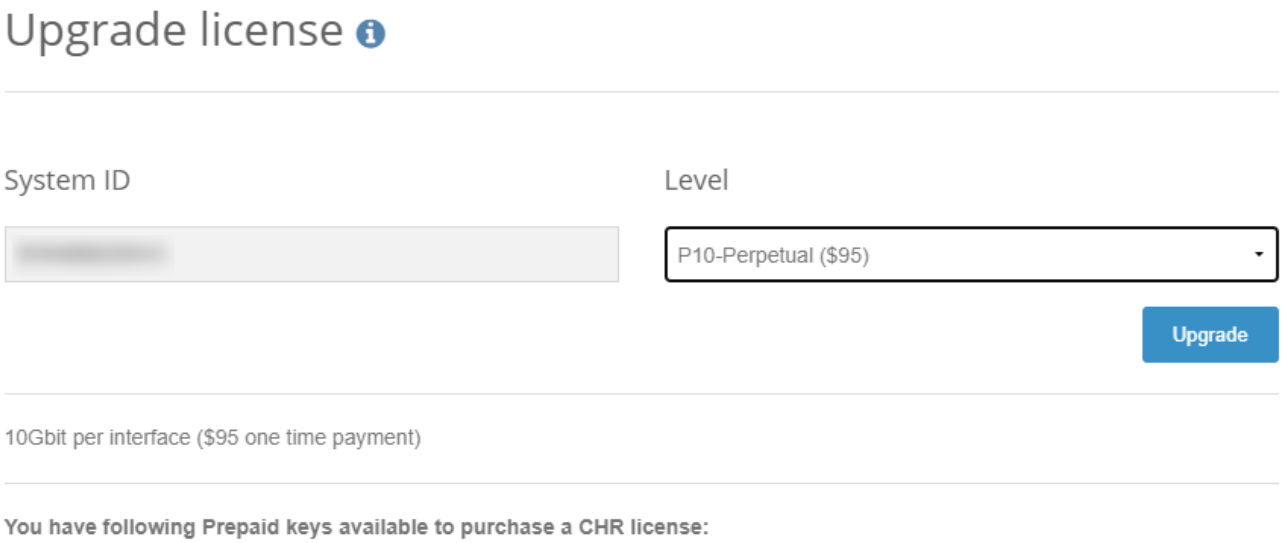
To upgrade from a Trial license to a Paid one go to MikroTik.com account server and choose 'all keys' in Cloud Hosted Router (CHR) section:



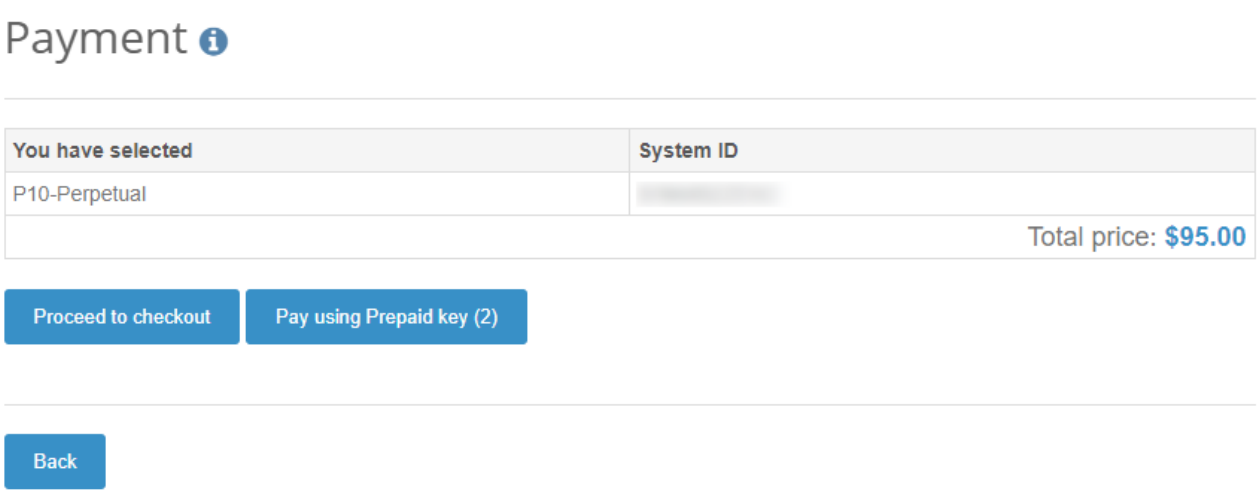
You will be presented with a list of your CHR machines and licenses:



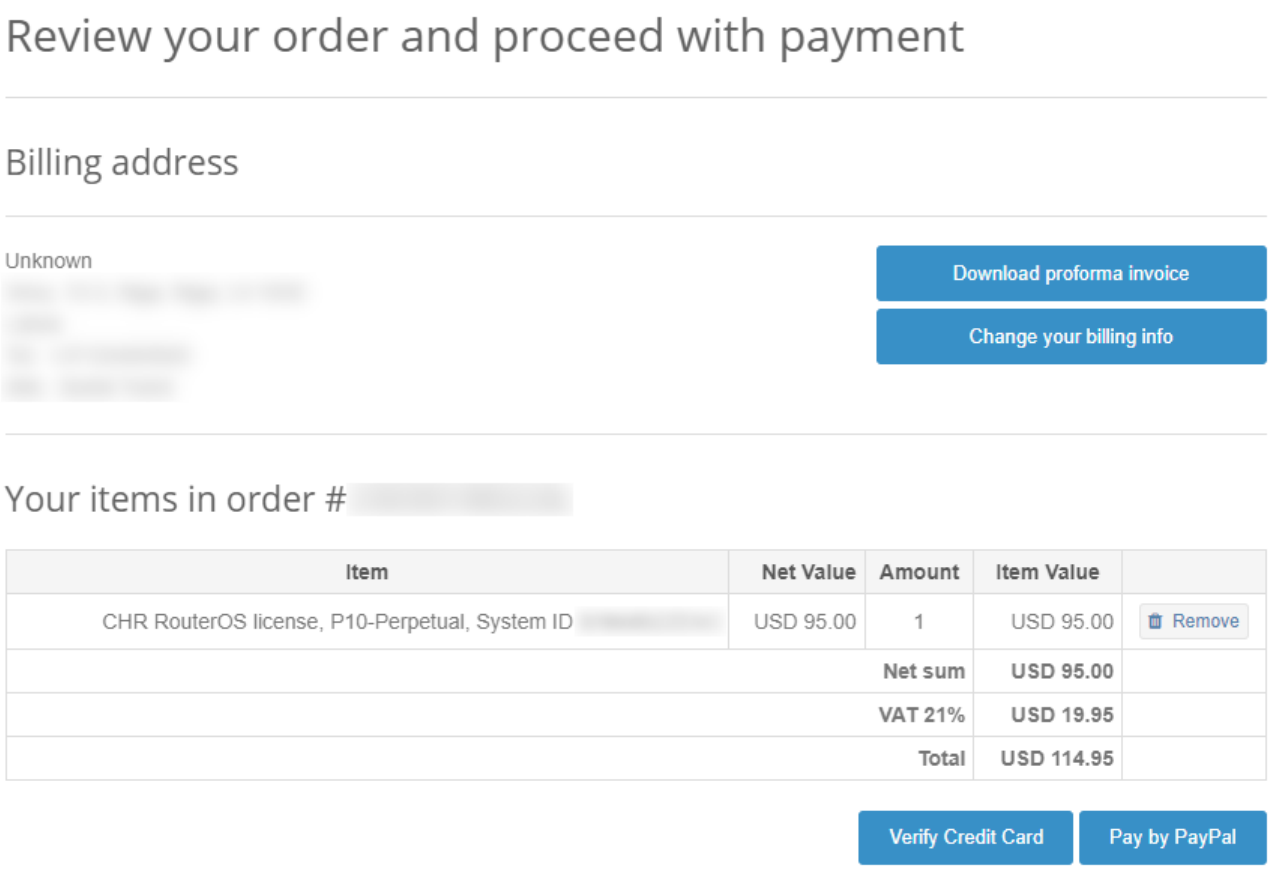
To upgrade from a Trial to a Paid license click 'Upgrade', choose the desired license level (it can be different than the level of the trial license), and click 'Upgrade key':



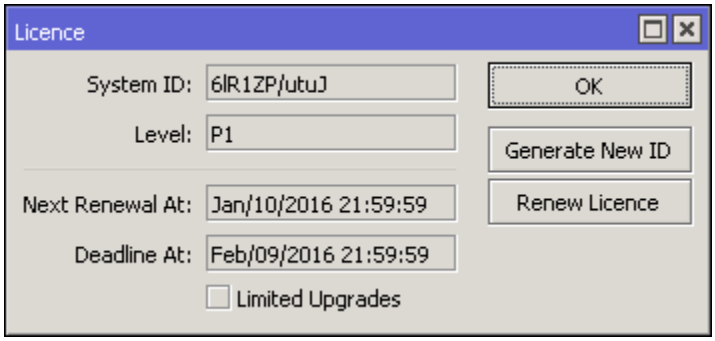
If there are prepaid keys available, it is possible to use it for CHR - press "Pay using Prepaid key". If there are no prepaid keys or you do not want to use them, press "Proceed to checkout".



Choose the payment method: It is possible to pay using a credit card (CC) or PayPal.



License Update



In '/system license' menu router will indicate the time *next-renewal-at* when it will attempt to contact the server located on licence.mikrotik.com. Communication attempts will be performed once an hour after the date on *next-renewal-at* and will not cease until the server responds with an error. If the *deadline-at* date is reached without successfully contacting the account server, the router will consider that the license has expired and will disallow further software updates. However, the router will continue to work with the same license tier as before.

If you want to upgrade perpetual license to a higher level please transfer the previous perpetual license to another CHR, to exclude the situation where the previous perpetual license is lost on upgrade.

Virtual Network Adapters

⚠ Fast Path is supported in RouterOS v7 for "vmxnet3" and "virtio-net" adapters.

RouterOS v6 does not support Fast Path.

Troubleshooting

Running on VMware ESXi

Changing MTU

VMware ESXi supports MTU of up to 9000 bytes. To get the benefit of that, you have to adjust your ESXi installation to allow a higher MTU. Virtual Ethernet interface added **after** the MTU change will be properly allowed by the ESXi server to pass jumbo frames. Interfaces added prior to MTU change on the ESXi server will be barred by the ESXi server (it will still report the old MTU as the maximum possible size). If you have this, you have to re-add interfaces to the virtual guests.

Example. There are 2 interfaces added to the ESXi guest, auto-detected MTU on the interfaces show MTU size as it was at the time when the interface was added:

```
[admin@chr-vm] > interface ethernet print
Flags: X - disabled, R - running, S - slave
 #    NAME           MTU MAC-ADDRESS       ARP
 0 R  ether1        9000 00:0C:29:35:37:5C enabled
 1 R  ether2        1500 00:0C:29:35:37:66 enabled
```

Using bridge on Linux

If Linux bridge supports IGMP snooping, and there are problems with IPv6 traffic it is required to disable that feature as it interacts with MLD packets (multicast) and is not passing them through.

```
echo -n 0 > /sys/class/net/vmbr0/bridge/multicast_snooping
```

Packets not passing from guests

The problem: after configuring a software interface (VLAN, EoIP, bridge, etc.) on the guest CHR it stops passing data to the outside world beyond the router.

The solution: check your VMS (Virtualization Management System) security settings, if other MAC addresses are allowed to pass and if packets with VLAN tags are allowed to pass through. Adjust the security settings according to your needs like allowing MAC spoofing or a certain MAC address range. For VLAN interfaces, it is usually possible to define allowed VLAN tags or VLAN tag range.

Using VLANs on CHR in various Hypervisors

In some hypervisors, before VLAN can be used on VMs, they need to first be configured on the hypervisor itself.

ESXI

Enable Promiscuous mode in a port group or virtual switch that you will use for a specific VM.

ESX documentation:

- https://kb.vmware.com/kb/1002934
- https://kb.vmware.com/kb/1004099

Hyper-V

Hyper-V documentation:

- https://technet.microsoft.com/en-us/library/cc816585(v=ws.10).aspx#Anchor_2

bhyve hypervisor

It won't be possible to run CHR on this hypervisor. CHR cannot be run as a para-virtualized platform.

Linode

When creating multiple Linodes with the same disk size, new Linodes will have the same systemID. This will cause issues to get a Trial/Paid license. To avoid this, run the command `/system license generate-new-id` after the first boot and before you request a trial or paid license. This will make sure the ID is unique.

Some useful articles:

Specific VLAN is untagged by NIC interface:

- https://blogs.msdn.microsoft.com/adamfazio/2008/11/14/understanding-hyper-v-vlans/
- https://www.aidanfinn.com/?p=10164

Allow passing other VLANs:

- https://social.technet.microsoft.com/Forums/windows/en-US/79d36d5b-c794-4502-8ed4-b7a4183b1891/vlan-tags-and-hyperv-switches?forum=winserverhyperv

Guest tools

VMWare

Time synchronization

Must be enabled from GUI ('Synchronize guest time with host'). Backward synchronization is disabled by default - if the guest is ahead of the host by more than ~5 seconds, synchronization is not performed

Power operations

- *poweron* and *resume* scripts are executed (if present and enabled) after power on and resume operations respectively.
- *poweroff* and *suspend* scripts are executed before power off and suspend operations respectively.
- If scripts take longer than 30 seconds or contain errors, the operation fails
- In case of failure, retrying the same operation will ignore any errors and complete it successfully
- Failed script output is saved to a file (e. g. 'poweroff-script.log', 'resume-script.log' etc)
- Scripts can be enabled/disabled from hypervisor GUI ('run VMware Tools Scripts') or by enabling/disabling scripts from the console

Quiescing/backup

Guest filesystem quiescing is performed only if requested.

- *freeze* script is executed before freezing the filesystem
- *freeze-fail* script is executed if the hypervisor failed to prepare for a snapshot or if *freeze* script failed
- *thaw* script is executed after the snapshot has been taken
- Script run time is limited to 60 seconds
- *freeze* script timeouts and errors result in the backup operation being aborted
- FAT32 disks are not quiesced
- Failed script output is saved to a file (e. g. 'freeze-script.log', 'freeze-fail-script.log', 'thaw-script.log')

Guest info

Networking, disk, and OS info are reported to the hypervisor every 30 seconds (GuestStats (memory) are disabled by default, and can be enabled by setting 'guestinfo.disable-perfmon = "FALSE"' in VM config).

- The order, in which network interfaces are reported, can be controlled by setting 'guestinfo.exclude-nics', 'guestinfo.primary-nics' and 'guestinfo.low-priority-nics' options. Standard wildcard patterns can be used.

Provisioning

You can use the ProcessManager from Vim API to execute scripts. Python bindings are available

- Main data structure: GuestProgramSpec
  - The *workingDirectory* and *envVariables* members are ignored
  - *programPath* must be set to either 'inline' or 'import'
  - If *programPath* is '**inline**', *arguments* are interpreted as script text
  - If *programPath* is '**import**', *arguments* are interpreted as file path

After using *GuestProgramSpec* together with an instance of GuestAuthentication as arguments to StartProgramInGuest unique *JobID* is obtained.

Script progress can be tracked by using the ListProcessesInGuest command. *ListProcessesInGuest* accepts an array of job id's; passing an empty array will report on all jobs started from the API

- *ListProcessesInGuest* returns an array of GuestProcessInfo instances:
  - *pid* field is set to *JobID*
  - *endTime* is only set after completion
  - *exitCode* is set to 0 on success and -1 on error
  - *name* is set to 'inline' or 'import' (same as *programPath* in *GuestProgramSpec*)

Information about completed jobs is kept around for ~1 minute, or until *ListProcessesInGuest* (with the corresponding *JobID*) is called. If the script fails, a file named 'vix_job_$JobID$ .txt' containing the script output is created. Script run time is limited to 120 seconds and script output is not saved on timeout,

- The vmrun command *runScriptInGuest* can also be used
- The PowerCLI cmdlet Invoke-VMScript is not supported
- Host/guest file transfer is not supported

Python example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys,time
from pyVim import connect
from pyVmomi import vmodl,vim


def runInline(content,vm,creds,source):
    ''' Execute script source on vm '''
    if isinstance(source, list):
        source = '\n'.join(source)
    ps = vim.vm.guest.ProcessManager.ProgramSpec(
                programPath = 'console',
                arguments = source
        )
    return content.guestOperationsManager.processManager.StartProgramInGuest(vm,creds,ps)

def runFromFile(content,vm,creds,fileName):
    ''' Execute script file located on CHR '''
    ps = vim.vm.guest.ProcessManager.ProgramSpec(
                programPath = 'import',
                arguments = fileName
    )
    return content.guestOperationsManager.processManager.StartProgramInGuest(vm,creds,ps)


def findDatastore(content,name):
    sessionManager = content.sessionManager

    dcenterObjView = content.viewManager.CreateContainerView(content.rootFolder, [vim.Datacenter], True)

    datacenter = None
    datastore = None
    for dc in dcenterObjView.view:
        dstoreObjView = content.viewManager.CreateContainerView(dc, [vim.Datastore], True)
        for ds in dstoreObjView:
            if ds.info.name == name:
                datacenter = dc
                datastore = ds
                break
        dstoreObjView.Destroy()

    dcenterObjView.Destroy()

    return datacenter,datastore

def _FAILURE(s,*a):
    print(s.format(*a))
    sys.exit(-1)

#--------------------------------------------------------------------------------#
```

```
if __name__ == '__main__':
    host = sys.argv[1] # ip or something
    user = 'root'
    pwd = 'MikroTik'
    vmName = 'chr-test'
    dataStoreName = 'datastore1'



    service = connect.SmartConnectNoSSL(host=host,user=user,pwd=pwd)
    if not service:
        _FAILURE("Could not connect to the specified host using specified username and password")

    content = service.RetrieveContent()


    #-----------------------------------------------------------------------------
    # Find datacenter and datastore


    datacenter,datastore = findDatastore(content,dataStoreName)

    if not datacenter or not datastore:
        connect.Disconnect(service)
        _FAILURE('Could not find datastore \'{}\'',dataStorename)


    #-----------------------------------------------------------------------------
    # Locate vm


    vmxPath = '[{0}] {1}/{1}.vmx'.format(dataStoreName, vmName)
    vm = content.searchIndex.FindByDatastorePath(datacenter, vmxPath)

    if not vm:
        connect.Disconnect(service)
        _FAILURE("Could not locate vm")


    #-----------------------------------------------------------------------------
    # Setup credentials from user name and pasword

    creds = vim.vm.guest.NamePasswordAuthentication(username = 'admin', password = '')


    #-----------------------------------------------------------------------------
    # Run script

    pm = content.guestOperationsManager.processManager

    try:
        # Run script
        src = [':ip address add address=192.168.0.1/24 interface=ether1;']
        jobID = runInline(content, vm, creds, src)

        # Or run file (from FTP root)
        # jobID = runFromFile(content,vm,creds, 'scripts/provision.rsc')


        #-----------------------------------------------------------------------------
        # Wait for job to finish

        pm = content.guestOperationsManager.processManager
        jobInfo = pm.ListProcessesInGuest(vm, creds, [jobID])[0]
        while jobInfo.endTime is None:
            time.sleep(1.0)
            jobInfo = pm.ListProcessesInGuest(vm, creds, [jobID])[0]

        if jobInfo.exitCode != 0:
            _FAILURE('Script failed!')
    except:
        raise
    else:
        connect.Disconnect(service)
```

KVM

QEMU guest agent is available. Supported agent commands can be retrieved by using the guest-info command. Host-guest file transfer can be performed by using guest-file-* commands. Guest networking information can be retrieved by using the guest-network-get-interfaces command.

- Scripts can be executed by using the guest-exec command together with the GuestExec data structure:
  - If the *path* member is provided, the corresponding file is executed
  - If the *path* member is not set and *input-data* member is provided, *input-data* value is used as script input
  - If *capture-output* is set, script output is reported back
  - *args* and *env* members are not used
- Script job progress can be monitored with guest-exec-status command. The GuestExecStatus data structure is populated as follows:
  - On success, *exitcode* member is set to 0
  - If the script timed out *exitcode* is set to 1
  - If the script contained errors *exitcode* is set to -1
  - *signal* member is not set
  - The *err-data* member is not used
  - If *capture-output* was true, Base64 encoded script output is stored in *out-data*
- An additional agent channel ('chr.provision_channel') is also available

CHR ProxMox installation

- Create a new guest with the system disk and other devices as required.
- Then you have to manually upload the CHR disk (in qcow format) on the ProxMox host.
- Use *scp* or any other comparable tool as that will use SSH for the upload and it does not require any additional configuration.
- Either copy the file to the server and then manually edit the VM's .conf file or replace the previously created system image file used for booting the guest.
- Local storage on ProxMox is in */var/lib/vz* directory. There should be a subdirectory called *images* with a directory for each VM (named by the VM number). You can copy the files directly there.
- To add the existing file to the VM, edit the VM's .conf file directly. Look in */etc/pve/qemu-server/* for a file with the VM number followed by .conf.

Note: It's a good idea to create a second test VM so you can refer to it's.conf file to make sure you get the syntax right

Alternative approach

- Create Basic VM via ProxMox web GUI.
- Make sure that VM storage is on local storage (this way there will be no need to work with the LVM config side, and the disk image can be moved later on to LVM or other desired storage if needed).
- Log into ProxMox host via SSH and navigate to the VM image directory. Default local storage is located in: *var/lib/vz/images/(VM_ID)*
- Via scp, wget or any other tool download CHR raw image (.img file) into this directory.
- Now convert the CHR raw image to qcow2 format using qemu-img tool:

```
qemu-img convert -f raw -O qcow2 chr-6.40.3.img vm-(VM_ID)-disk-1.qcow2
```

Bash script approach

If you have access to the ProxMox host then CHR VM can also be created quickly via BASH script. Below is an example of one such script.

What this script does:

- Stores tmp files in: */root/temp* dir.
- Downloads raw image archive from MikroTik download page.
- Converts image file to qcow format.
- Creates a basic VM that is attached to the MGMT bridge.

```
#!/bin/bash

#vars
version="nil"
vmID="nil"

echo "############## Start of Script ##############

## Checking if temp dir is available..."
if [ -d /root/temp ]
then
    echo "-- Directory exists!"
else
    echo "-- Creating temp dir!"
    mkdir /root/temp
fi
# Ask user for version
echo "## Preparing for image download and VM creation!"
read -p "Please input CHR version to deploy (6.38.2, 6.40.1, etc):" version
# Check if image is available and download if needed
if [ -f /root/temp/chr-$version.img ]
then
    echo "-- CHR image is available."
else
    echo "-- Downloading CHR $version image file."
    cd  /root/temp
    echo "--------------------------------------------------------------------------------"
    wget https://download.mikrotik.com/routeros/$version/chr-$version.img.zip
    unzip chr-$version.img.zip
    echo "--------------------------------------------------------------------------------"
fi
# List already existing VM's and ask for vmID
echo "== Printing list of VM's on this hypervisor!"
qm list
echo ""
read -p "Please Enter free vm ID to use:" vmID
echo ""
# Create storage dir for VM if needed.
if [ -d /var/lib/vz/images/$vmID ]
then
    echo "-- VM Directory exists! Ideally try another vm ID!"
    read -p "Please Enter free vm ID to use:" vmID
else
    echo "-- Creating VM image dir!"
    mkdir /var/lib/vz/images/$vmID
fi
# Creating qcow2 image for CHR.
echo "-- Converting image to qcow2 format "
qemu-img convert \
    -f raw \
    -O qcow2 \
    /root/temp/chr-$version.img \
    /var/lib/vz/images/$vmID/vm-$vmID-disk-1.qcow2
# Creating VM
echo "-- Creating new CHR VM"
qm create $vmID \
  --name chr-$version \
  --net0 virtio,bridge=vmbr0 \
  --bootdisk virtio0 \
  --ostype l26 \
  --memory 256 \
  --onboot no \
  --sockets 1 \
  --cores 1 \
  --virtio0 local:$vmID/vm-$vmID-disk-1.qcow2
echo "############## End of Script ##############"
```

Useful tips

- Useful snippet to clean up the BASH script from Windows formatting that may interfere with the script if it's edited on a Windows workstation:

```
sed -i -e 's/\r$//' *.sh
```

CHR Vultr installation

Vultr has more than two dozen data center locations where you can choose to deploy MikroTik CHR for the best throughput and latency.
Follow these steps to install MikroTik CHR at Vultr.

Step 1: Deploy a server in rescue mode

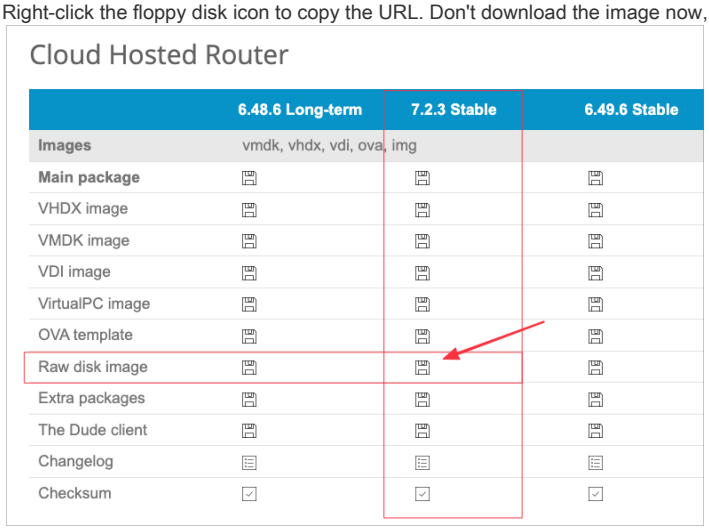
In this step, you'll deploy a new server at Vultr with SystemRescue, a bootable Linux ISO.

1. Deploy a new Cloud Compute instance.
2. Choose the location with the best performance for your needs. You can use Vultr's network-looking glass to test the throughput and latency of any location.
3. Select the **ISO Library** tab in the **Server Image** section.
4. Choose **SystemRescue x64**.
5. Choose a server size with enough bandwidth allowance for your requirements.
6. Give the server a hostname, and a label, and then click **Deploy Now**.

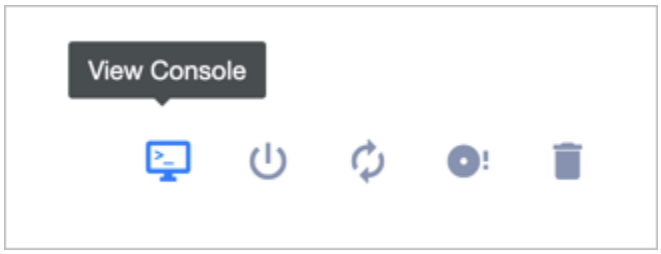
When the server finishes deploying, proceed to the next step.

Step 2: Write the CHR image to the disk

1. In your web browser, navigate to the MikroTik download page.
2. Locate the latest Stable RAW CHR disk image. Vultr requires version **7.2.3 Stable** or later.
3. Right-click the floppy disk icon to copy the URL. Don't download the image now, you'll download it to the server in a later step.



4. Navigate to the server's information page in the Vultr customer portal.
5. Connect to the web console.



6. In the web console, download the CHR image to the server with "wget". If you copied the download URL to your clipboard, you can send it to the server through the web console.

   Substitute your version for x.x.x in the examples that follow.

   ```
   # wget https://download.mikrotik.com/routeros/x.x.x/chr-x.x.x.img.zip
   ```

7. Unzip the downloaded file.

```
# unzip chr-x.x.x.img.zip
```

8. Write the MikroTik CHR image to the server's disk with dd.

```
# dd if=chr-x.x.x.img of=/dev/vda
```

   - **if** is the image that you unzipped in the previous step.
   - **of** is the server's disk: /dev/vda.

This procedure takes a couple of minutes; proceed to the next step when complete.

Step 3: Connect to MikroTik CHR

1. Navigate to the server's settings page.
2. Choose the **Custom ISO** menu, then click **Remove ISO**. The server will reboot.
3. Connect to the web console.
4. Log in as admin. There is no password set, so press **Enter** at the prompt.
5. View the software license, then choose a new, strong password.
6. Close the web console, then open a terminal on your local computer.
7. SSH as admin to the server's IP address.

```
$ ssh admin@192.0.2.2
```

8. Enter the strong password you set in the prior step.

This completes the basic installation. Please secure your MicroTik CHR router and consult the documentation to configure the server for production use. Visit Vultr site for configuration manual and for their firewall features.

Upgrading to v7

Introduction

This document describes the recommended steps for upgrading RouterOS to v7 major release and the possible caveats when doing so.

Upgrading from v6 to v7 happens the same way, as upgrading within v6 releases. Please follow the Upgrade manual for more detailed steps. If you are currently running RouterOS version 6 or older, we first suggest upgrading to the latest stable or long-term release in v6.

> In most RouterOS setups that run fine with the aforementioned v6 versions, no extra steps are required. Upgrading to v7 will automatically convert the configuration and your device will function right away.

> **Note:** We do not recommend running v7 on hardware that does not have at least 64 MB of RAM.

Feature list compatibility

As previously stated, nearly all RouterOS systems can use the "Check for updates" functionality and upgrade to v7 in a few clicks, but there are some features, where extra steps may be required:

Feature	Status
CAPsMAN	OK
Interfaces	OK
Wireless	OK
Bridge/Switching	OK
Tunnels/PPP	OK
IPv6	OK
BGP	OK, but attention is required *
OSPF	OK, but attention is required **
MPLS	OK, but attention is required ***
Routing filters	OK, but attention is required ****
PIM-SM	See notes
IGMP Proxy	OK
Tools	OK
Queues	OK
Firewall	OK
HotSpot	OK
Static Routing	OK
User Manager	See notes

Notes

The routing protocol configuration upgrade is triggered only once. This means that if a router was downgraded to ROSv6, the configuration was modified and the router got upgraded back to ROSv7, then the resulting configuration is the one that was present before the downgrade. To re-trigger v6 configuration conversion, load ROSv6 backup with the option `force-v6-to-v7-configuration-upgrade=yes`.

BGP

All known configurations will upgrade from 6.x to 7.x successfully. But keep in mind that there is a complete redesign of the configuration. v7 BGP implementation provides with **`connection`**, **`template`** and **`session`** menus.

`Template` contains all BGP protocol-related configuration options. It can be used as a template for dynamic peers and apply a similar config to a group of peers. Most of the parameters are similar to the previous implementation except that some are grouped in the output and input section making the config more readable and easier to understand whether the option is applied on input or output.

BGP **`connection`** minimal set of parameters are `remote.address`, `template`, `connect`, `listen` and `local.role`
Connect and listen to parameters specify whether peers will try to connect and listen to a remote address or just connect or just listen. It is possible that in setups where peer uses the multi-hop connection `local.address` must be configured too. Peer role is now a mandatory parameter, for basic setups, you can just use ibgp, ebgp.

Now you can monitor the status of all connected and disconnected peers from `/routing bgp session` menu.
Other great debugging information on all routing processes can be monitored from `/routing stats` menu.

Networks are added to the firewall address-list and referenced in the BGP **`connection`** configuration.

OSPF

All known configurations will upgrade from 6.x to 7.x successfully.
OSPFv2 and OSPFv3 are now merged into one single menu `/routing ospf`. At the moment there are no default instances and areas. To start OSPF you need to create an instance and then add area to the instance.

RouterOSv7 uses templates to match the interface against the template and apply configuration from the matched template. OSPF menus `interface` and `neighbor` contains read-only entries for status monitoring.

MPLS

Upgrade MPLS setups with caution, and make sure to backup configuration before the upgrade.

Routing filters

All supported options are upgraded without any issue, in the case of an unsupported option - an empty entry is created. The routing filter configuration is changed to a script-like configuration.

The rule now can have "if .. then" syntax to set parameters or apply actions based on conditions from the "if" statement.

Multiple rules without action are stacked in a single rule and executed in order like a firewall, the reason is that the "set" parameter order is important, and writing one "set"s per line, allows for an easier understanding from top to bottom on what actions were applied.

More RouterOSv7 routing filter examples are here.

PIM-SM

Upgrading RouterOS to v7 will not preserve PIM-related configuration. After the upgrade, multicast routing configuration will be available under the `/routing/pimsm` menu and an additional "multicast" package is not required anymore. More information is available here.

User Manager

RouterOSv7 provides the new and redesigned implementation of User Manager, configuration is now integrated into RouterOS WinBox and console (WEB admin configuration interface is not available), more information is available here. Direct migration from older User Manager is not possible, it is possible to migrate older database from `/user-manager/database/migrate-legacy-db` However, it might be a good idea to start configuration from the scratch.

New features

A New Kernel is implemented in RouterOSv7, which leads to performance changes due to route cache, as well some tasks might require higher CPU and RAM usage for different processes.

- completely new NTP client and server implementation
- merged individual packages, only bundle and a few extra packages remain *(dropped support for LCD and KVM packages)*
- new Command Line Interface (CLI) style (RouterOS v6 commands are still supported)
- support for Let's Encrypt certificate generation
- support for REST API
- support for UEFI boot mode on x86
- CHR FastPath support for "vmxnet3" and "virtio-net" drivers
- support for "Cake" and "FQ_Codel" type queues
- support for IPv6 NAT
- support for Layer 3 hardware acceleration on all CRS3xx devices
- support for MBIM driver with basic functionality support for all modems with MBIM mode
- support for MLAG on CRS3xx devices
- support for VRRP grouping and connection tracking data synchronization between nodes
- support for Virtual eXtensible Local Area Network (VXLAN)
- support for L2TPv3
- support for OpenVPN UDP transport protocol
- support for WireGuard
- support for hardware offloaded VLAN filtering on RTL8367 (RB4011, RB100AHx4) and MT7621 (hEX, hEX S, RBM33G) switches
- support for ZeroTier on ARM and ARM64 devices
- completely new alternative wireless package "wifiwave2" with 802.11ac Wave2, WPA3, and 802.11w management frame protection support (requires ARM CPU and 256MB RAM)
- support for hardware offloaded VLAN filtering on RTL8367 (RB4011, RB100AHx4) and MT7621 (hEX, hEX S, RBM33G) switches
- support CPU frequency scaling for x86 devices

Dropped support

In RouterOS v7 has been dropped support for:

- LCD package
- KVM package

RouterBOOT

RouterBOOT is responsible for starting RouterOS in RouterBOARD devices.

Main and Backup loaders

By default, the main (regular) loader is used, but RouterBOARD devices also have a secondary (backup) bootloader, which can be used in case the main doesn't work. It is possible to call the backup loader with a configuration setting in RouterOS:

```
system/routerboard/settings/set force-backup-booter=yes
```

It is also possible to use the backup booter by turning on the device, with the RESET button pushed. It is only possible to upgrade the main RouterBOOT, so in case of failure, you can use the backup booter to start the device and downgrade the main loader. For upgrade instructions, follow the separate instructions in RouterBOARD#UpgradingRouterBOOT

RouterBOARD reset button

RouterBOOT reset button has three functions:

- Hold this button during boot time until the LED light starts flashing, and release the button to reset the RouterOS configuration (total 5 seconds)
- Keep holding for 5 more seconds, LED turns solid, release now to turn on CAPs mode (total 10 seconds)
- Or Keep holding the button for 5 more seconds until the LED turns off, then release it to make the RouterBOARD look for Netinstall servers

> ⚠ If you hold the button before applying power, backup RouterBOOT will be used in addition to all the above actions. To do the above actions without loading the backup loader, push the button right after applying power to the device.

Reset the password

https://www.youtube.com/watch?v=6Unz92rABs8

Configuration Reset For **Wireless Wire** kits

The reset button has the same functionality as on other devices, explained in detail https://help.mikrotik.com/docs/display/ROS/Reset+Button

5-second button hold on startup (USR LED light starts flashing) - resets to password-protected state.

10-second button hold on startup (USR LED turns solid after flashing) - completely removes configuration.

Configuration

For RouterBOARD devices that feature a serial console connector, it is possible to access the RouterBOOT loader configuration menu. The required cable is described in the Serial Console manual. RouterBOARD serial port is configured to **115200bit/s**, **8 data bits**, **1 stop bit**, and **no parity**. We suggest disabling the hardware flow control.

This example shows the menu which is available in RouterBOOT 7.4beta4:

```
RouterBOOT booter 7.4beta4

CRS328-24P-4S+

built by build at Jun/15/2022 11:34:09 from revision 73B4521C

CPU frequency: 800 MHz
  Memory size: 512 MiB
 Storage size:  16 MiB

Press Ctrl+E to enter etherboot mode
Press any key within 2 seconds to enter setup


RouterBOOT-7.4beta4
What do you want to configure?
   d - boot delay
   k - boot key
   s - serial console
   n - silent boot
   o - boot device
   z - extra kernel parameters
   r - reset booter configuration
   e - format storage
   w - repartition nand
   g - upgrade firmware
   i - board info
   p - boot protocol
   b - booter options
   j - boot os
   t - hardware tests
   l - erase license
   x - exit setup
your choice:
```

The options are self-explanatory.

letter	description	explanation
d	boot delay	Delays starting of RouterOS to allow an interface to initialize
k	boot key	The button that will open the configuration menu
s	serial console	Sets the baud rate of the serial port
n	silent boot	Suppresses all output on the serial port, in case some device is connected to it (like a GPS device or a temperature monitor)
o	boot device	Allows to enable Netinstall booting
z	extra kernel parameters	
r	reset booter configuration	Resets the settings in this menu. **Warning, no confirmation!**
e	format storage	Destroys all data on the NAND, including RouterOS configuration and license
w	repartition nand	Refer to the Partitions document for more info
y	active partition	Choose an active partition from which to try to load RouterOS
g	upgrade firmware	Allows upgrading RouterBOOT version through the network, or the XModem protocol
i	board info	
p	boot protocol	
b	booter options	Select which bootloader to use by default

t	do memory testing	booter options
j	boot os	do memory testing
t	hardware tests	
l	erase license	
x	exit setup	

Hitting the appropriate keyboard letter will give you a list of further options, they are shown below:

```
# d - boot delay:

Select boot delay:
   1 - 1s
 * 2 - 2s
   3 - 3s
   4 - 4s
   5 - 5s
   6 - 6s
   7 - 7s
   8 - 8s
   9 - 9s

# k - boot key:

Select key which will enter setup on boot:
 * 1 - any key
   2 - <Delete> key only

# s - serial console:

Select baud rate for serial console:
 * 1 - 115200
   2 - 57600
   3 - 38400
   4 - 19200
   5 - 9600
   6 - 4800
   7 - 2400
   8 - 1200
   9 - off

# n - silent boot:

Silent boot:
   0 - off
 * 1 - on

# o - boot device:

Select boot device:
   e - boot over Ethernet
 * n - boot from NAND, if fail then Ethernet
   1 - boot Ethernet once, then NAND
   o - boot from NAND only
   b - boot chosen device
   f - boot Flash Configure Mode
   3 - boot Flash Configure Mode once, then NAND


# f - cpu frequency:

Select CPU frequency:
   a -  200MHz
   b -  400MHz
   c -  600MHz
   d -  800MHz
   e - 1000MHz
 * f - 1200MHz

# r - reset booter configuration:

# e - format nand:

Do you realy want to format your storage device?
that would result in losing all your data
type "yes" to confirm:

# w - repartition nand:
```

```
Select parititon count:
   1 - partition
 * 2 - partitions
   3 - partitions
   4 - partitions

# y - active partition:

Select active partiton:
 * 0 - partition
   1 - partition

# g - upgrade firmware:

Upgrade firmware options:
   e - upgrade firmware over ethernet
   s - upgrade firmware over serial port

# i - board info:

Board Info:

        Board type: CCR1009-8G-1S-1S+
     Serial number: 48FF01DDE6FD
  Firmware version: 3.19
     CPU frequency: 1200 MHz
       Memory size: 2048 MiB
         NAND size: 128 MiB
        Build time: 2014-09-23 15:02:34
  eth1 MAC address: 00:0C:42:00:BE:4A
  eth2 MAC address: 00:0C:42:00:BE:4B
  eth3 MAC address: 00:0C:42:00:BE:4C
  eth4 MAC address: 00:0C:42:00:BE:4D
  eth5 MAC address: 00:0C:42:00:BE:4E
  eth6 MAC address: 00:0C:42:00:BE:4F
  eth7 MAC address: 00:0C:42:00:BE:50
  eth8 MAC address: 00:0C:42:00:BE:51
  eth9 MAC address: 00:0C:42:00:BE:52
 eth10 MAC address: 00:0C:42:00:BE:53

# p - boot protocol:

Choose which boot protocol to use:
 * 1 - bootp protocol
   2 - dhcp protocol

# b - booter options:

Select which booter you want to load:
 * 1 - load regular booter
   2 - force backup-booter loading

#t - do memory testing:

launches built in memory test!

# x - exit setup:

Exit bios configuration menu and continues with system startup.
```

Simple Upgrade

RouterBOOT can be upgraded from RouterOS by:

- Run command */system routerboard upgrade*
- Reboot your router to apply the upgrade (*/system reboot*)]

```
[admin@admin] > system/routerboard/upgrade
Do you really want to upgrade firmware? [y/n]
```

⚠ Every ROS version has a new RouterBoot version included in it, once you perform a ROS upgrade we always recommend upgrading RouterBoot also.

Checking RouterBOOT version

This command shows the current RouterBOOT version of your device and the available upgrade which is included in *routeros-x.yy.npk* package, or if you uploaded a **.FWF* file corresponding to the device model:

```
[admin@admin] >  system/routerboard/print
               ;;; Firmware upgraded successfully, please reboot for changes
                   to take effect!
      routerboard: yes
       board-name: hAP ac
            model: RouterBOARD 962UiGS-5HacT2HnT
    serial-number: 6737057562DD
    firmware-type: qca9550L
  factory-firmware: 3.29
  current-firmware: 6.49.5
  upgrade-firmware: 7.4beta5
```

In this case, you see, there is **a newer version** of the Bootloader firmware available already inside your current RouterOS version and it has been updated and requires a reboot.

A downgrade is also possible by uploading *.FWF file with an older version may be required for troubleshooting purposes when contacting MikroTik support.

IPv4 and IPv6 Fundamentals

In This Section:

Networking Models

Computer networks consist of many different components and protocols working together. To understand the concept of how node to node communication happens, let's get familiar to the OSI model and TCP/IP model. Both models help to visualize how communication between nodes is happening.

OSI Model

The Open Systems Interconnection (OSI) model is a 7-layer model that today is used as a teaching tool. The OSI model was originally conceived as a standard architecture for building network systems, but in real-world networks are much less defined than the OSI model suggests.

- **Layer 7 (Application)** - a protocol that defines the communication between the server and the client, for example, HTTP protocol. If the web browser wants to download an image, the protocol will organize and execute the request;
- **Layer 6 (Presentation)** - ensures data is received in a usable format. Encryption is done here (but in reality it may not be true, for example, IPSec);
- **Layer 5 (Session)** - responsible for setting up, managing and closing sessions between client and server;
- **Layer 4 (Transport)** - transport layers primary responsibility is assembly and reassembly, a data stream is divided into chunks (segments), assigned sequence numbers and encapsulated into protocol header (TCP, UDP, etc.);
- **Layer 3 (Network)** - responsible for logical device addressing, data is encapsulated within an IP header and now called "packet";
- **Layer 2 (Data link)** - Data is encapsulated within a custom header, either 802.3 (Ethernet) or 802.11 (wireless) and is called "frame", handles flow control;
- **Layer 1 (Physical)** - Communication media that sends and receives bits, electric signaling, and hardware interface;

TCP/IP Model

This model has the same purpose as the OSI model but fits better into modern network troubleshooting. Comparing to the OSI model, TCP/IP is a 4-layer model:

- **Application layer (4)** - includes application, presentation and session layers of the OSI model, which significantly simplifies network troubleshooting;
- **Transport layer (3)** - same as a transport layer in the OSI model (TCP, UDP protocols);
- **Internet layer (2)** - does the same as Network layer in the OSI model (include ARP, IP protocols);
- **Link layer (1)** - also called the Network Access layer. Includes both Layer1 and 2 of the OSI model, therefore its primary concern is physical data exchange between network nodes;

TCP/IP	OSI Model	Protocols
Application Layer	Application Layer	DNS, DHCP,HTTP,SSH etc.
	Presentation Layer	JPEG,MPEG,PICT etc.
	Session Layer	PAP, SCP, ZIP etc.
Transport Layer	Transport Layer	TCP, UDP
Internet Layer	Network Layer	ICMP, IGMP, IPv4, IPv6, IPSec
Link Layer	Data Link Layer	ARP, CDP, MPLS, PPP etc.
	Physical Layer	Bluetooth, Ethernet, Wi-Fi etc.

Ethernet

The most commonly used link layer protocol (OSI Layer2) in computer networks is the Ethernet protocol. In order to communicate, each node has a unique assigned address, called MAC (Media Access Control address) sometimes it is also called an Ethernet address.

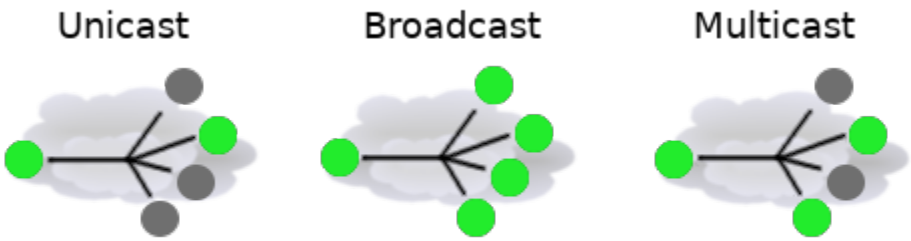
It is 48-bit long and typically fixed by the manufacturer (cannot be changed), but in recent years customization of MAC addresses is widely used, RouterOS also allows to set custom MAC address.

Most commonly used MAC format is 6 hexadecimal numbers separated by colons (`D4:CA:6D:01:22:96`)

RouterOS shows MAC address in a configuration for all Ethernet-like interfaces (Wireless, 60G, VPLS, etc.)

```
[admin@rack1_b32_CCR1036] /interface ethernet> print
Flags: X - disabled, R - running, S - slave
 #    NAME                 MTU MAC-ADDRESS       ARP             SWITCH
 0 R  ether1              1500 D4:CA:6D:01:22:96 enabled
 1 R  ether2              1500 D4:CA:6D:01:22:97 enabled
 2 R  ether3              1500 D4:CA:6D:01:22:98 enabled
 3    ether4              1500 D4:CA:6D:01:22:99 enabled
 4    ether5              1500 D4:CA:6D:01:22:9A enabled
 5    ether6              1500 D4:CA:6D:01:22:9B enabled
 6    ether7              1500 D4:CA:6D:01:22:9C enabled
 7 R  ether8              1500 D4:CA:6D:01:22:9D enabled
 8    sfp-sfpplus1        1500 D4:CA:6D:01:22:94 enabled
 9    sfp-sfpplus2        1500 D4:CA:6D:01:22:95 enabled
```

There are three types of addresses:



- **Unicast** address is sent to all nodes within the collision domain, which typically is Ethernet cable between two nodes or in case of wireless all receivers that can detect wireless signals. Only remote node with matching MAC address will accept the frame (unless the promiscuous mode is enabled)
- One of the special addresses is **broadcast** address (`FF:FF:FF:FF:FF:FF`), a broadcast frame is accepted and forwarded over Layer2 network by all nodes
- Another special address is **multicast**. Frames with multicast addresses are received by all nodes configured to receive frames with this address.

IP Networking

Ethernet protocol is sufficient to get data between two nodes on an Ethernet network, but it is not used on its own. For Internet/Networking layer (OSI Layer 3) IP (Internet Protocol) is used to identify hosts with unique logical addresses.

Most of the current networks use IPv4 addresses, which are 32bit address written in dotted-decimal notation (`192.168.88.1`)

There can be multiple logical networks and to identify which network IP address belongs to, the netmask is used. Netmask typically is specified as a number of bits used to identify a logical network. The format can also be in decimal notation, for example, the 24-bit netmask can be written as `255.255.255.0`

Let's take a closer look at 192.168.3.24/24:

```
11000000 10101000 00000011 00011000 => 192.168.3.24
11111111 11111111 11111111 00000000 => /24 or 255.255.255.0
```

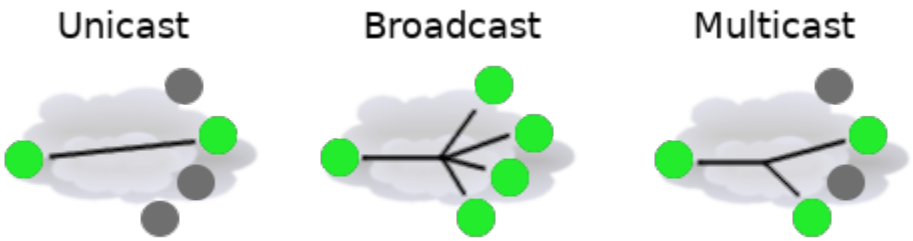
As can be seen from the illustration above high 24 bits are masked, leaving us with a range of 0-255.

From this range, the first address is used to identify the network (in our example network address would be 192.168.3.0) and the last one is used for network broadcast (192.168.3.255). That leaves us with a range from 1 to 254 for host identification which is called **unicast** addresses.

The same as in Ethernet protocol there can be also special addresses:

- **broadcast** - address to send data to all possible destinations ("all-hosts broadcast"), which permits the sender to send the data only once, and all receivers receive a copy of it. In the IPv4 protocol, the address 255.255.255.255 is used for local broadcast. In addition, a directed (limited) broadcast can be made to network broadcast address;
- **multicast** - address associated with a group of interested receivers. In IPv4, addresses `224.0.0.0` through `239.255.255.255` are designated as multicast addresses. The sender sends a single datagram from its unicast address to the multicast group address and the intermediary routers take care of making copies and sending them to all receivers that have joined the corresponding multicast group;

In case of logical IP network, unicast, broadcast and multicast visualization would look a bit different



There are also address ranges reserved for a special purpose, for example, private address range, that should be used only in local networks and typically are dropped when forwarded to the internet:

- 10.0.0.0/8 - start: 10.0.0.0; end: 10.255.255.255
- 172.16.0.0/12 - start: 172.16.0.0; end:172.31.255.255
- 192.168.0.0/16 - start: 192.168.0.0; end: 192.168.255.255

ARP and Tying It All Together

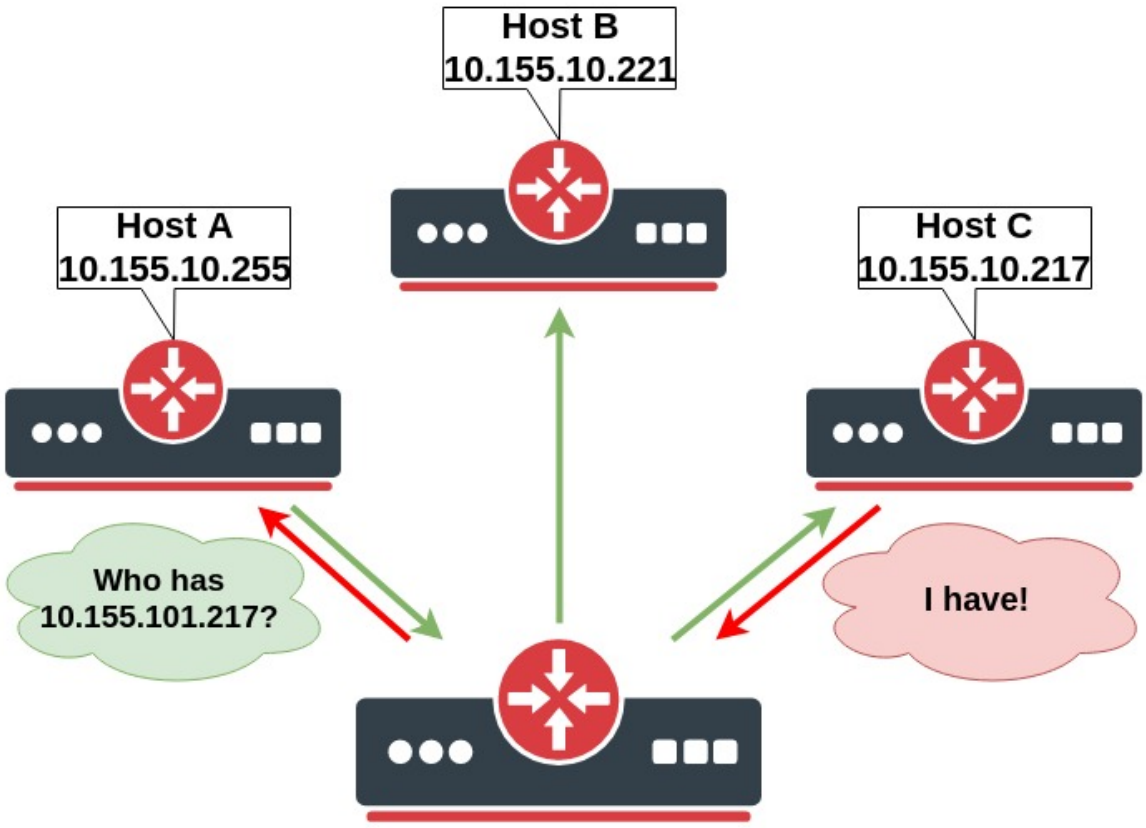
Even though IP packets are addressed using IP addresses, hardware addresses must be used to actually transport data from one host to another.

This brings us to Address Resolution Protocol (ARP) which is used for mapping the IP address of the host to the hardware address (MAC). ARP protocol is referenced in RFC 826.

Each network device has a table of currently used ARP entries. Normally the table is built dynamically, but to increase network security, it can be partially or completely built statically by means of adding static entries.

> ⚠ Address Resolution Protocol is a thing of the past. IPv6 completely eliminates use of the ARP.

When a host on the local area network wants to send an IP packet to another host in this network, it must look for the Ethernet MAC address of destination host in its ARP cache. If the destination host's MAC address is not in the ARP table, then the ARP request is sent to find the device with a corresponding IP address. ARP sends a broadcast request message to all devices on the LAN by asking the devices with the specified IP address to reply with its MAC address. A device that recognizes the IP address as its own returns ARP response with its own MAC address:



Let's make a simple configuration and take a closer look at processes when Host A tries to ping Host C.

At first, we add IP addresses on Host A:

```
/ip address add address=10.155.101.225 interface=ether1
```

Host B:

```
/ip address add address=10.155.101.221 interface=ether1
```

Host C:

```
/ip address add address=10.155.101.217 interface=ether1
```

Now, let's run a packet sniffer that saves packet dump to the file and run the ping command on Host A:

```
/tool sniffer
  set file-name=arp.pcap filter-interface=ether1
  start
/ping 10.155.101.217 count=1
  stop
```

Now you can download arp.pcap file from the router and open it in Wireshark for analyzing:

PcsCompu_85:69:b5	Broadcast	ARP	42 Who has 10.155.101.217? Tell 10.155.101.225
PcsCompu_3c:79:3a	PcsCompu_85:69:b5	ARP	42 10.155.101.217 is at 08:00:27:3c:79:3a
10.155.101.225	10.155.101.217	ICMP	70 Echo (ping) request id=0x1c01, seq=0/0, ttl=255 (reply in 428)
10.155.101.217	10.155.101.225	ICMP	70 Echo (ping) reply id=0x1c01, seq=0/0, ttl=64 (request in 427)

- Host A sends ARP message asking who has "10.155.101.217"
- Host C responds that 10.155.101.217 can be reached at 08:00:27:3C:79:3A MAC address
- Both Host A and Host C now have updated their ARP tables and now ICMP (ping) packets can be sent

If we look at ARP tables of both host we can see relevant entries, in RouterOS ARP table can be viewed by running command: `/ip arp print`

```
[admin@host_a] /ip arp> print
Flags: X - disabled, I - invalid, H - DHCP, D - dynamic, P - published,
C - complete
 #    ADDRESS         MAC-ADDRESS       INTERFACE
 0 DC 10.155.101.217  08:00:27:3C:79:3A ether1

 [admin@host_b] /ip arp> print
Flags: X - disabled, I - invalid, H - DHCP, D - dynamic, P - published,
C - complete
 #    ADDRESS         MAC-ADDRESS       INTERFACE
 0 DC 10.155.101.225  08:00:27:85:69:B5 ether1
```

ARP Modes

Now the example above demonstrated default behavior, where ARP is enabled on interfaces, but there might be scenarios where different ARP behavior is necessary. RouterOS allows configuring different ARP modes for interfaces that support ARP.

Enabled

ARPs will be discovered automatically and new dynamic entries will be added to the ARP table. This is a default mode for interfaces in RouterOS and illustrated in the example above.

Disabled

If the ARP feature is turned off on the interface, i.e., *arp=disabled* is used, ARP requests from clients are not answered by the router. Therefore, static ARP entry should be added to the clients as well. For example, the router's IP and MAC addresses should be added:

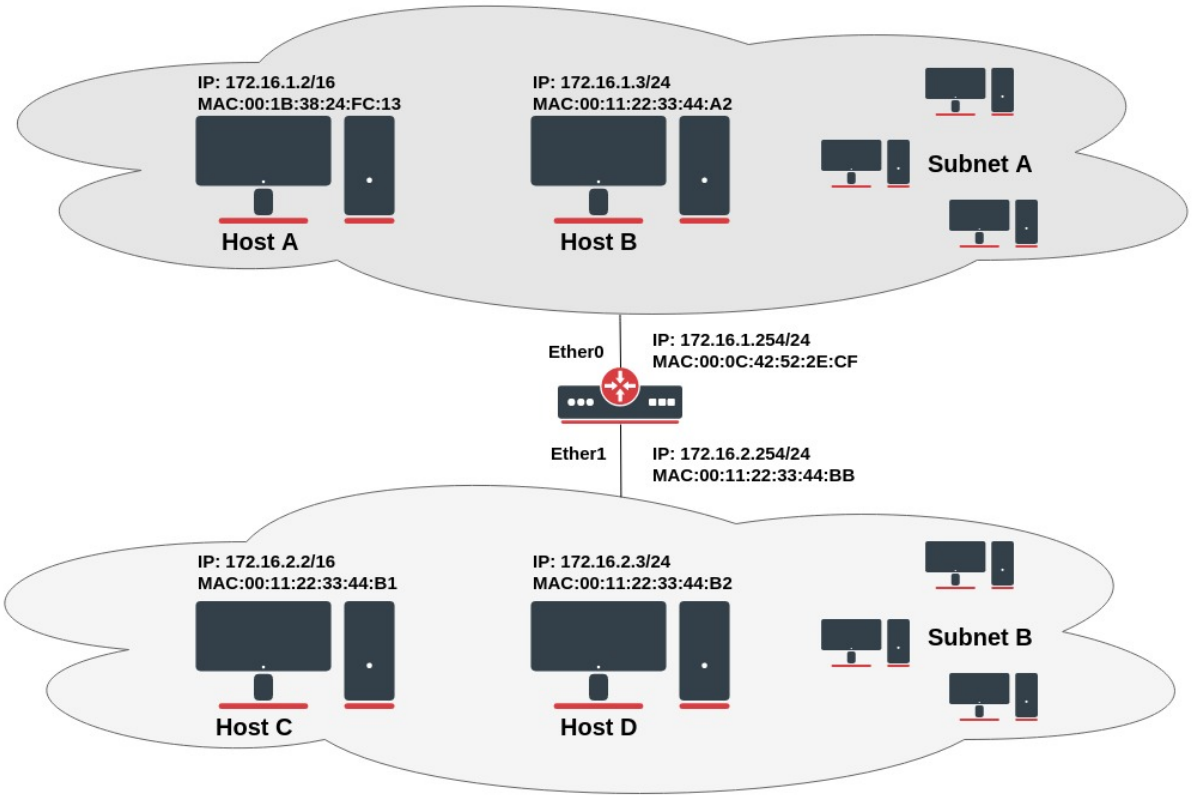
```
[admin@host_a] > /ip arp add mac-address=08:00:27:3C:79:3A address=10.155.101.217 interface=ether1
```

Reply Only

If the ARP property is set to *reply-only* on the interface, then the router only replies to ARP requests. Neighbour MAC addresses will be resolved using */ip arp* statically, but there will be no need to add the router's MAC address to other hosts' ARP tables like in cases where ARP is disabled.

Proxy ARP

A router with properly configured proxy ARP feature acts as a transparent ARP proxy between directly connected networks. This behavior can be useful, for example, if you want to assign dial-in (PPP, PPPoE, PPTP) clients IP addresses from the same address space as used on the connected LAN.



Let's look at the example setup from the image above. Host A (172.16.1.2) on Subnet A wants to send packets to Host D (172.16.2.3) on Subnet B. Host A has a /16 subnet mask which means that Host A believes that it is directly connected to all 172.16.0.0/16 network (the same LAN). Since the Host A believes that is directly connected it sends an ARP request to the destination to clarify the MAC address of Host D. (in the case when Host A finds that destination IP address is not from the same subnet it sends a packet to the default gateway.). Host A broadcasts an ARP request on Subnet A.

Info from packet analyzer software:

```
 No.     Time    Source             Destination       Protocol  Info

 12   5.133205  00:1b:38:24:fc:13  ff:ff:ff:ff:ff:ff  ARP      Who has 173.16.2.3?  Tell 173.16.1.2


Packet details:

Ethernet II, Src: (00:1b:38:24:fc:13), Dst: (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    Source: (00:1b:38:24:fc:13)
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    [Is gratuitous: False]
    Sender MAC address: 00:1b:38:24:fc:13
    Sender IP address: 173.16.1.2
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 173.16.2.3
```

With this ARP request, Host A (172.16.1.2) is asking Host D (172.16.2.3) to send its MAC address. The ARP request packet is then encapsulated in an Ethernet frame with the MAC address of Host A as the source address and a broadcast (FF:FF:FF:FF:FF:FF) as the destination address. Layer 2 broadcast means that frame will be sent to all hosts in the same layer 2 broadcast domain which includes the ether0 interface of the router, but does not reach Host D, because router by default does not forward layer 2 broadcasts.

Since the router knows that the target address (172.16.2.3) is on another subnet but it can reach Host D, it replies with its own MAC address to Host A.

```
No.     Time    Source            Destination         Protocol   Info

13   5.133378  00:0c:42:52:2e:cf  00:1b:38:24:fc:13   ARP        172.16.2.3 is at 00:0c:42:52:2e:cf

Packet details:

Ethernet II, Src: 00:0c:42:52:2e:cf, Dst: 00:1b:38:24:fc:13
   Destination: 00:1b:38:24:fc:13
   Source: 00:0c:42:52:2e:cf
   Type: ARP (0x0806)
Address Resolution Protocol (reply)
   Hardware type: Ethernet (0x0001)
   Protocol type: IP (0x0800)
   Hardware size: 6
   Protocol size: 4
   Opcode: reply (0x0002)
   [Is gratuitous: False]
   Sender MAC address: 00:0c:42:52:2e:cf
   Sender IP address: 172.16.1.254
   Target MAC address: 00:1b:38:24:fc:13
   Target IP address: 172.16.1.2
```

This is the Proxy ARP reply that the router sends to Host A. Router sends back unicast proxy ARP reply with its own MAC address as the source address and the MAC address of Host A as the destination address, by saying "send these packets to me, and I'll get it to where it needs to go."

When Host A receives ARP response it updates its ARP table, as shown:

```
C:\Users\And>arp -a
Interface: 173.16.2.1 --- 0x8
  Internet Address      Physical Address      Type
  173.16.1.254          00-0c-42-52-2e-cf    dynamic
  173.16.2.3            00-0c-42-52-2e-cf    dynamic
  173.16.2.2            00-0c-42-52-2e-cf    dynamic
```

After MAC table update, Host A forwards all the packets intended for Host D (172.16.2.3) directly to router interface ether0 (00:0c:42:52:2e:cf) and the router forwards packets to Host D. The ARP cache on the hosts in Subnet A is populated with the MAC address of the router for all the hosts on Subnet B. Hence, all packets destined to Subnet B are sent to the router. The router forwards those packets to the hosts in Subnet B.

Multiple IP addresses by the host are mapped to a single MAC address (the MAC address of this router) when proxy ARP is used.

Proxy ARP can be enabled on each interface individually with command *arp=proxy-arp*:

```
 [admin@MikroTik] /interface ethernet> set 1 arp=proxy-arp
 [admin@MikroTik] /interface ethernet> print
 Flags: X - disabled, R - running
   #    NAME                MTU   MAC-ADDRESS         ARP
   0  R ether1              1500  00:30:4F:0B:7B:C1 enabled
   1  R ether2              1500  00:30:4F:06:62:12 proxy-arp
 [admin@MikroTik] interface ethernet>
```

Local Proxy ARP

if the arp property is set to *local-proxy-arp* on an interface, then the router performs proxy ARP to/from this interface only. I.e. for traffic that comes in and goes out of the same interface. In a normal LAN, the default behavior is for two network hosts to communicate directly with each other, without involving the router.

This is done to support (Ethernet) switch features, like RFC 3069, where the individual ports are **NOT** allowed to communicate with each other, but they are allowed to talk to the upstream router. As described in RFC 3069, it is possible to allow these hosts to communicate through the upstream router by proxy_arp'ing. Don't need to be used together with proxy_arp. This technology is known by different names:

- In RFC 3069 it is called VLAN Aggregation;
- Cisco and Allied Telesis call it Private VLAN;
- Hewlett-Packard calls it Source-Port filtering or port-isolation;
- Ericsson calls it MAC-Forced Forwarding (RFC Draft).

TCP/IP

TCP Session Establishment and Termination

TCP is a connection-oriented protocol. The difference between a connection-oriented protocol and a connection-less protocol is that a connection-oriented protocol does not send any data until a proper connection is established.

TCP uses **a three-way handshake** whenever the transmitting device tries to establish a connection to the remote node. As a result end-to-end virtual (logical) circuit is created where flow control and acknowledgment for reliable delivery are used. TCP has several message types used in connection establishment and termination process.

Connection establishment process

1. The host A who needs to initialize a connection sends out an **SYN** (Synchronize) packet with a proposed initial sequence number to the destination "host B";
2. When the host B receives an **SYN** message, it returns a packet with both **SYN** and **ACK** flags set in the TCP header (SYN-ACK);
3. When the host A receives the SYN-ACK, it sends back the **ACK** (Acknowledgment) packet;
4. Host B receives **ACK** and at this stage, the connection is **ESTABLISHED;**

Connection-oriented protocol services are often sending acknowledgments (ACKs) after successful delivery. After the packet with data is transmitted, the sender waits for acknowledgment from the receiver. If time expires and the sender did not receive ACK, a packet is retransmitted.

Connection termination

When the data transmission is complete and the host wants to terminate the connection, the termination process is initiated. Unlike TCP Connection establishment, which uses a three-way handshake, connection termination uses four-way massages. A connection is terminated when both sides have finished the shutdown procedure by sending a FIN (finish) and receiving an ACK (Acknowledgment).

1. The host A, who needs to terminate the connection, sends a special message with the **FIN** flag, indicating that it has finished sending the data;
2. The host B, who receives the **FIN** segment, does not terminate the connection but enters into a "passive close" (CLOSE_WAIT) state and sends the **ACK** for the **FIN** back to the host A. If host B does not have any data to transmit to the host A it will also send the **FIN** message. Now the host B enters into LAST_ACK state. At this point host B will no longer accept data from host A, but can continue to transmit data to host A.
3. When the host A receives the last **FIN** from the host B, it enters into a (TIME_WAIT) state, and sends an **ACK** back to the host B;
4. Host B gets the **ACK** from the host A and connection is terminated;

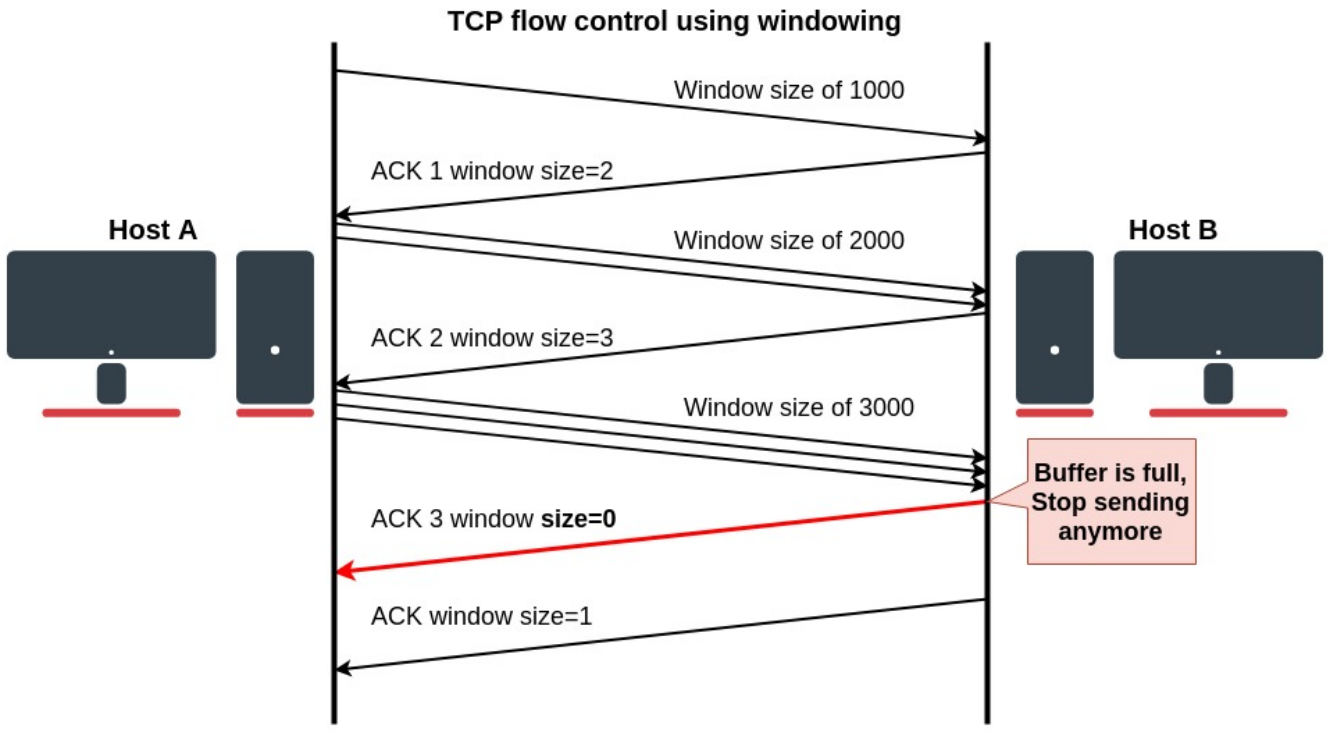
TCP Segments transmission (windowing)

Now that we know how the TCP connection is established we need to understand how data transmission is managed and maintained. In TCP/IP networks transmission between hosts is handled by TCP protocol.

Let's think about what happens when data-grams are sent out faster than the receiving device can process. The receiver stores them in memory called a buffer. But since buffer space is not unlimited, when its capacity is exceeded receiver starts to drop the frames. All dropped frames must be re-transmitted again which is the reason for low transmission performance.

To address this problem, TCP uses a flow control protocol. The window mechanism is used to control the flow of the data. When a connection is established, the receiver specifies the window field in each TCP frame. Window size represents the amount of received data that the receiver is willing to store in the buffer. Window size (in bytes) is sent together with acknowledgments to the sender. So the size of the window controls how much information can be transmitted from one host to another without receiving an acknowledgment. The sender will send only the amount of bytes specified in window size and then will wait for acknowledgments with updated window size.

If the receiving application can process data as quickly as it arrives from the sender, then the receiver will send a positive window advertisement (increase the size of the window) with each acknowledgment. It works until the sender becomes faster than the receiver and incoming data will eventually fill the receiver's buffer, causing the receiver to advertise acknowledgment with a zero window. A sender that receives a zero window advertisement must stop transmit until it receives a positive window. Let's take a look at the illustrated windowing process:



1. The "host A" starts to transmit with a window size of 1000, one 1000byte frame is transmitted;
2. Receiver "host B" returns ACK with window size to increase to 2000;

3. The host A receives ACK and transmits two frames (1000 bytes each);
4. After that, the receiver advertises an initial window size to 3000. Now sender transmits three frames and waits for an acknowledgement;
5. The first three segments fill the receiver's buffer faster than the receiving application can process the data, so the advertised window size reaches zero indicating that it is necessary to wait before further transmission is possible;
6. The size of the window and how fast to increase or decrease the window size is available in various TCP congestion avoidance algorithms such as Reno, Vegas, Tahoe, etc;

IP Addressing

Overview

IP addresses serve for general host identification purposes in IP networks (RFC 791). A typical (IPv4) address consists of four octets. For proper addressing the router also needs the network mask value, id est which bits of the complete IP address refer to the address of the host, and which - to the address of the network. The network address value is calculated by binary AND operation from a network mask and IP address values. It's also possible to specify an IP address followed by a slash "/" and the number of bits that form the network address.

In most cases, it is enough to specify the address, the netmask, and the interface arguments. The network prefix and the broadcast address are calculated automatically.

It is possible to add multiple IP addresses to an interface or to leave the interface without any addresses assigned to it. In the case of bridging or PPPoE connection, the physical interface may not have any address assigned, yet be perfectly usable. Configuring an IP address to a physical interface included in a bridge would mean actually setting it on the bridge interface itself.

You can use */ip address print detail* to see which interface the address belongs to.

IPv4 Addressing

IPv4 uses 4-byte addresses which are segmented in four 8-bit fields called octets. Each octet is converted to a decimal format and separated by a dot. For example:

```
11000000 10101000 00000011 00011000 => 192.168.3.24
```

The IPv4 network consists of three addresses:

- **network address** - a standard way to refer to an IPv4 address assigned to a network. For example, we could refer to the network 192.168.1.0 or 172.16.0.0 as a “Network Address.”
- **broadcast address** - a special address for each network that allows communication to all the hosts in that network. The broadcast address uses the highest address in the network range. for example, broadcast address if 192.168.1.0/24 network will be 192.168.1.255
- **host address** - any other address that is not a network address and broadcast address can be used as a host address. For example, 192.168.1.2 - 254 host addresses can be used from 192.168.1.0/24 address range

There are several types of IP addressing

- **unicast** - normally refers to a single sender or a single receiver, and can be used for both sending and receiving. Usually, a unicast address is associated with a single device or host, but it is not a one-to-one correspondence.

- **broadcast** - address to send data to all possible destinations ("all-hosts broadcast"), which permits the sender to send the data only once, and all receivers receive a copy of it. In the IPv4 protocol, the address *255.255.255.255* is used for local broadcast. In addition, a directed (limited) broadcast can be made by combining the network prefix with a host suffix composed entirely of binary 1s. For example, the destination address used for directed broadcast to devices on the 192.0.2.0/24 network is 192.0.2.255
- **multicast** - address associated with a group of interested receivers. In IPv4, addresses 224.0.0.0 through 239.255.255.255 are designated as multicast addresses. The sender sends a single datagram from its unicast address to the multicast group address and the intermediary routers take care of making copies and sending them to all receivers that have joined the corresponding multicast group.

Private Address Range

The following IP address ranges are reserved (RFC 6890) for private addressing. These addresses are not routed in the global routing table and should be translated to global addresses with network address translation (NAT):

- 10.0.0.0/8 - start: 10.0.0.0; end: 10.255.255.255
- 172.16.0.0/12 - start: 172.16.0.0; end:172.31.255.255
- 192.168.0.0/16 - start: 192.168.0.0; end: 192.168.255.255

Other Reserved Address Ranges

- 198.18.0.0/15 - benchmarking
- 192.88.99.0/24 - 6to4 relay anycast address range
- 192.0.2.0/24, 198.51.100.0/24, 203.0.113.0/24 - documentation
- 169.254.0.0/16 - auto-configuration address range

Adding IP Address

Consider a setup where two routers are directly connected with the cable and we do not want to waste address space:

R1 configuration:

```
/ip address
add address=10.1.1.1/32 interface=ether1 network=172.16.1.1
```

R2 configuration:

```
/ip address
add address=172.16.1.1/32 interface=ether1 network=10.1.1.1
```

IPv6 Addressing

Internet Protocol version 6 (IPv6) is the newer version of the Internet Protocol (IP). It was initially expected to replace IPv4 in a short enough time, but for now, it seems that these two versions will coexist on the Internet in foreseeable future. Nevertheless, IPv6 becomes more important, as the date of the unallocated IPv4 address pool's exhaustion approaches.

The two main benefits of IPv6 over IPv4 are:

- much larger address space;
- support of stateless and stateful address auto-configuration;
- built-in security;
- new header format (faster forwarding).

IPv6 uses 16 bytes addresses compared to 4-byte addresses in IPv4. IPv6 address syntax and types are described in RFC 4291.

There are multiple IPv6 address types, that can be recognized by their prefix. RouterOS distinguishes the following:

- multicast (with prefix ff00::/8)
- link-local (with prefix fe80::/10)
- unique local addresses (with prefix fc00::/7)
- loopback (the address::1/128)
- unspecified (the address::/128)

- other (all other addresses, including the obsoleted site-local addresses, and RFC 4193 unique local addresses; they all are treated as global unicast).

Properties

Property	Description
address (*Address/Netmask*; Default:)	Ipv6 address. Allowed netmask range is 0..128. Address can also be constructed from the pool if **from-pool** property is specified. For example if address is set to ::1/64 then address will be constructed as follows <prefix_from_pool>::1/64
advertise (*yes* | *no*; Default: **no**)	Whether to enable stateless address configuration. The prefix of that address is automatically advertised three times to hosts using ICMPv6 protocol. The option is set by default for addresses with prefix length 64. If address is removed or changed, then old prefix will be deprecated by automatically advertising the old prefix with lifetime set to "0s" three times to hosts using ICMPv6 protocol
comment comment **(*string*; Default:)**	Descriptive name of an item
disabled (*yes* | *no*; Default: **no**)	Whether address is disabled or not. By default it is not disabled
eui-64 (*yes* | *no*; Default: **no**)	Whether to calculate EUI-64 address and use it as last 64 bits of the IPv6 address.
from-pool (*string*; Default:)	Name of the pool from which prefix will be taken to construct IPv6 address taking last part of the address from **address** property.
no-dad (*yes* | *no*; Default: **no**)	If enabled (yes) - disables Duplicate Address Detection (DAD) for IPv6 addresses on an interface. This can be useful in scenarios where you want to assign static IPv6 addresses to devices and avoid the delay caused by DAD.
interface (*string*; Default:)	Name of an interface on which Ipv6 address is set.

Read-only properties

Property	Description
actual-interface (*string*)	Actual interface on which address is set up. For example, if address was configured on ethernet interface and ethernet interface was added to bridge, then actual interface is bridge not ethernet.
dynamic (*yes* | *no*)	Whether address is dynamically created
global (*yes* | *no*)	Whether address is global
invalid (*yes* | *no*)	Whether address is invalid
link-local (*yes* | *no*)	Whether address is link local

One difference between IPv6 and IPv4 addresses is that IPv6 automatically generates a **link-local** IPv6 address for each active interface that has IPv6 support.

IPv6 addresses are represented a little bit differently than IPv4 addresses. For IPv6, the 128-bit address is divided into eight 16-bit blocks, and each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. The resulting representation is called colon-hexadecimal.

In the example below IPv6 address in binary format is converted to a colon-hexadecimal representation

```
0010000000000001 0000010001110000 0001111100001001 0000000100110001
0000000000000000 0000000000000000 0000000000000000 0000000000001001
```

```
2001:0470:1f09:0131:0000:0000:0000:0009
```

The IPv6 address can be further simplified by removing leading zeros in each block:

```
2001:470:1f09:131:0:0:0:9
```

As you can see IPv6 addresses can have long sequences of zeros. This contiguous sequence can be compressed to **::**

```
2001:470:1f09:131::9
```


Zero compression can only be used once. Otherwise, you could not determine the number of 0 bits represented by each instance of a double-colon

IPv6 prefix is written in **address/prefix-length** format. Compared to IPv4 decimal representation of a network mask cannot be used. Prefix examples:

```
2001:470:1f09:131::/64
2001:db8:1234::/48
2607:f580::/32
2000::/3
```

Address Types

Several IPv6 address types exist:

- Unicast
- Anycast
- Multicast

As you can see there are no Broadcast addresses in the IPv6 network, compared to the IPv4 broadcast functionality was completely replaced with multicast.

Unicast Addresses

Packets addressed to a unicast address are delivered only to a single interface. To this group belong:

- globally unique addresses and can be used to connect to addresses with global scope anywhere;
- link-local addresses;
- unique local addresses (ULA RFC4193)
- site-local addresses (FEC0::/48) - deprecated;
- special-purpose addresses;
- compatibility addresses;

A global unicast address can be automatically assigned to the node by **Stateless Address auto-configuration.**

Link-local Address

A link-local address is required on every IPv6-enabled interface, applications may rely on the existence of a link-local address even when there is no IPv6 routing, that is why the link-local address is generated automatically for every active interface using its interface identifier (calculated EUI-64 from MAC address if present). The address prefix is always **FE80::/64** and IPv6 router never forwards link-local traffic beyond the link.

These addresses are comparable to the auto-configuration addresses 169.254.0.0/16 of IPv4.

A link-local address is also required for IPv6 Neighbor Discovery processes.

> If the interface is set as a bridge port, an interface-specific link-local address is removed leaving only the bridge link-local address

Unique Local Address

Unique Local Address (ULA) is reserved for local use in the home and enterprise environments not routed in public address space and is equivalent to IPv4 private address ranges.

The reserved address range is **fc00::/7**

Special Purpose Address

Address	Description
Unspecified address (:: /128)	Never assigned to an interface or used as a destination address, used only to indicate the absence of an address. Equivalent to IPv4 0.0.0.0 address.
loopback address (::1 /128)	Used to identify a loopback interface, enabling a node to send packets to itself. It is equivalent to the IPv4 loopback address of 127.0.0.1.
2002::/16	This prefix is used for 6to4 addressing. Here, an address from the IPv4 network 192.88.99.0/24 is also used.
2001:db8::/32	Address range reserved for documentation. These should never be seen as the source or destination.
2001:0010::/28	Orchid fixed term experiment. Should not be seen as a source or destination
2001:0002::/48	Used for benchmarking, should not be seen as source or destination
2001:0000::/32	Teredo

Compatibility Address

Address	Description
IPv4 compatible address	used by dual-stack nodes that are communicating with IPv6 over an IPv4 infrastructure. When the IPv4-compatible address is used as an IPv6 destination, IPv6 traffic is automatically encapsulated with an IPv4 header and sent to the destination by using the IPv4 infrastructure. The address is written in the following format `::w.x.y.z`, where w.x.y.z is the dotted decimal representation of a public IPv4 address.
IPv4 mapped address	used to represent an IPv4-only node to an IPv6 node. It is used only for internal representation. The IPv4-mapped address is never used as a source or destination address for an IPv6 packet. The IPv6 protocol does not support the use of IPv4-mapped addresses. The address is written in the following format: `::ffff:w.x.y.z`, where `w.x.y.z` is the dotted-decimal representation of a public IPv4 address.

Multicast Address

The most important multicast aspects are:

- traffic is sent to a single address but is processed by multiple hosts;
- group membership is dynamic, allowing hosts to join and leave the group at any time;
- in IPv6, Multicast Listener Discovery (MLD) messages are used to determine group membership on a network segment, also known as a link or subnet;
- a host can send traffic to the group's address without belonging to the corresponding group.

A single IPv6 multicast address identifies each multicast group. Each group's reserved IPv6 address is shared by all host members of the group who listen and receive any IPv6 messages sent to the group's address.

The multicast address consists of the following parts:

- The first 8 bits in the multicast address are always 1111 1111 (which is FF in hexadecimal format).

- The flag uses the 9th to 12th bit and shows if this multicast address is predefined (well-known) or not. If it is well-known, all bits are 0s.
- Scope ID indicates to which scope multicast address belongs, for example, Scope ID=2 is link-local scope.
- The group ID is used to specify a multicast group. There are predefined group IDs, such as Group ID=1 - all nodes. Therefore, if the multicast address is ff02::1, that means Scope ID=2 and Group ID=1, indicating all nodes in link-local scope. This is analogous to broadcast on IPv4.

Here is the table of reserved IPV6 addresses for multicast:

Address	Description
FF02::1	The all-nodes address is used to reach all nodes on the same link.
FF02::2	The all-routers address is used to reach all routers on the same link.
FF02::5	The all-Open Shortest Path First (OSPF) router address is used to reach all OSPF routers on the same link.
FF02::6	The all-OSPF-designated router's address is used to reach all OSPF-designated routers on the same link.
FF02::1:FFXX:XXXX	The solicited-node address is used in the address resolution process to resolve the IPv6 address of a link-local node to its link-layer address. The last 24 bits (XX:XXXX) of the solicited-node address are the last 24 bits of an IPv6 unicast address.

The following table is a partial list of IPv6 multicast addresses that are reserved for IPv6 multicasting and registered with the Internet Assigned Numbers Authority (IANA). For a complete list of assigned addresses read IANA document.

Multicast addresses can be used to discover nodes in a network. For example, discover all nodes

```
mrz@bumba:/media/aaa/ver$ ping6 ff02::1%eth0
PING ff02::1%eth0(ff02::1) 56 data bytes
64 bytes from fe80::21a:4dff:fe5d:8e56: icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from fe80::20c:42ff:fe0d:2c38: icmp_seq=1 ttl=64 time=4.03 ms (DUP!)
64 bytes from fe80::20c:42ff:fe28:7945: icmp_seq=1 ttl=64 time=5.59 ms (DUP!)
64 bytes from fe80::20c:42ff:fe49:fce5: icmp_seq=1 ttl=64 time=5.60 ms (DUP!)
64 bytes from fe80::20c:42ff:fe21:f1ec: icmp_seq=1 ttl=64 time=5.88 ms (DUP!)
64 bytes from fe80::20c:42ff:fe72:a1b0: icmp_seq=1 ttl=64 time=6.70 ms (DUP!)
```

discover all routers

```
mrz@bumba:/media/aaa/ver$ ping6 ff02::2%eth0
PING ff02::2%eth0(ff02::2) 56 data bytes
64 bytes from fe80::20c:42ff:fe28:7945: icmp_seq=1 ttl=64 time=0.672 ms
64 bytes from fe80::20c:42ff:fe0d:2c38: icmp_seq=1 ttl=64 time=1.44 ms (DUP!)
```

Anycast Address

An anycast address is a new type of address incorporated in IPv6.

Anycasting is a new networking paradigm supporting service-oriented Addresses where an identical address can be assigned to multiple nodes providing a specific service. An anycast packet (i.e., one with an anycast destination address) is delivered to one of these nodes with the same anycast address.

An anycast address is not assigned a specific address range. It is assigned from the unicast address range.

Interface Identifier

The last 64 bits of an IPv6 address are the interface identifier that is unique to the 64-bit prefix of the IPv6 address. There are several ways how to determine interface identifier:

- EUI-64;
- randomly generated to provide a level of anonymity;

- manually configured.

EUI-64

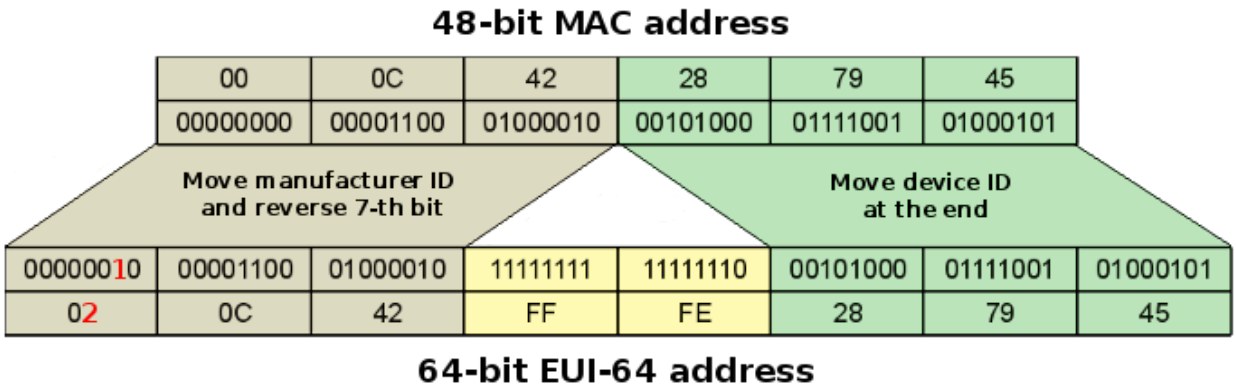
Traditional interface identifiers for network adapters are 48-bit MAC addresses. This address consists of a 24-bit manufacturer ID and a 24-bit board ID.

IEEE EUI-64 is a new standard for network interface addresses. The company ID is still 24 bits in length, but the extension ID is 40 bits, creating a much larger address space for network adapters.

To create a EUI-64 address from the interface MAC address:

- 0xFFFE is inserted into the MAC address between the manufacturer ID and the board ID.
- The seventh bit of the first byte is reversed.

Let's make an example with the following MAC address 00:0C:42:28:79:45.



The image above illustrates the conversion process. When the result is converted to colon-hexadecimal notation, we get the interface identifier `20C:42FF:FE28:7945`. As a result, the corresponding link-local address is

```
FE80::20C:42FF:FE28:7945/64
```

In RouterOS, if the EUI-64 parameter of an address is configured, the last 64 bits of that address will be automatically generated and updated using interface identifier. The last bits must be configured to be zero for this case. Example:

```
[admin@MikroTik] > ipv6 address add address=fc00:3::/64 interface=ether3 eui-64=yes
[admin@MikroTik] > ipv6 address print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
#    ADDRESS                                     INTERFACE                   ADVERTISE
...
5  G fc00:3::20c:42ff:fe1d:3d4/64                ether3                      yes
[admin@MikroTik] > interface ethernet set ether3 mac-address=10:00:00:00:00:01
[admin@MikroTik] > ipv6 address print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
#    ADDRESS                                     INTERFACE                   ADVERTISE
...
5  G fc00:3::1200:ff:fe00:1/64                   ether3                      yes
```

Configuring IPv6 Address

This example shows how to set up simple addressing with global IPv6 addresses between two routers.

R1 configuration:

```
/ipv6 address
add address=2001:DB8::1/64 interface=ether1 advertise=no
```

R2 configuration:

```
/ipv6 address
add address=2001:DB8::2/64 interface=ether1 advertise=no
```

Check the address list:

```
[admin@R1] /ipv6 address> print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
#    ADDRESS                                     FROM-POOL INTERFACE     ADVERTISE
0  G 2001:db8::1/64                                      ether1        no
3 DL fe80::219:d1ff:fe39:3535/64                         ether1        no
```

Notice that our added address has a G flag indicating that this address can be globally routed. We also have a link-local address on the interface which is created automatically for every IPv6-capable interface.

Test connectivity:

```
[admin@R1] /ipv6 address> /ping 2001:DB8::2
HOST                                     SIZE TTL TIME  STATUS
2001:db8::2                56  64 12ms  echo reply
2001:db8::2                56  64 0ms   echo reply
    sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=6ms max-rtt=12ms
```

SLAAC IPv6 Address

If under IPv6/Settings menu "accept-router-advertisements" option is enabled and the router receives a Router Advertisement packet, then the SLAAC IPv6 address will be automatically assigned to the interface on which the advertisements were received. This address will have DG flags meaning that the address is dynamic and global. Such addresses will show valid and lifetime parameters.

```
[admin@R1] /ipv6/address/print detail where dynamic && global
Flags: X - disabled, I - invalid, D - dynamic; G - global, L - link-local
 0 DG address=2001:db8::::ba69:f4ff:fe84:545/64 from-pool="" interface=ether1
      actual-interface=test_fp eui-64=no advertise=no no-dad=no valid=4w2d
      preferred=1w
```

If SLAAC addresses are accepted, then also dynamic route toward the Internet will be generated. It will also contain a few limitations if specified on the advertisement packet. For example, hop-limit and MTU. If multiple addresses are received on the same interface, then the lowest of the MTU values per interface will be used.

```
[admin@R1] /routing/route/print detail where slaac
Flags: X - disabled, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, a - ldp-address, l - ldp-
mapping, g - slaac, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp, B - blackhole
 Ag + afi=ip6 contribution=active dst-address=::/0 routing-table=main
       pref-src="" gateway=fe80::ba69:f4ff:fe84:7b2%ether1
       immediate-gw=fe80::ba69:f4ff:fe84:7b2%ether1 distance=1 scope=30
       target-scope=10 belongs-to="slaac" mtu=1400 hoplimit=10
       debug.fwp-ptr=0x201C2C00
```

IPv6 Neighbor Discovery

Summary

Standards: `RFC 2462, RFC 2461, RFC 4861`

RouterOS has IPv6 Neighbor Discovery and stateless address autoconfiguration support using Router Advertisement Daemon (RADVD).

Node description

Node is a device that implements IPv6. In IPv6 networks nodes are divided into two types:

- **Routers** - a node that forwards IPv6 packets not explicitly addressed to itself.
- **Hosts** - any node that is not a router.

Routers and hosts are strictly separated, meaning that routers cannot be hosts and hosts cannot be routers at the same time.

Stateless address autoconfiguration

There are several types of autoconfiguration:

- *stateless* - address configuration is done by receiving Router Advertisement messages. These messages include stateless address prefixes and require that host is not using stateful address configuration protocol.
- *stateful* - address configuration is done by using the stateful address configuration protocol (DHCPv6). The stateful protocol is used if RA messages do not include address prefixes.
- *both* - RA messages include stateless address prefixes and require that hosts use a stateful address configuration protocol.

A highly useful feature of IPv6 is the ability to automatically configure itself without the use of a stateful configuration protocol like DHCP (See example).

> Address autoconfiguration can only be performed on multicast-capable interfaces.

It is called stateless address autoconfiguration since there is no need to manage the state on the router side. It is a very simple, robust, and effective autoconfiguration mechanism.

RouterOS uses RADVD to periodically advertise information about the link to all nodes on the same link. The information is carried by ICMPv6 "router advertisement" packet, and includes the following fields:

- IPv6 subnet prefix
- Default router link-local address
- Other parameters that may be optional: are link MTU, default hop limit, and router lifetime.

Then host catches the advertisement, and configures the global IPv6 address and the default router. Global IPv6 address is generated from the advertised **subnet prefix** and EUI-64 interface identifier.

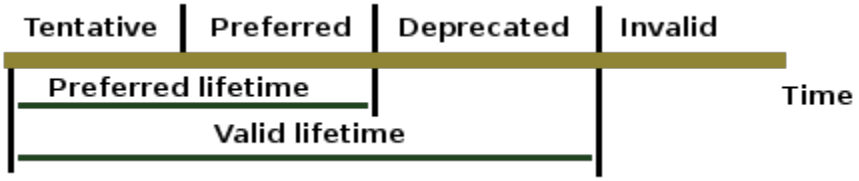
Optionally, the host can ask for an advertisement from the router by sending an ICMPv6 "router solicitation" packet. On Linux **rtsol** utility transmits the router solicitation packet. If you are running a mobile node, you may want to transmit router solicitations periodically.

Address states

When an auto-configuration address is assigned it can be in one of the following states:

- ***tentative*** - in this state host verifies that the address is unique. Verification occurs through duplicate address detection.
- ***preferred*** - at this state address is verified as unique and the node can send and receive unicast traffic to and from a preferred address. The period of time of the preferred state is included in the RA message.
- ***deprecated*** - the address is still valid, but is not used for new connections.
- ***invalid*** - node can no longer send or receive unicast traffic. An address enters the invalid state after the valid lifetime expires.

The image above illustrates the relation between states and lifetimes.



Neighbor discovery

Sub-menu: `/ipv6 nd`

In this submenu, IPv6 Neighbor Discovery (ND) protocol is configured.

Neighbor Discovery (ND) is a set of messages and processes that determine relationships between neighboring nodes. ND, compared to IPv4, replaces Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP) Router Discovery, and ICMP Redirect and provides additional functionality.

ND is used by hosts to:

- Discover neighboring routers.
- Discover addresses, address prefixes, and other configuration parameters.

ND is used by routers to:

- Advertise their presence, host configuration parameters, and on-link prefixes.
- Inform hosts of a better next-hop address to forward packets to a specific destination.

ND is used by nodes to:

- Both resolve the link-layer address of a neighboring node to which an IPv6 packet is being forwarded and determine when the link-layer address of a neighboring node has changed.
- Determine whether IPv6 packets can be sent to and received from a neighbor.

Properties

Property	Description
advertise-dns (*yes* | *no*; Default: **yes**)	Option to redistribute DNS server information using RADVD. You will need a running client-side software with Router Advertisement DNS support to take advantage of the advertised DNS information. `Read more >>`
advertise-mac-address (*yes* | *no* ; Default: **yes**)	When set, the link-layer address of the outgoing interface is included in the RA.
comment (*string*; Default:)	Descriptive name of an item
dns-servers (*unspecified* | *ipv6 addresses*; Default: **unspecified**)	Specify a single IPv6 address or list of addresses that will be provided to hosts for DNS server configuration.
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled or not. By default, entry is enabled.

hop-limit (*unspecified* | *integer*[*0..255*]; Default: **unspecified**)	The default value that should be placed in the Hop Count field of the IP header for outgoing (unicast) IP packets.
interface (*all* | *string*; Default:)	The interface on which to run neighbor discovery. • all - run ND on all running interfaces.
managed-address-configuration (*yes* | *no*; Default: **no**)	The flag indicates whether hosts should use stateful autoconfiguration (DHCPv6) to obtain addresses.
mtu (*unspecified* | *integer*[*0..4294967295*]; Default: **unspecified**)	The MTU option is used in router advertisement messages to ensure that all nodes on a link use the same MTU value in those cases where the link MTU is not well known. • **unspecified** - do not send the MTU option.
other-configuration (*yes* | *no*; Default: **no**)	The flag indicates whether hosts should use stateful autoconfiguration to obtain additional information (excluding addresses).
pref64-prefixes (*unspecified* | *ipv6 prefixes*; Default: **unspecified**)	Specify IPv6 prefix or list of prefixes within /32, /40. /48, /56, /64, or /96 subnet that will be provided to hosts as NAT64 prefixes.
ra-delay (*time*; Default: **3s**)	The minimum time allowed between sending multicast router advertisements from the interface.
ra-interval (*time*[*3s..20m50s*]-*time*[*4s..30m*]; Default: **3m20s-10m**)	The min-max interval allowed between sending unsolicited multicast router advertisements from the interface.
ra-preference (*low* | *medium* | *high*; Default: **medium**)	Specify the router preference that is communicated to IPv6 hosts through router advertisements. The `preference` value in the router advertisements enables IPv6 hosts to select a default router to reach a remote destination
ra-lifetime (*none* | *time*; Default: **30m**)	Sets the RA lifetime. A Lifetime of 0 indicates that the router is not a default router.(see Section 6.2.3 of RFC 4861)
reachable-time (*unspecified* | *time*[*0..1h*]; Default: **unspecified**)	The time that a node assumes a neighbor is reachable after having received a reachability confirmation. Used by the Neighbor Unreachability Detection algorithm (see Section 7.3 of RFC 2461)
retransmit-interval (*unspecified* | *time*; Default: **unspecified**)	The time between retransmitted Neighbor Solicitation messages. Used by address resolution and the Neighbor Unreachability Detection algorithm (see Sections 7.2 and 7.3 of RFC 2461)

If ND is automatically generated by LTE configuration, then the maximum lifetime for RA will be capped at 1 hour.

Prefix

Sub-menu: `/ipv6 nd prefix`

Prefix information sent in RA messages used by stateless address auto-configuration.

Note: The autoconfiguration process applies only to hosts and not routers.

Properties

Property	Description
6to4-interface (*none* | *string*; Default:)	If this option is specified, this prefix will be combined with the IPv4 address of the interface name to produce a valid 6to4 prefix. The first 16 bits of this prefix will be replaced by 2002 and the next 32 bits of this prefix will be replaced by the IPv4 address assigned to the interface name at configuration time. The remaining 80 bits of the prefix (including the SLA ID) will be advertised as specified in the configuration file.
autonomous (*yes* | *no*; Default: **yes**)	When set, indicates that this prefix can be used for autonomous address configuration. Otherwise, prefix information is silently ignored.

comment (*string*; Default:)	Descriptive name of an item
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled or not. By default, entry is enabled.
on-link (*yes* | *no*; Default: **yes**)	When set, indicates that this prefix can be used for on-link determination. When not set the advertisement makes no statement about the on-link or off-link properties of the prefix. For instance, the prefix might be used for address configuration with some of the addresses belonging to the prefix being on-link and others being off-link.
preferred-lifetime (*infinity* | *time*; Default: **1w**)	Timeframe (relative to the time the packet is sent) after which generated address becomes "deprecated". Deprecated is used only for already existing connections and is usable until valid lifetime expires. `Read more >>`
prefix (*ipv6 prefix*; Default: **::/64**)	A prefix from which stateless address autoconfiguration generates the valid address.
valid-lifetime (*infinity* | *time*; Default: **4w2d**)	The length of time (relative to the time the packet is sent) an address remains in the valid state. The valid lifetime must be greater than or equal to the preferred lifetime. `Read more >>`
interface (*string*; Default:)	Interface name on which stateless auto-configuration will be running.

Neighbors List

Sub-menu: `/ipv6 neighbor`

List of all discovered nodes by IPv6 neighbor discovery protocol (neighbor cache).

Read-only Properties

Property	Description
address (*ipv6 address*)	link-local address of the node.
comment (*string*)	
interface (*string*)	The interface on which the node was detected.
mac-address (*string*)	Mac address of the discovered node.
router (*yes* | *no*)	Whether the discovered node is a router

status (*noarp* | *incomplete* | *stale* | *reachable* | *delay* | *probe*)	Status of the cached entry: • **noarp** - the neighbor entry is valid. No attempts to validate this entry will be made but it can be removed when its lifetime expires • **incomplete** - address resolution is in progress and the link-layer address of the neighbor has not yet been determined; • **reachable** - the neighbor is known to have been reachable recently (within tens of seconds ago); • **stale** - the neighbor is no longer known to be reachable but until traffic is sent to the neighbor, no attempt should be made to verify its reachability; • **delay** - the neighbor is no longer known to be reachable, and traffic has recently been sent to the neighbor, probes are delayed for a short period in order to give upper layer protocol a chance to provide reachability confirmation; • **probe** - the neighbor is no longer known to be reachable, and unicast Neighbor Solicitation probes are being sent to verify reachability.

Examples

Stateless autoconfiguration example

```
[admin@MikroTik] > ipv6 address print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
# ADDRESS INTERFACE ADVERTISE
0 G 2001:db8::1/64 ether1 yes
```

As an example, the **advertise** flag is enabled which indicates that dynamic`/ipv6 nd prefix`entry is added.

```
[admin@MikroTik] > ipv6 nd prefix print
Flags: X - disabled, I - invalid, D - dynamic
0 D prefix=2001:db8::/64 interface=ether1 on-link=yes autonomous=yes
 valid-lifetime=4w2d preferred-lifetime=1w
```

On a host that is directly attached to the router, we see that an address was added. The address consists of the prefix part (first 64 bits) that takes the prefix from the prefix advertisement, and the host part (last 64 bits) that is automatically generated from the local MAC address:

```
atis@atis-desktop:~$ ip -6 addr
 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
inet6 2001:db8::21a:4dff:fe56:1f4d/64 scope global dynamic
 valid_lft 2588363sec preferred_lft 601163sec
inet6 fe80::21a:4dff:fe56:1f4d/64 scope link
valid_lft forever preferred_lft forever
```

The host has received the *2001:db8::/64* prefix from the router and configured an address with it.

There is also an option to redistribute DNS server information using RADVD:

```
[admin@MikroTik] > ip dns set server=2001:db8::2
[admin@MikroTik] > ip dns print servers: 2001:db8::2
 ...
[admin@MikroTik] > ipv6 nd set [f] advertise-dns=yes
```

You will need a running client-side software with Router Advertisement DNS support to take advantage of the advertised DNS information.

On Ubuntu/Debian Linux distributions you can install **rdnssd** package which is capable of receiving the advertised DNS addresses.

```
mrz@bumba:/$ sudo apt-get install rdnssd
```

```
mrz@bumba:/$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
 # DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
 nameserver 2001:db8::2

mrz@bumba:/$ ping6 www.mikrotik.com
PING www.mikrotik.com(2a02:610:7501:1000::2) 56 data bytes
 64 bytes from 2a02:610:7501:1000::2: icmp_seq=1 ttl=61 time=2.11 ms
 64 bytes from 2a02:610:7501:1000::2: icmp_seq=2 ttl=61 time=1.33 ms
^C
 --- www.mikrotik.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.334/1.725/2.117/0.393 ms
mrz@bumba:/$
```

IP Pools

Summary

IP pools are used to define range of IP addresses that can be used by various RouterOS utilities, for example, DHCP server, Point-to-Point servers and more. Separate lists for IPv4 and IPv6 are available. Whenever possible, the same IP address is given out to each client (OWNER/INFO pair).

IPv4 Pool

Sub-menu: `/ip pool`

Property	Description
comment (*string*; Default:)	Short description of the pool
name (*string*; Default:)	Unique identifier of the pool
next-pool (*string*; Default:)	When IP address acquisition is performed a pool that has no free addresses, and the next-pool property is set, then IP address will be acquired from next-pool
ranges (*IP*; Default:)	IP address list of non-overlapping IP address ranges in the form of: from1-to1,from2-to2,...,fromN-toN. For example, 10.0.0.1-10.0.0.27,10.0.0.32-10.0.0.47

Example

To define a pool named "my-pool" with the 10.0.0.1-10.0.0.126 address range excluding gateway's address 10.0.0.1 and server's address 10.0.0.100, and the other pool dhcp-pool, with the 10.0.0.200-10.0.0.250 address range:

```
[admin@MikroTik] ip pool> add name=my-pool ranges=10.0.0.2-10.0.0.99,10.0.0.101-10.0.0.126
[admin@MikroTik] ip pool> add name=dhcp-pool ranges=10.0.0.200-10.0.0.250
[admin@MikroTik] ip pool> print
  # NAME                                      RANGES
  0 ip-pool                                   10.0.0.2-10.0.0.99
                                              10.0.0.101-10.0.0.126
  1 dhcp-pool                                 10.0.0.200-10.0.0.250
```

Used addresses

Sub-menu: `/ip pool used`

Here you can see all used IP addresses from IP pools.

Read-only properties

Property	Description

address (*IP*)	IP address that is assigned to client from the pool
info (*string*)	For DHCP MAC address from leases menu and for PPP connections username of PPP type client
owner (*string*)	Service which is using this IP address
pool (*string*)	Name of the IP pool

IPv6 Pool

Sub-menu: `/ipv6 pool`

Property	Description
name (*string*; Default:)	Descriptive name of the pool.
prefix (*IPv6/0..128*; Default:)	Ipv6 address prefix
prefix-length (*integer [1..128]*; Default:)	The option represents the prefix size that will be given out to the client.

Read-only properties

Property	Description
dynamic (*yes* | *no*)	Whether the pool is dynamic.
expire-time (*time*)	Expire time is set to dynamic pools added by DHCPv6 client.

Example

The example will create a pool of "2001::/60" to give out /62 prefixes:

```
[admin@test-host] /ipv6 pool> add
name: test prefix: 2001::/60
prefix-length: 62
[admin@test-host] /ipv6 pool> print
# NAME PREFIX PREFIX-LENGTH
0 test 2001::/60 62bits
```

Used addresses

Sub-menu: `/ipv6 pool used`

Read-only properties

Property	Description
info (*string*)	Shows DUID related information received from the client (value in hex). Can contain also a raw timestamp in hex.
owner (*string*)	What reserved the prefix ("DHCP", etc.)
pool (*string*)	Name of the pool.
prefix (*IPv6/0..128*)	IPv6 prefix that is assigned to the client from the pool.

IP Routing

Overview

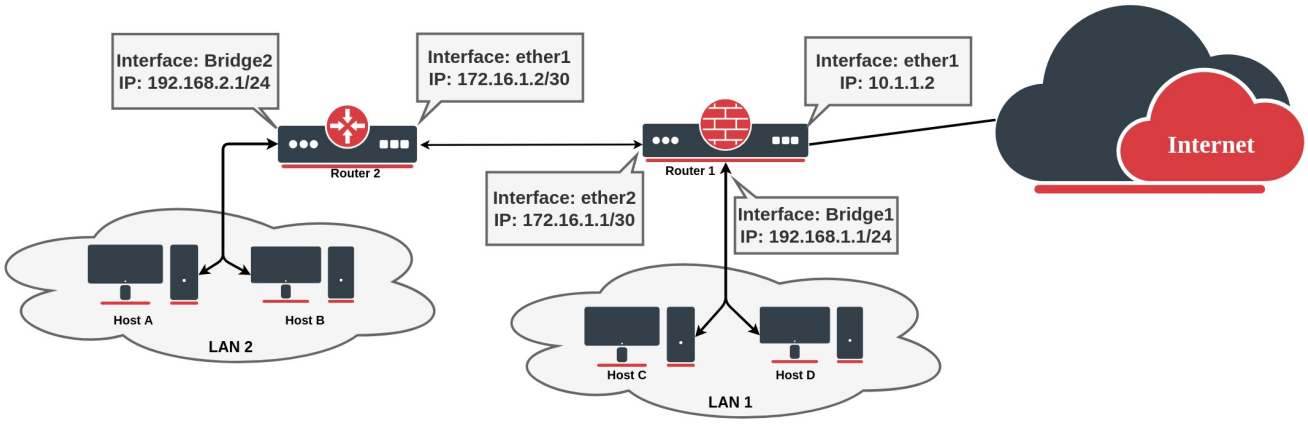
Routing is the process of selecting paths across the networks to move packets from one host to another.

How Routing Works

Let's look at a basic configuration example to illustrate how routing is used to forward packets between two local networks and to the Internet.

In this setup, we have several networks:

- two client networks (192.168.2.0/24 and 192.168.1.0/24);
- one network to connect routers (172.16.1.0/30), usually called backbone;
- the last network (10.1.1.0/24) connects our gateway router (Router1) to the internet.



Router 2:

```
/ip address
add address=172.16.1.2/30 interface=ether1
add address=192.168.2.1/24 interface=bridge2
```

Router1 (gateway) where ether1 connects to the internet:

```
/ip address
add address=10.1.1.2/24 interface=ether1
add address=172.16.1.1/30 interface=ether2
add address=192.168.1.1/24 interface=bridge1
```

If we look, for example, at the Router1 routing table, we can see that the router knows only about *directly connected* networks. At this point, when the Client from LAN1 tries to reach the client from LAN2 (192.168.2.0/24), a packet will be dropped on the router, because the destination is unknown for the particular router:

```
[admin@MikroTik] > /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active; C - connect, S - static, r - ri
p, b - bgp, o - ospf, d - dhcp, v - vpn
Columns: DST-ADDRESS, GATEWAY, Distance
    DST-ADDRESS    GATEWAY D
DAC 10.1.1.0/24    ether1  0
DAC 172.16.1.0/30  ether2  0
DAC 192.168.1.0/24 bridge1 0
```

To fix this we need to add a route that tells the router what is the next device in the network to reach the destination. In our example next hop is Router2, so we need to add a route with the gateway that points to the Router's 2 connected address. This type of route is known as a ***static route***:

```
[admin@MikroTik] > /ip route add dst-address=192.168.2.0/24 gateway=172.16.1.2
[admin@MikroTik] > /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active; C - connect, S - static, r - ri
p, b - bgp, o - ospf, d - dhcp, v - vpn
Columns: DST-ADDRESS, GATEWAY,       Distance
        DST-ADDRESS    GATEWAY       D
    DAC 10.1.1.0/24    ether1        0
    DAC 172.16.1.0/30  ether2        0
    DAC 192.168.1.0/24 bridge1       0
0   AS  192.168.2.0/24 172.16.1.2
```

At this point packet from LAN1 will be successfully forwarded to LAN2, but we are not over yet. Router2 does not know how to reach LAN1, so any packet from LAN2 will be dropped on Router2.

If we look again at the network diagram, we can clearly see that Router2 has only one point of exit. It is safe to assume that all other unknown networks should be reached over the link to Router1. The easiest way to do this is by adding a ***default route***: To add a default route set destination 0.0.0.0/0 or leave it blank:

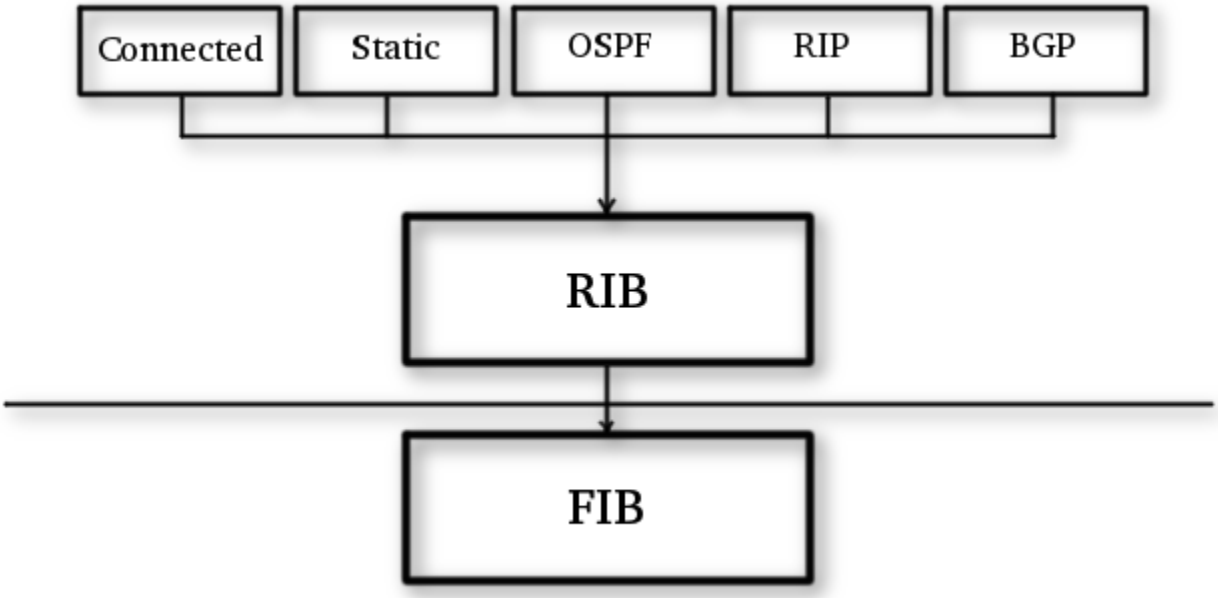
```
/ip route add gateway=172.16.1.1
```

As we have seen from the example setup, there are different groups of routes, based on their origin and properties.

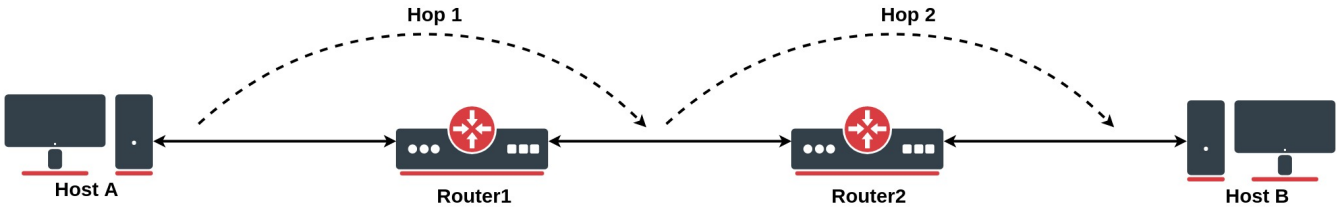
Routing Information

RouterOS routing information consists of two main parts:

- **FIB** (Forwarding Information Base), is used to make packet forwarding decisions. It contains a copy of the necessary routing information.
- **RIB** (Routing Information Base) contains all learned prefixes from routing protocols (connected, static, BGP, RIP, OSPF).



Routing Information Base



Routing Information Base is a database that lists entries for particular network destinations and their *gateways* (address of the next device along the path or simply *next-hop*). One such entry in the routing table is called a *route*.

A *hop* occurs when a packet is passed from one network segment to another.

By default, all routes are organized in one "main" routing table. It is possible to make more than one routing table which we will discuss further in this article, but for now, for sake of simplicity, we will consider that there is only one "main" routing table.

RIB table contains complete routing information, including static routes and policy routing rules configured by the user, routing information learned from dynamic routing protocols (RIP, OSPF, BGP), and information about connected networks.

Its purpose is not just to store routes, but also to filter routing information to calculate the best route for each destination prefix, to build and update the Forwarding Information Base, and to distribute routes between different routing protocols.

Connected Routes

Connected routes represent the network on which hosts can be directly reached (direct attachment to Layer2 broadcast domain). These routes are created automatically for each IP network that has at least one enabled interface attached to it (as specified in the */ip address* or */ipv6 address* configuration). RIB tracks the status of connected routes but does not modify them. For each connected route there is one IP address item such that:

- **address** part of the *dst-address* of the connected route is equal to a network of IP address item.
- **netmask** part of *dst-address* of the connected route is equal to the netmask part of the address of the IP address item.
- **gateway** of the connected route is equal to the actual-*interface* of the IP address item (same as an interface, except for bridge interface ports) and represents an interface where directly connected hosts from the particular Layer3 network can be reached.

The **preferred source** is not used anymore for connected routes. FIB chooses the source address based on the out-interface. This allows making setups that in ROS v6 and older were considered invalid. See examples for more details.

Default Route

A default route is used when the destination cannot be resolved by any other route in the routing table. In RouterOS *dst-address* of the default route is ***0.0.0.0/0*** *(for IPv4) and* ***::/0*** *(for IPv6)* routes. If the routing table contains an active default route, then the routing table lookup in this table will never fail.

Typically home router routing table contains only connected networks and one default route to forward all outgoing traffic to the ISP's gateway:

```
[admin@TempTest] /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active; C - connect, S - static, r - ri
p, b - bgp, o - ospf, d - dhcp, v - vpn
Columns: DST-ADDRESS, GATEWAY, Distance
#      DST-ADDRESS     GATEWAY      D
   DAd 0.0.0.0/0       10.155.125.1 1
   DAC 10.155.125.0/24 ether12      0
   DAC 192.168.1.0/24  vlan2        0
```

Hardware Offloaded Route

Devices with Layer 3 Hardware Offloading (L3HW, otherwise known as IP switching or HW routing) allow offloading packet routing onto the switch chip. When L3HW is enabled, such routes will display H-flag:

```
[admin@MikroTik] > /ip/route print where static
Flags: A - ACTIVE; s - STATIC, y - COPY; H - HW-OFFLOADED
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#     DST-ADDRESS       GATEWAY         D
0 AsH 0.0.0.0/0         172.16.2.1      1
1 AsH 10.0.0.0/8        10.155.121.254  1
2 AsH 192.168.3.0/24    172.16.2.1      1
```

By default, all the routes are participating to be hardware candidate routes. To further fine-tune which traffic to offload, there is an option for each IP or IPv6 static route to disable/enable `suppress-hw-offload`.

For example, if we know that the majority of traffic flows to the network where servers are located, we can enable offloading only to that specific destination:

```
/ip route set [find where static && dst-address!="192.168.3.0/24"] suppress-hw-offload=yes
```

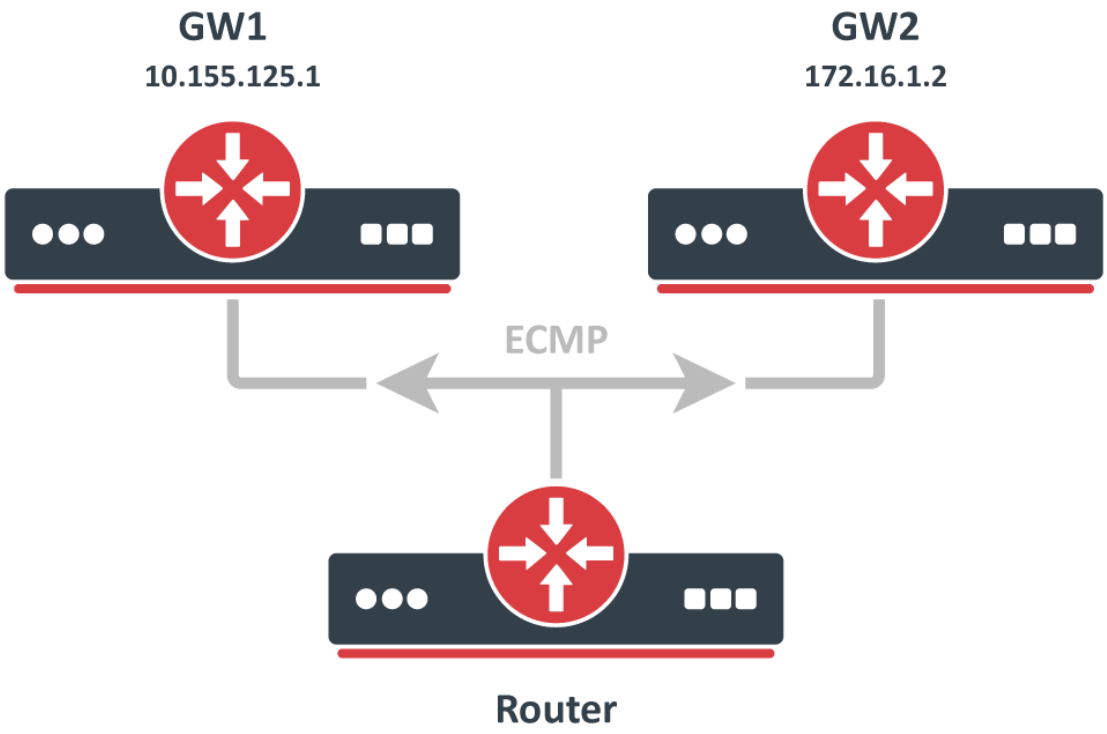
Now only the route to 192.168.3.0/24 has an H-flag, indicating that it will be the only one eligible to be selected for HW offloading:

```
[admin@MikroTik] > /ip/route print where static
Flags: A - ACTIVE; s - STATIC, y - COPY; H - HW-OFFLOADED
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#     DST-ADDRESS       GATEWAY         D
0 As  0.0.0.0/0         172.16.2.1      1
1 As  10.0.0.0/8        10.155.121.254  1
2 AsH 192.168.3.0/24    172.16.2.1      1
```

H-flag does not indicate that the route is actually HW offloaded, it indicates only that route can be selected to be HW offloaded.

Multipath (ECMP) routes

To implement some setups, such as load balancing, it might be necessary to use more than one path to a given destination.



ECMP (Equal cost multi-path) routes have multiple gateways (next-hop) values. All reachable next-hops are copied to FIB and are used to forward packets.

These routes can be created manually, as well as dynamically by any of the dynamic routing protocols (OSPF, BGP, RIP). Multiple equally preferred routes to the same destination will have assigned + flag and grouped together automatically by RouterOS (see example below).

```
[admin@TempTest] /ip/route> print
Flags: D - DYNAMIC; I - INACTIVE, A - ACTIVE; C - CONNECT, S - STATIC, m - MODEM; + - ECMP
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#       DST-ADDRESS      GATEWAY       D
0   AS+ 192.168.2.0/24   10.155.125.1  1
1   AS+ 192.168.2.0/24   172.16.1.2    1
```

By default, ECMP uses Layer3 hash policy

Route Selection

There can be multiple routes with the same destination received from various routing protocols and from static configurations but only one (best) destination can be used for packet forwarding. To determine the best path, RIB runs a Route Selection algorithm that picks the best route from all candidate routes per destination.

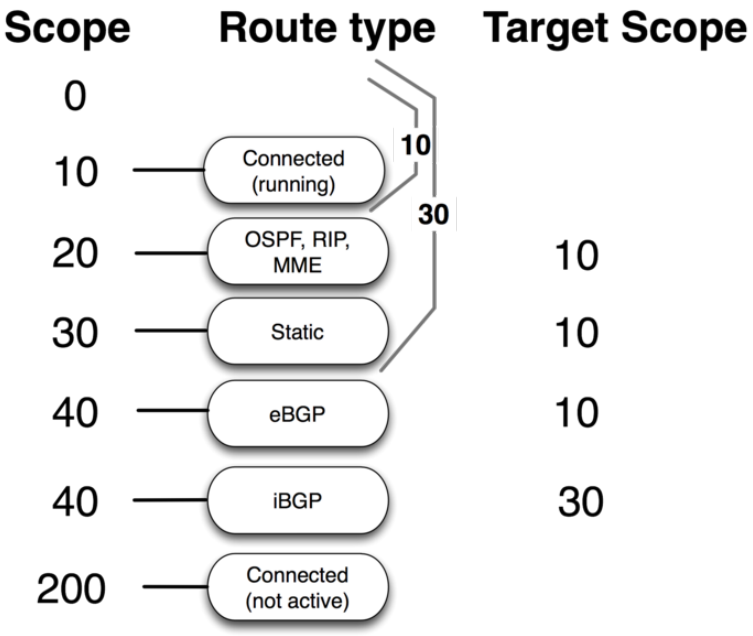
Only routes that meet the following criteria can participate in the route selection process:

- Route is not disabled.

- If the type of route is *unicast* it must have at least one reachable next-hop. (if a gateway is from a connected network and there is a connected route active, the gateway is considered as reachable)
- Route should not be *synthetic*.

The candidate route with the lowest distance becomes an active route. If there is more than one candidate route with the same distance, the selection of the active route is arbitrary.

Nexthop Lookup



Nexthop lookup is a part of the route selection process. Its main purpose is to find a directly reachable gateway address (next-hop). Only after a valid next-hop is selected router knows which interface to use for packet forwarding.

Nexthop lookup becomes more complicated if routes have a gateway address that is several hops away from this router (e. g. iBGP, multihop eBGP). Such routes are installed in the FIB after the next-hop selection algorithm determines the address of the directly reachable gateway (immediate next-hop).

It is necessary to restrict the set of routes that can be used to look up immediate next-hops. Nexthop values of RIP or OSPF routes, for example, are supposed to be directly reachable and should be looked up only using connected routes. This is achieved using scope and target-scope properties.

Routes with a scope greater than the maximum accepted value are not used for next-hop lookup. Each route specifies the maximum accepted scope value for its nexthop in the target-scope property. The default value of this property allows nexthop lookup only through connected routes, with the exception of iBGP routes that have a larger default value and can lookup nexthop also through IGP and static routes.

There are changes in RouterOS v7 nexthop lookup.

Routes are processed in scope order, and updates to routes with a larger scope cannot affect the state of nexthop lookup for routes with a smaller scope.

Consider an example from v6:

```
/ip route add dst-address=10.0.1.0/24 gateway=10.0.0.1
    scope=50 target-scope=30 comment=A
/ip route add dst-address=10.0.2.0/24 gateway=10.0.0.1
    scope=30 target-scope=20 comment=B
/ip route add dst-address=10.0.0.0/24 scope=20 gateway=WHATEVER
    comment=C
```

Gateway 10.0.0.1 is recursively resolved through C using the smallest referring scope (scope 20 from route B), both routes are active. Now we change both A and B at the same time:

```
/ip route set A target-scope=10
```

Suddenly, applying an update to route A makes the gateway of route B inactive. This is because in v6 there is only one gateway object per address.

v7 keeps multiple gateway objects per address, one for each combination of scope and gateway check.

When `target-scope` or gateway check of a route is changed, ROS v7 ***will not affect other routes***, as it does in v6. In v7 target-scope and gateway check are properties that are internally attached to the gateway, not to the route.

Gateway check can be extended by setting `check-gateway` parameter. Gateway reachability can be checked by sending ARP probes, or ICMP messages or by checking active BFD sessions. The router periodically (every 10 seconds) checks the gateway by sending either an ICMP echo request (*ping*) or an ARP request (*arp*). If no response from the gateway is received for 10 seconds, the request times out. After two timeouts gateway is considered unreachable. After receiving a reply from the gateway it is considered reachable and the timeout counter is reset.

Route Storage

Routing information is stored to take as little memory as possible in a common case. These optimizations have non-obvious worst cases and impact on performance.

All routes and gateways are kept in a single hierarchy by the prefix/address.

```
Dst [4]/0 1/0+4                           18  <-- number of prefixes
     ^  ^ ^ ^ ^
     |  | | | |
     |  | | | \- bytes taken by Route distinguisher or Interface Id
     |  | | \--- vrf/routing table
     |  | \----- AFI
     |  \------- netmask length of prefix
     \---------- bytes taken by prefix value

     [subject to change without notice]
```

Each of these 'Dst' corresponds to a unique 'dst-address' of route or address of the gateway. Each 'Dst' requires one or more 'T2Node' objects as well.

All routes with the same 'dst-address' are kept in Dst in a list sorted by route preference.
Note: WORST CASE: having a lot of routes with the same 'dst-address' is really slow! even if they are inactive! because updating a sorted list with tens of thousands of elements is slow!

Route order changes only when route attributes change. If the route becomes active/inactive, the order does not change.

Each Route has three copies of route attributes:

- **private** -- what is received from the peer, before passing in-filters.
- **updated** -- what is the result of applying in-filters.
- **current** -- what are the attributes currently used by the route.

Periodically (when needed), ***update*** attributes are calculated from ***private*** attributes. This happens when route update is received, or when in-filter is updated.

When the routing table is recalculated, ***current*** attributes are set to the value from ***updated*** attributes.

This means, that usually if there is no in-filter that changes route attributes, ***private***, ***updated,*** and ***current*** share the same value.

Route attributes are kept in several groups:

- L1 Data - all flags, list of extra properties, as-path;
- L2 Data - nexthops, RIP, OSPF, BGP metrics, route tags, originators, etc.
- L3 Data - distance, scope, kernel type, MPLS stuff
- extra properties - communities, originator, aggregator-id, cluster-list, unknown

Having for example many different combinations of ***distance*** and ***scope*** route attributes will use more memory!

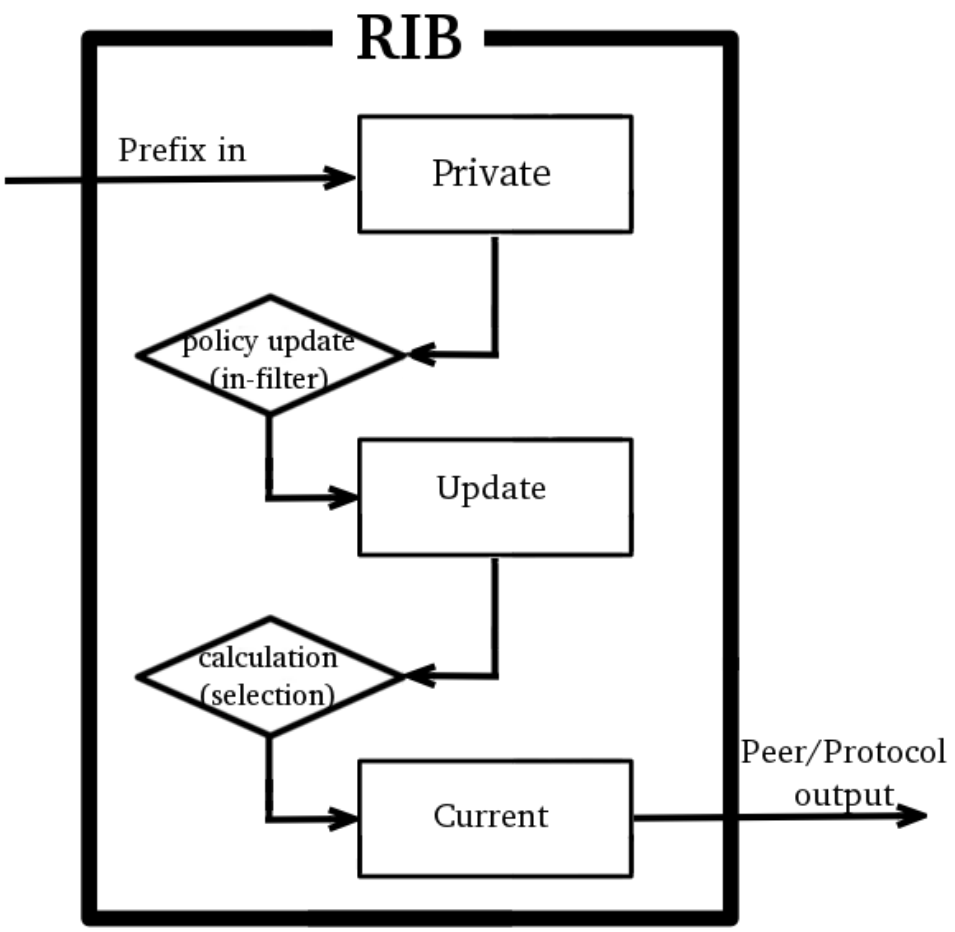
Matching communities or as-path using regexp will cache the result, to speed up filtering. Each as-path or community value has a cache for all regexp, which is filled on-demand with match results.
Note: WORST CASE: changing attributes in 'in-filter' will make the route program use more memory! Because 'private' and 'updated' attributes will be different! Having a lot of different regexps will make matching slow and use a lot of memory! Because each value will have a cache with thousands of entries!

Detailed info about used memory by routing protocols can be seen in `/routing stats memory` menu

Forwarding Information Base

FIB (Forwarding Information Base) contains a copy of the information that is necessary for packet forwarding:

- all active routes
- policy routing rules



Each route has **dst-address** property, that specifies all destination addresses this route can be used for. If several routes apply to a particular IP address, the most specific one (with the largest netmask) is used. This operation (finding the most specific route that matches the given address) is called "routing table lookup".

Only one Best route can be used for packet forwarding. In cases where the routing table contains several routes with the same **dst-address**, all equally best routes are combined into one ECMP route. The best route is installed into FIB and marked as "active".

When forwarding decision uses additional information, such as the source address of the packet, it is called **policy routing**. Policy routing is implemented as a list of policy routing rules, that select different routing tables based on the destination address, source address, source interface, and routing mark (which can be changed by firewall mangle rules) of the packet.

Routing table lookup

FIB uses the following information from the packet to determine its destination:

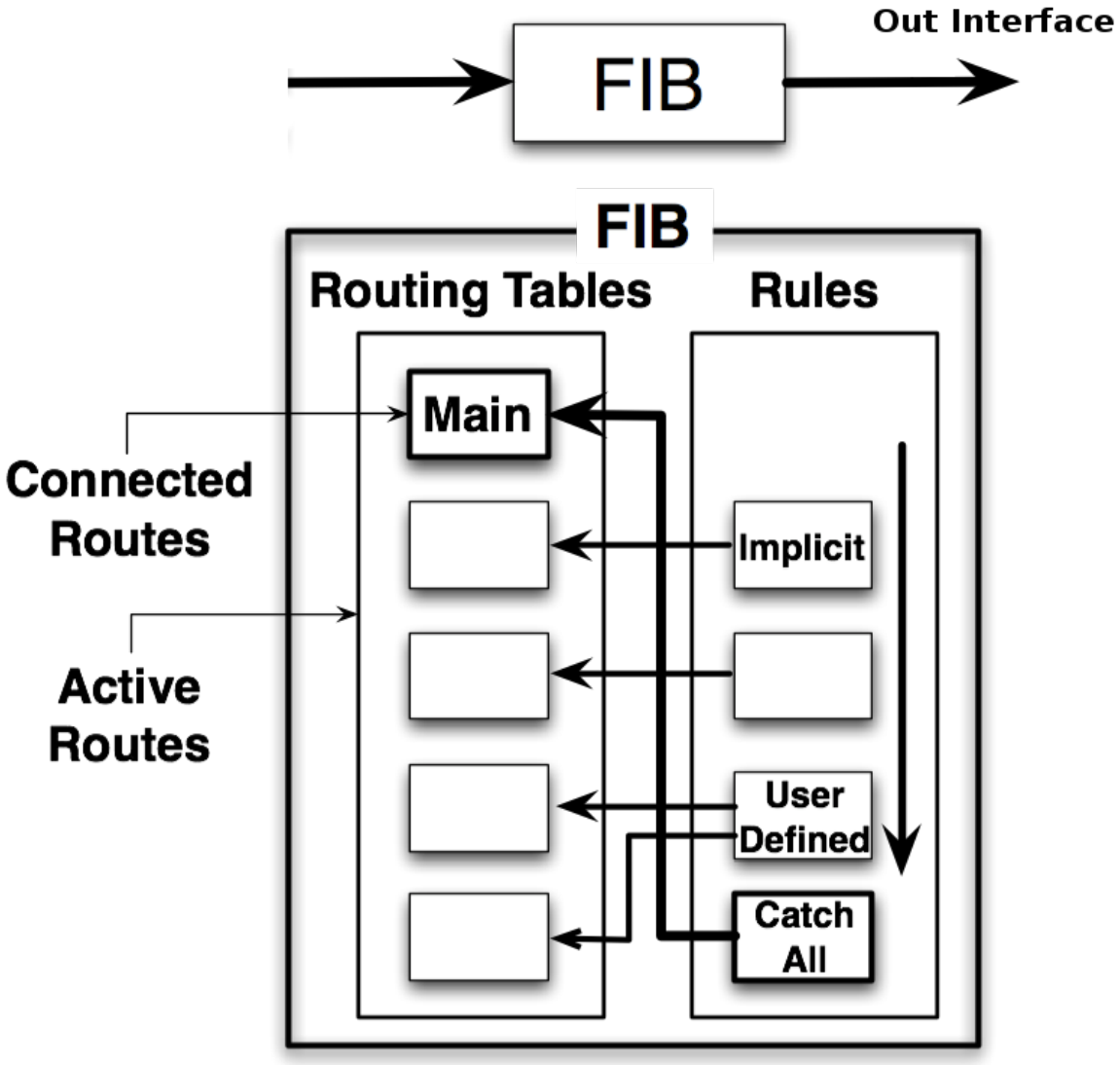
- source address
- destination address
- source interface
- routing mark

Possible routing decisions are:

- receive packet locally
- discard the packet (either silently or by sending an ICMP message to the sender of the packet)
- send the packet to a specific IP address on a specific interface

Run routing decision:

- check that the packet has to be locally delivered (the destination address is the address of the router)



- process implicit policy routing rules
- process policy routing rules added by a user
- process implicit catch-all rule that looks up the destination in the "main" routing table
- the returned result is "network unreachable"

The result of the routing decision can be:

- IP address of nexthop + interface
- point-to-point interface
- local delivery
- discard
- ICMP prohibited
- ICMP host unreachable
- ICMP network unreachable

Rules that do not match the current packet are ignored. If a rule has action:

- **drop** or **unreachable**, then it is returned as a result of the routing decision process.

- **lookup** then the destination address of the packet is looked up in the routing table that is specified in the rule. If the lookup fails (no route matches the destination address of the packet), then FIB proceeds to the next rule.
- **lookup-only** is similar to **lookup** except that lookup fails if none of the routes in the table matches the packet.

Otherwise:

- if the type of the route is *blackhole*, *prohibit*, or *unreachable*, then return this action as the routing decision result;
- if this is a connected route or route with an interface as the **gateway** value, then return this interface and the destination address of the packet as the routing decision result;
- if this route has an IP address as the value of the **gateway**, then return this address and associated interface as the routing decision result;
- if this route has multiple values of nexthop, then pick one of them in a round-robin fashion.

Show Routes

In RouterOS you have three menus to see the current state of routes in the routing table:

- `/ip route` - list IPv4 routes and basic properties
- `/ipv6 route` - list IPv6 routes and basic properties
- `/routing route` - list all routes with extended properties

> `/routing route` menu currently is read-only. To add or remove routes `/ip(ipv6) route` menus should be used.

Example output

```
[admin@MikroTik] /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active; C - connect, S - stati
c, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn
Columns: DST-ADDRESS, GATEWAY, DIstance
#       DST-ADDRESS      GATEWAY      DI
0   XS   10.155.101.0/24  1.1.1.10
1   XS                    11.11.11.10
   D d   0.0.0.0/0        10.155.101.1 10
2   AS   0.0.0.0/0        10.155.101.1 1
3   AS + 1.1.1.0/24       10.155.101.1 10
4   AS + 1.1.1.0/24       10.155.101.2 10
5   AS   8.8.8.8          2.2.2.2      1
   DAC   10.155.101.0/24  ether12      0


|  ||| |   |                |         |
|  ||| |   |                |         \----Distance
|  ||| |   |                \--Configured gateway
|  ||| |   \-- dst prefix
|  ||| \----- ECMP flag
|  ||\------- protocol flag (bgp, osf,static,connected etc.)
|  |\-------- route status flag (active, inactive, disabled)
|  \--------- shows if route is dynamic
\----------- console order number (shown only for static editable routes)
```

`routing route` output is very similar to ip route except that it shows routes from all address families in one menu and lists filtered routes as well.

```
[admin@MikroTik] /routing/route> print
Flags: X - disabled, I - inactive, F - filtered, U - unreachable, A - active; c - connect, s - static,
r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, a - ldp-address, l - ldp-mapping
Columns: DST-ADDRESS, GATEWAY, DIStance, SCOpe, TARget-scope, IMMEDIATE-GW
     DST-ADDRESS           GATEWAY      DIS SCO TAR IMMEDIATE-GW
Xs   10.155.101.0/24
Xs
d    0.0.0.0/0             10.155.101.1 10  30  10  10.155.101.1%ether12
As   0.0.0.0/0             10.155.101.1 1   30  10  10.155.101.1%ether12
```

```
As   1.1.1.0/24             10.155.101.1 10  30  10  10.155.101.1%ether12
As   8.8.8.8                2.2.2.2      1   254 254 10.155.101.1%ether12
Ac   10.155.101.0/24        ether12      0   10      ether12
Ic   2001:db8:2::/64        ether2       0   10
Io   2001:db8:3::/64        ether12      110 20  10
Ic   fe80::%ether2/64       ether2       0   10
Ac   fe80::%ether12/64      ether12      0   10      ether12
Ac   fe80::%bridge-main/64  bridge-main  0   10      bridge-main
A    ether12                             0   250
A    bridge-main                         0   250
```

`routing route print detail` shows more advanced info useful for debugging

```
[admin@MikroTik] /routing route> print detail
Flags: X - disabled, I - inactive, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, a - ldp-address, l - ldp-ma>
+ - ecmp
Xs dst-address=10.155.101.0/24
Xs
d afi=ip4 contribution=best-candidate dst-address=0.0.0.0/0 gateway=10.155.101.1
immediate-gw=10.155.101.1%ether12 distance=10 scope=30 target-scope=10
belongs-to="DHCP route" mpls.in-label=0 .out-label=0 debug.fwp-ptr=0x201C2000

As afi=ip4 contribution=active dst-address=0.0.0.0/0 gateway=10.155.101.1
immediate-gw=10.155.101.1%ether12 distance=1 scope=30 target-scope=10
belongs-to="Static route" mpls.in-label=0 .out-label=0 debug.fwp-ptr=0x201C2000
```

IP Settings

Summary

Several IPv4 and IPv6 related kernel and system-wide parameters are configurable.

IPv4 Settings

Sub-menu: `/ip settings`

Property	Description
accept-redirects (*yes* | *no*; Default: **no**)	Whether to accept ICMP redirect messages. Typically should be enabled on a host and disabled on routers.
accept-source-route (*yes* | *no*; Default: **no**)	Whether to accept packets with the SRR option. Typically should be enabled on the router.
allow-fast-path (*yes* | *no*; Default: **yes**)	Allows Fast Path.
arp-timeout (*time interval*; Default: **30s**)	Sets Linux **base_reachable_time** (base_reachable_time_ms) on all interfaces that use ARP. The initial validity of the ARP entry is picked from the interval [timeout/2 - 3*timeout/2] (default from 15s to 45s) after the neighbor was found. Can use postfix ms, s, m, h, d for milliseconds, seconds, minutes, hours, or days. if no postfix is set then seconds (s) are used. The parameter means how long a valid ARP record will be considered complete if no one communicates with the specific MAC/IP during this time. The parameter does not represent a time when an ARP entry is removed from the ARP cache (see **max-neighbor-entries** setting).
icmp-rate-limit (*integer [0..4294967295]*; Default: **10**)	Limit the maximum rates for sending ICMP packets whose type matches icmp-rate-mask to specific targets. **0** disables any limiting, other values indicate the minimum space between responses in milliseconds.
icmp-rate-mask (*[0..FFFFFFFF]*; Default: **0x1818**)	Mask made of ICMP types for which rates are being limited. More info in Linux man pages
ip-forward (*yes* | *no*; Default: **yes**)	Enable/disable packet forwarding between interfaces. Resets all configuration parameters to defaults according to RFC1812 for routers.

rp-filter (*loose* | *no* | *strict*; Default: **no**)	Disables or enables source validation. • no - No source validation. • strict - Strict mode as defined in RFC3704 Strict Reverse Path. Each incoming packet is tested against the FIB and if the interface is not the best reverse path the packet check will fail. By default failed packets are discarded. • loose - Loose mode as defined in RFC3704 Loose Reverse Path. Each incoming packet's source address is also tested against the FIB and if the source address is not reachable via any interface the packet check will fail. The current recommended practice in RFC3704 is to enable strict mode to prevent IP spoofing from DDoS attacks. If using asymmetric routing or other complicated routing or VRRP, then the loose mode is recommended. **Warning:** strict mode does not work with routing tables
secure-redirects (*yes* | *no*; Default: **yes**)	Accept ICMP redirect messages only for gateways, listed in the default gateway list.
send-redirects (*yes* | *no*; Default: **yes**)	Whether to send ICMP redirects. Recommended to be enabled on routers.
tcp-syncookies (*yes* | *no*; Default: **no**)	Send out syncookies when the syn backlog queue of a socket overflows. This is to prevent the common 'SYN flood attack'. syncookies seriously violate TCP protocol, and disallow the use of TCP extensions, which can result in serious degradation of some services (f.e. SMTP relaying), visible not by you, but to your clients and relays, contacting you.
max-neighbor-entries (*integer [0..4294967295]*; Default:)	Sets Linux **gc_thresh3.** A maximum number of allowed neighbors in the ARP table. Since RouterOS version 7.1, the default value depends on the installed amount of RAM. It is possible to set a higher value than the default, but it increases the risk of out-of-memory condition. The default values for certain RAM sizes: • 2048 for 64 MB, • 4096 for 128 MB, • 8192 for 256 MB, • 16384 for 512 MB or higher. The ARP cache stores ARP entries, and if some of these entries are incomplete, they can stay in the cache for an indefinite period of time. This will only happen if the number of entries in the cache is less than one-fourth of the maximum number allowed. The reason for this is to prevent the unnecessary running of the garbage-collector when the ARP table is not close to being full.
route-cache (*yes* | *no*; Default: **yes**)	Disable or enable the Linux route cache. Note that disabling the route cache, will also disable the fast path.

Read-Only Properties

Property	Description
ipv4-fast-path-active (*yes* | *no*)	Indicates whether fast-path is active
ipv4-fast-path-bytes (*integer*)	Amount of fast-pathed bytes
ipv4-fast-path-packets (*integer*)	Amount of fast-pathed packets
ipv4-fasttrack-active (*yes* | *no*)	Indicates whether fasttrack is active
ipv4-fasttrack-bytes (*integer*)	Amount of fasttracked bytes
ipv4-fasttrack-packets (*integer*)	Amount of fasttracked packet.

IPv6 Settings

Sub-menu: `/ipv6 settings`

> Changing /ipv6 settings will not dynamically remove the old SLAAC configuration present on your router. A reboot is required to apply the new settings.

Property	Description
accept-redirects (*no* | *yes-if-forwarding-disabled*; Default: **yes-if-forwarding-disabled**)	Whether to accept ICMP redirect messages. Typically should be enabled on the host and disabled on routers
accept-router-advertisements (*no* | *yes* | *yes-if-forwarding-disabled*; Default: **yes-if-forwarding-disabled**)	Accept router advertisement (RA) messages. If enabled, the router will be able to get the address using stateless address configuration
disable-ipv6 (*yes* | *no*; Default: **no**)	Enable/disable system wide IPv6 settings (prevents LL address generation)
forward (*yes* | *no*; Default: **yes**)	Enable/disable packet forwarding between interfaces
max-neighbor-entries (*integer [0.. 4294967295]*; Default:)	A maximum number or IPv6 neighbors. Since RouterOS version 7.1, the default value depends on the installed amount of RAM. It is possible to set a higher value than the default, but it increases the risk of out-of-memory condition. The default values for certain RAM sizes: • 1024 for 64 MB, • 2048 for 128 MB, • 4096 for 256 MB, • 8192 for 512 MB, • 16384 for 1024 MB or higher.

Management tools

In This Section:

After successful RouterOS software installation (if it was needed) it is time to access the router for the first time. There are various ways how to connect to it:

- Command Line Interface (CLI) via Telnet, SSH, serial cable or keyboard and monitor if the router has VGA card.
- Web interface (WebFig)
- WinBox configuration utility
- Using mobile utility (MikroTik mobile app)

Normally you connect to the router by IP addresses with any telnet or SSH client software (a simple text-mode telnet client is usually called telnet and is distributed together with almost any OS).

If no IP or MAC connectivity is available, some devices allow connection through a serial port (DB9 or RJ45, depending on the model);

- Accessing devices over serial console

API

Summary

Application Programmable Interface (API) allows users to create custom software solutions to communicate with RouterOS to gather information, adjust the configuration, and manage the router. API closely follows syntax from the command-line interface (CLI). It can be used to create translated or custom configuration tools to aid ease of use in running and managing routers with RouterOS.

API service must be enabled before trying to establish the API connection. By default, API uses TCP:*8728* and TCP:*8729* (secure).

API-SSL service is capable of working in two modes - with and without a certificate. In the case no certificate is used in */ip service* settings then an anonymous Diffie-Hellman cipher has to be used to establish a connection. If a certificate is in use, a TLS session can be established.

Protocol

Communication with the router is done by sending sentences and receiving one or more sentences in return. A sentence is a sequence of words terminated by zero-length words. Word is part of a sentence encoded in a certain way - encoded length and data. Communication happens by sending sentences to the router and receiving replies to sent sentences. Each sentence sent to the router using API should contain a command as a first word followed by words in no particular order, the end of the sentence is marked by a zero-length word. When the router receives a full sentence (command word, no or more attribute words, and zero-length word) it is evaluated and executed, then a reply is formed and returned.

API words

Words are part of a sentence. Each word has to be encoded in a certain way - the length of the word followed by the word content. The length of the word should be given as a count of bytes that are going to be sent.

The length of the word is encoded as follows:

Value of length	# of bytes	Encoding
0 <= len <= 0x7F	1	len, lowest byte
0x80 <= len <= 0x3FFF	2	len | 0x8000, two lower bytes
0x4000 <= len <= 0x1FFFFF	3	len | 0xC00000, three lower bytes

0x200000 <= len <= 0xFFFFFFF	4	len | 0xE0000000
len >= 0x10000000	5	0xF0 and len as four bytes

- Each word is encoded as length, followed by that many bytes of content;
- Words are grouped into sentences. The end of a sentence is terminated by a zero-length word;
- The scheme allows encoding of length up to **0x7FFFFFFFFF**, only four-byte length is supported;
- **len** bytes are sent most significant first (network order);
- If the first byte of the word is **>= 0xF8**, then it is a reserved control byte. After receiving an unknown control byte API client cannot proceed, because it does not know how to interpret the following bytes;
- Currently, control bytes are not used;

In general, *words* can be described like this **<<***encoded word length***><***word content***>>**. *Word content* can be separated into 5 parts: *command word*, *attribute word*, *API attribute word*. *query word,* and *reply word*

Command word

The first word in the sentence has to be a command followed by attribute words and a zero-length word or terminating word. The name of the command word should begin with '/'. Names of commands closely follow CLI, with spaces replaced with '/'. Some commands are specific to API;

Command word structure in the strict order:

- encoded length
- content prefix /
- CLI converted command

API-specific commands:

```
login
cancel
```

Command word content examples:

```
/login
```

```
/user/active/listen
```

```
/interface/vlan/remove
```

```
/system/reboot
```

Attribute word

Each *command word* has its list of *attribute words* depending on content.

Attribute word structure consists of 5 parts in this order:

- encoded length
- content prefix equals sign - **=**
- attribute name
- separating equals sign - **=**
- value of an attribute if there is one. It is possible that the attribute does not have a value

⚠ Value can hold multiple *equal* signs in the value of an *attribute word* since the way the word is encoded.

⚠ Value can be empty.

Examples without encoded length prefix:

```
=address=10.0.0.1
```

```
=name=iu=c3Eeg
```

```
=disable-running-check=yes
```

> Order of attribute words and API parameters is not important and should not be relied on

API attribute word

API attribute word structure is in the strict order:

- encoded length
- content prefix with the dot.
- attribute name
- name postfixed with equals =sign
- attribute value

Currently, the only such API attribute is the *tag*.

> If the sentence contains an *API attribute word* tag then each returned sentence in reply from the router to that tagged sentence will be tagged with the same tag.

Query word

Sentences can have additional query parameters that restrict their scope. A detailed explanation is in the query section.

Example of a sentence using query word attributes:

```
/interface/print
?type=ether
?type=vlan
?#|!
```

- Query words begin with '?'.
- Currently, only the *print* command handles query words.

> The order of query words is significant

Reply word

It is only sent by the router in response to the full sentence received from the client.

- The first word of reply begins with '!';
- Each sentence sent generates at least one reply (if a connection does not get terminated);
- The last reply for every sentence is the reply that has the first word !done;
- Errors and exceptional conditions begin with !trap;
- Data replies begin with !re
- If the API connection is closed, RouterOS sends !fatal with a reason as a reply and then closes the connection;

API sentences

API sentence is the main object of communication using API.

- Empty sentences are ignored.
- A sentence is processed after receiving zero length word.
- There is a limit on the number and size of sentences that the client can send before it has logged in.
- Order of attribute words should not be relied on. As order and count are changeable by *.proplist* attribute.
- The sentence structure is as follows:
  - The first word should contain a *command word*;
  - Should contain *zero-length word* to terminate the sentence;
  - Can contain none or several *attribute words*. There is no particular order in what attribute words have to be sent in the sentence, order is not important for *attribute words*;

- Can contain none or several *query words*. The order of *query words* in the sentence is important.

> ⚠ *Zero-length word* terminates the sentence. If it is not provided router will not start to evaluate sent words and will consider all the input as part of the same sentence.

Initial login

Note: that each command and response ends with an empty word.

Login method post-v6.43:

/login
=name=admin
=password=
!done

- Now the client sends a username and password in the first message.
- Password is sent in plain text.
- in case of an error, the reply contains =message=*error message*.
- In case of a successful login, the client can start to issue commands.

Tags

- It is possible to run several commands simultaneously, without waiting for the previous one to complete. If the API client is doing this and needs to differentiate command responses, it can use the 'tag' API parameter in command sentences.
- If you include the 'tag' parameter with a non-empty value in the command sentence, then the 'tag' parameter with the same value will be included in all responses generated by this command.
- If you do not include the 'tag' parameter or its value is empty, then all responses for this command will not have a 'tag' parameter.

Command description

- /cancel
  - optional argument: =tag=*tag of command to cancel*, without it, cancels all running commands
  - does not cancel itself
  - all canceled commands are interrupted and in the usual case generate '!trap' and '!done' responses
  - please note that /cancel is separate command and can have its own unique '.tag' parameter, that is not related to '=tag' argument of this command

- listen
  - listen command is available where console print command is available, but it does not have the expected effect everywhere (i.e. may not work)
  - "!re" sentences are generated as something changes in a particular item list
  - when an item is deleted or disappears in any other way, the '!re' sentence includes the value '=.dead=yes'
  - This command does not terminate. To terminate it, use /cancel command.

- getall
  - getall command is available where console print command is available (getall is an alias for print).
  - replies contain =.id=*Item internal number* property.

- print
  - API print command differs from the console counterpart in the following ways:
    - where an argument is not supported. Items can be filtered using query words (see below).
    - .proplist argument is a comma-separated list of property names that should be included for the returned items.
      - returned items may have additional properties.
      - order of returned properties is not defined.
      - if a list contains duplicate entries, handling of such entries is not defined.

- if a property is present in ".proplist", but absent from the item, then that item does not have this property value (?name will evaluate to false for that item).
- if ".proplist" is absent, all properties are included as requested by the print command, even those that have slow access time (such as file contents and performance counters). Thus the use of .proplist is encouraged. The omission of .proplist may have a high-performance penalty if the "=detail=" argument is set.

Queries

The print command accepts query words that limit the set of returned sentences.

- Query words begin with '?'.
- The order of query words is significant. A query is evaluated starting from the first word.
- A query is evaluated for each item in the list. If the query succeeds, the item is processed, if a query fails, the item is ignored.
- A query is evaluated using a stack of boolean values. Initially, the stack contains an infinite amount of 'true' values. At the end of the evaluation, if the stack contains at least one 'false' value, the query fails.
- Query words operate according to the following rules:

Query	Description
?name	pushes 'true' if an item has a value of property *name*, 'false' if it does not.
?-name	pushes 'true' if an item does not have a value of property *name*, 'false' otherwise.
?*name*=*x* **?=*name*=*x***	pushes 'true' if the property *name* has a value equal to *x*, 'false' otherwise.
?<name=*x*	pushes 'true' if the property *name* has a value less than *x*, 'false' otherwise.
?>name=*x*	pushes 'true' if the property *name* has a value greater than *x*, 'false' otherwise.
?#*operations*	applies operations to the values in the stack. • operation string is evaluated from left to right. • the sequence of decimal digits followed by any other character or end of the word is interpreted as a stack index. top value has an index 0. • an index that is followed by a character pushes a copy of the value at that index. • an index that is followed by the end of the word replaces all values with the value at that index. • **!** character replaces the top value with the opposite. • **&** pops two values and pushes the result of logical 'and' operation. • **|** pops two values and pushes the result of logical 'or' operation. • **.** after an index does nothing. • **.** after another character pushes a copy of the top value.

Regular expressions are not supported in API, so do not try to send a query with the **~** symbol

Examples:

- Get all ethernet and VLAN interfaces:

```
/interface/print
?type=ether
?type=vlan
?#|
```

- Get all routes that have a non-empty comment:

```
/ip/route/print
?>comment=
```

- Forum thread with a detailed explanation of the use of queries

OID

The print command can return OID values for properties that are available in SNMP.

In the console, OID values can be seen by running the 'print oid' command. In API, these properties have a name that ends with ".oid", and can be retrieved by adding their name to the value of '.proplist'. An example:

/system/resource/print
=.proplist=uptime,cpu-load,uptime.oid,cpu-load.oid
!re
=uptime=01:22:53
=cpu-load=0
=uptime.oid=.1.3.6.1.2.1.1.3.0
=cpu-load.oid=.1.3.6.1.2.1.25.3.3.1.2.1
!done

!trap

When for some reason API sentence fails trap is sent in return accompanied by a **message** attribute and on some occasions **category** argument.

message

When an API sentence fails, some generic message or message from the used internal process is returned to give more details about the failure

```
<<< /ip/address/add
<<< =address=192.168.88.1
<<< =interface=asdf <<<

>>> !trap
>>> =category=1
>>> =message=input does not match any value of interface
```

category

if it is a general error, it is categorized and the error category is returned. possible values for this attribute are

- 0 - missing item or command
- 1 - argument value failure
- 2 - execution of command interrupted
- 3 - scripting related failure
- 4 - a general failure
- 5 - API related failure
- 6 - TTY related failure
- 7 - value generated with :return command

Command examples

/system/package/getall

/system/package/getall
!re
=.id=*5802
=disabled=no
=name=routeros-x86
=version=3.0beta2
=build-time=oct/18/2006 16:24:41
=scheduled=
!re
=.id=*5805
=disabled=no
=name=system
=version=3.0beta2
=build-time=oct/18/2006 17:20:46
=scheduled=
... more !re sentences ...
!re
=.id=*5902
=disabled=no
=name=advanced-tools
=version=3.0beta2
=build-time=oct/18/2006 17:20:49
=scheduled=
!done

/user/active/listen

/user/active/listen
!re
=.id=*68
=radius=no

=when=oct/24/2006 08:40:42
=name=admin
=address=0.0.0.0
=via=console
!re
=.id=*68
=.dead=yes
... more !re sentences ...

/cancel, simultaneous commands

/login
!done
=ret=856780b7411eefd3abadee2058c149a3
/login
=name=admin
=response=005062f7a5ef124d34675bf3e81f56c556
!done
-- first start listening for interface changes (tag is 2)
/interface/listen
.tag=2
-- disable interface (tag is 3)
/interface/set
=disabled=yes
=.id=ether1
.tag=3
-- this is done for disable command (tag 3)
!done
.tag=3
-- enable interface (tag is 4)
/interface/set
=disabled=no

=.id=ether1
.tag=4
-- this update is generated by a change made by the first set command (tag 3)
!re
=.id=*1
=disabled=yes
=dynamic=no
=running=no
=name=ether1
=mtu=1500
=type=ether
.tag=2
-- this is done for enable command (tag 4)
!done
.tag=4
-- get interface list (tag is 5)
/interface/getall
.tag=5
-- this update is generated by a change made by the second set command (tag 4)
!re
=.id=*1
=disabled=no
=dynamic=no
=running=yes
=name=ether1
=mtu=1500
=type=ether
.tag=2
-- these are replies to getall command (tag 5)
!re
=.id=*1
=disabled=no

=dynamic=no
=running=yes
=name=ether1
=mtu=1500
=type=ether
.tag=5
!re
=.id=*2
=disabled=no
=dynamic=no
=running=yes
=name=ether2
=mtu=1500
=type=ether
.tag=5
-- here interface getall ends (tag 5)
!done
.tag=5
-- stop listening - request to cancel command with tag 2, cancel itself uses tag 7
/cancel
=tag=2
.tag=7
-- listen command is interrupted (tag 2)
!trap
=category=2
=message=interrupted
.tag=2
-- cancel command is finished (tag 7)
!done
.tag=7

-- listen command is finished (tag 2)
!done
.tag=2

Example client

A simple API client in Python3

Example output:

```
debian@localhost:~/api-test$ ./api.py 10.0.0.1 admin ''
<<< /login
<<<
>>> !done
>>> =ret=93b438ec9b80057c06dd9fe67d56aa9a
>>>
<<< /login
<<< =name=admin
<<< =response=00e134102a9d330dd7b1849fedfea3cb57
<<<
>>> !done
>>>
/user/getall

<<< /user/getall
<<<
>>> !re
>>> =.id=*1
>>> =disabled=no
>>> =name=admin
>>> =group=full
>>> =address=0.0.0.0/0
>>> =netmask=0.0.0.0
>>>
>>> !done
>>>
```

See also

API examples

API implementations in different languages, provided by different sources. They are not ordered in any particular order.

- in Python3 by MikroTik
- in .NET (C#) high-level API solution forum thread additional info by danikf
- in PHP by boen_robot
- in C by Håkon Nessjøen
- in Java by Gideon LeGrange
- in Erlang by Valery Comtihon
- in GO by André Luiz dos Santos
- in Python3 by Arturs Laizans
- in C++17 by Ayman Al-Qadhi

Python3 Example

A simple Python3 example client.

- usage: api.py *ip-address username password secure*
  i.e. api.py 10.0.0.1 Admin Badpassword123 True
- after that, type words from the keyboard, terminating them with a new line
- Since an empty word terminates a sentence, you should press enter **twice** after the last word before a sentence will be sent to the router.

```
#!/usr/bin/python3
# -*- coding: latin-1 -*-
import sys, posix, time, binascii, socket, select, ssl
import hashlib

class ApiRos:
        "Routeros api"
        def __init__(self, sk):
                self.sk = sk
                self.currenttag = 0

        def login(self, username, pwd):
                for repl, attrs in self.talk(["/login", "=name=" + username, "=password=" + pwd]):
                        if repl == '!trap':
                                return False
                        elif '=ret' in attrs.keys():
                                #for repl, attrs in self.talk(["/login"]):
                                chal = binascii.unhexlify((attrs['=ret']).encode(sys.stdout.encoding))
                                md = hashlib.md5()
                                md.update(b'\x00')
                                md.update(pwd.encode(sys.stdout.encoding))
                                md.update(chal)
                                for repl2, attrs2 in self.talk(["/login", "=name=" + username, "=response=00"
                                        + binascii.hexlify(md.digest()).decode(sys.stdout.encoding) ]):
                                        if repl2 == '!trap':
                                                return False
                return True

        def talk(self, words):
                if self.writeSentence(words) == 0: return
                r = []
                while 1:
                        i = self.readSentence();
                        if len(i) == 0: continue
                        reply = i[0]
                        attrs = {}
                        for w in i[1:]:
                                j = w.find('=', 1)
                                if (j == -1):
                                        attrs[w] = ''
                                else:
                                        attrs[w[:j]] = w[j+1:]
                        r.append((reply, attrs))
                        if reply == '!done': return r

        def writeSentence(self, words):
                ret = 0
                for w in words:
                        self.writeWord(w)
                        ret += 1
                self.writeWord('')
                return ret

        def readSentence(self):
                r = []
                while 1:
                        w = self.readWord()
                        if w == '': return r
```

```
                r.append(w)

        def writeWord(self, w):
                print(("<<< " + w))
                self.writeLen(len(w))
                self.writeStr(w)

        def readWord(self):
                ret = self.readStr(self.readLen())
                print((">>> " + ret))
                return ret

        def writeLen(self, l):
                if l < 0x80:
                        self.writeByte((l).to_bytes(1, sys.byteorder))
                elif l < 0x4000:
                        l |= 0x8000
                        tmp = (l >> 8) & 0xFF
                        self.writeByte(((l >> 8) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte((l & 0xFF).to_bytes(1, sys.byteorder))
                elif l < 0x200000:
                        l |= 0xC00000
                        self.writeByte(((l >> 16) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 8) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte((l & 0xFF).to_bytes(1, sys.byteorder))
                elif l < 0x10000000:
                        l |= 0xE0000000
                        self.writeByte(((l >> 24) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 16) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 8) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte((l & 0xFF).to_bytes(1, sys.byteorder))
                else:
                        self.writeByte((0xF0).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 24) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 16) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte(((l >> 8) & 0xFF).to_bytes(1, sys.byteorder))
                        self.writeByte((l & 0xFF).to_bytes(1, sys.byteorder))

        def readLen(self):
                c = ord(self.readStr(1))
                # print (">rl> %i" % c)
                if (c & 0x80) == 0x00:
                        pass
                elif (c & 0xC0) == 0x80:
                        c &= ~0xC0
                        c <<= 8
                        c += ord(self.readStr(1))
                elif (c & 0xE0) == 0xC0:
                        c &= ~0xE0
                        c <<= 8
                        c += ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                elif (c & 0xF0) == 0xE0:
                        c &= ~0xF0
                        c <<= 8
                        c += ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                elif (c & 0xF8) == 0xF0:
                        c = ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                        c <<= 8
                        c += ord(self.readStr(1))
                return c
```

```
        def writeStr(self, str):
                n = 0;
                while n < len(str):
                        r = self.sk.send(bytes(str[n:], 'UTF-8'))
                        if r == 0: raise RuntimeError("connection closed by remote end")
                        n += r

        def writeByte(self, str):
                n = 0;
                while n < len(str):
                        r = self.sk.send(str[n:])
                        if r == 0: raise RuntimeError("connection closed by remote end")
                        n += r

        def readStr(self, length):
                ret = ''
                # print ("length: %i" % length)
                while len(ret) < length:
                        s = self.sk.recv(length - len(ret))
                        if s == b'': raise RuntimeError("connection closed by remote end")
                        # print (b">>>" + s)
                        # atgriezt kaa byte ja nav ascii chars
                        if s >= (128).to_bytes(1, "big") :
                                return s
                        # print((">>> " + s.decode(sys.stdout.encoding, 'ignore')))
                        ret += s.decode(sys.stdout.encoding, "replace")
                return ret

def open_socket(dst, port, secure=False):
        s = None
        res = socket.getaddrinfo(dst, port, socket.AF_UNSPEC, socket.SOCK_STREAM)
        af, socktype, proto, canonname, sockaddr = res[0]
        skt = socket.socket(af, socktype, proto)
        if secure:
                s = ssl.wrap_socket(skt, ssl_version=ssl.PROTOCOL_TLSv1_2, ciphers="ECDHE-RSA-AES256-GCM-
SHA384") #ADH-AES128-SHA256
        else:
                s = skt
        s.connect(sockaddr)
        return s

def main():
        s = None
        dst = sys.argv[1]
        user = "admin"
        passw = ""
        secure = False
        port = 0

        # use default username and pasword if not specified
        arg_nr = len(sys.argv)
    if arg_nr > 2: user = sys.argv[2]
    if arg_nr > 3: passw = sys.argv[3]
    if arg_nr > 4: secure = sys.argv[4]

        if (port==0):
                port = 8729 if secure else 8728

        s = open_socket(dst, port, secure)
        if s is None:
                print ('could not open socket')
                sys.exit(1)

        apiros = ApiRos(s);
        if not apiros.login(user, passw):
                return

        inputsentence = []

        while 1:
                r = select.select([s, sys.stdin], [], [], None)
```

```
                if s in r[0]:
                        # something to read in socket, read sentence
                        x = apiros.readSentence()

                if sys.stdin in r[0]:
                        # read line from input and strip off newline
                        l = sys.stdin.readline()
                        l = l[:-1]

                        # if empty line, send sentence and start with new
                        # otherwise append to input sentence
                        if l == '':
                                apiros.writeSentence(inputsentence)
                                inputsentence = []
                        else:
                                inputsentence.append(l)

if __name__ == '__main__':
        main()
```

Branding

RouterOS allows slight system customization with the help of a branding package (modify default configuration, LCD logo, WebFig homepage, etc.).

This is a special system package, which you can generate from within your mikrotik.com account, in the account section "Branding maker". The resulting file will have a .dpk extension and can be installed by all the same means as an .npk package.

To install the package on a router, branding package has to upload to it and then a router has to reboot, Netinstall tool can be used for the same effect.

The generated package can be installed in any RouterOS version.

> Note that specific branding features are available starting from specific RouterOS versions.

Options

Options that can be configured using a branding package:

- **Router name** - branding package name, device identity and platform name in RouterOS, can only be one word, don't use spaces or special characters;
- **Company URL** - value that appears in the console when you connect to RouterOS device;
- **Manual URL** - documentation link, which can be opened in WebFig;
- **ASCII Logo** - a text logo that is shown when logging into the command line interface, i.e. Telnet, SSH, WinBox Terminal. A logo can be created in the branding maker or copied from any other plaintext editor. A logo height cannot be larger than 8 lines, width is not limited, but note that in a narr ow terminal window a logo might be distorted.
- **Hide "Mikrotik" from SNMP information** - MikroTik name will be hidden in SNMP information;
- **Do not run script on install** - do not run Default configuration script on branding package install;
- **Disable "/system reset-configuration no-defaults=yes"** - allows only to reset to default configuration, and it will not be possible to reset a router without configuration or in CAPs mode;
- **Hide Default configuration prompt** - hide Default configuration prompt after configuration reset *(available starting from RouterOS 7.15)*
- **Hide default caps-mode-script** - hide default caps-mode-script *(available starting from RouterOS 7.15)*

Custom files

A custom files like custom default configuration, skins, WebFig login page, etc., can be added in branding package.

- **WebFig login page** - customized a default RouterOS information page, which shows up when accessing the router IP address. When making the HTML file, you can use these variables:
  - %version% will change to the router's current version;
  - %host% will change to the router's IP address.

  The file must be named "index2.html". Make sure you use properly nested HTML to make your page compatible with all browsers. You can also upload images or JavaScript files, they must reference to the same path as the index file, no custom folder names can be used;

- **WebFig logo** - Router WEB page logo, will overwrite the original. It should be named "mikrotik_logo.png";
- **\hotspot** - Hotspot login page logo, the file must be named "logobottom.png";
- **\skins** - a skin file with name your_file_name.json. To apply a particular skin to a specific user group, you don't need to log into the router to do that. You can do it with branding by uploading a Default configuration file;
- **Default configuration** - a RouterOS default configuration file that will override RouterOS default configuration. This configuration will be kept even after RouterOS reset. It is possible to reapply the factory passwords by utilizing the read-only variables *$defconfPassword* and *$defconfWifiPassw ord* (access to factory passwords is available starting RouterOS 7.10);
- **LCD logo** - LCD logo will be displayed on devices equipped with LCD screen. A Logo size cannot be larger than 160px width and 72px height. CC R1xxx series has white (0xffffff) background, 2011 series have black (0x000000) background;
- **Custom files** - custom files will be simply copied into a folder named "branding" and will be accessible from within RouterOS.
- **CAPs mode script** - a RouterOS CAPs mode script that will override RouterOS default CAPs mode script. It is possible to reapply the factory passwords by utilizing the read-only variables *$defconfPassword* and *$defconfWifiPassword* (available starting from RouterOS 7.15).

Command Line Interface

The console is used for accessing the MikroTik Router's configuration and management features using text terminals, either remotely using a serial port, telnet, SSH, console screen within WinBox, or directly using monitor and keyboard. The console is also used for writing scripts. This manual describes the general console operation principles. Please consult the Scripting Manual on some advanced console commands and on how to write scripts.

Login Options

Console login options enable or disable various console features like color, terminal detection, and many other.

Additional login parameters can be appended to the login name after the '+' sign.

```
login_name ::= user_name [ '+' parameters ]
parameters ::= parameter [ parameters ]
parameter ::= [ number ] 'a'..'z'
number ::= '0'..'9' [ number ]
```

If the parameter is not present, then the default value is used. If the number is not present then the implicit value of the parameter is used.

Example: admin+c80w - will disable console colors and set terminal width to 80.

Param	Default	Implicit	Description
"w"	auto	auto	Set terminal width
"h"	auto	auto	Set terminal height
"c"	on	off	disable/enable console colors
"t"	on	off	Do auto-detection of terminal capabilities
"e"	on	off	Enables "dumb" terminal mode

Banner and Messages

The login process will display the MikroTik banner and short help after validating the user name and password.

```
  MMM      MMM       KKK                          TTTTTTTTTTT      KKK
  MMMM    MMMM       KKK                          TTTTTTTTTTT      KKK
  MMM MMMM MMM  III  KKK  KKK  RRRRRR     OOOOOO      TTT     III  KKK  KKK
  MMM  MM  MMM  III  KKKKK     RRR  RRR  OOO  OOO     TTT     III  KKKKK
  MMM      MMM  III  KKK KKK   RRRRRR    OOO  OOO     TTT     III  KKK KKK
  MMM      MMM  III  KKK  KKK  RRR  RRR   OOOOOO      TTT     III  KKK  KKK

  MikroTik RouterOS 6.22 (c) 1999-2014       https://www.mikrotik.com/

[?]             Gives the list of available commands
command [?]     Gives help on the command and list of arguments

[Tab]           Completes the command/word. If the input is ambiguous,
                a second [Tab] gives possible options

/               Move up to base level
..              Move up one level
/command        Use command at the base level
```

After the banner can be printed other important information, like **/system note** set by another admin, the last few critical log messages, demo version upgrade reminder, and default configuration description.

For example, the demo license prompt and last critical messages are printed

```
UPGRADE NOW FOR FULL SUPPORT
----------------------------
FULL SUPPORT benefits:
- receive technical support
- one year feature support
- one year online upgrades
    (avoid re-installation and re-configuring your router)
To upgrade, register your license "software ID"
on our account server www.mikrotik.com

Current installation "software ID": ABCD-456

Please press "Enter" to continue!


dec/10/2007 10:40:06 system,error,critical login failure for user root from 10.0.0.1 via telnet
dec/10/2007 10:40:07 system,error,critical login failure for user root from 10.0.0.1 via telnet
dec/10/2007 10:40:09 system,error,critical login failure for user test from 10.0.0.1 via telnet
```

Command Prompt

At the end of the successful login sequence, the login process prints a banner that shows the command prompt, and hands over control to the user.

Default command prompt consists of user name, system identity, and current command path **/>**

For example, change the current path from the root to the interface then go back to the root

```
  [admin@MikroTik] > interface [enter]
  [admin@MikroTik] /interface> / [enter]
  [admin@MikroTik] >
```

Use **up arrow** to recall previous commands (if this is a multiline command, then you can press **F8** in order to expand it) from command history (commands that added sensitive data, like passwords, will not be available in the history), **TAB** key to automatically complete words in the command you are typing, **ENTER** key to execute the command, **Control-C** to interrupt currently running command and return to prompt and **?** to display built-in help, in **RouterOS v7**, **F1** has to be used instead.

The easiest way to log out of the console is to press **Control-D** at the command prompt while the command line is empty (You can cancel the current command and get an empty line with **Control-C**, so **Control-C** followed by **Control-D** will log you out in most cases).

It is possible to write commands that consist of multiple lines. When the entered line is not a complete command and more input is expected, the console shows a continuation prompt that lists all open parentheses, braces, brackets, and quotes, and also trailing backslash if the previous line ended with **backsl ash**-white-space.

```
    [admin@MikroTik] > {
    {... :put (\
    {(\... 1+2)}
    3
```

When you are editing such multiple line entries, the prompt shows the number of current lines and total line count instead of the usual username and system name.

```
line 2 of 3> :put (\
```

Sometimes commands ask for additional input from the user. For example, the command '`/password`' asks for old and new passwords. In such cases, the prompt shows the name of the requested value, followed by colon and space.

```
    [admin@MikroTik] > /password
    old password: ******
    new password: **********
    retype new password: **********
```

Hierarchy

The console allows the configuration of the router's settings using text commands. Since there is a lot of available commands, they are split into groups organized in a way of hierarchical menu levels. The name of a menu level reflects the configuration information accessible in the relevant section.

For example, you can issue the `/ip route print` command:

```
[admin@MikroTik] > /ip route print
Flags: D - dynamic; X - disabled, I - inactive, A - active;
C - connect, S - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn
 #     DST-ADDRESS        GATEWAY            DISTANCE
 0  XS 4.4.4.4            10.155.101.1
   D o 0.0.0.0/0          10.155.101.1            110
 1  AS 0.0.0.0/0          10.155.101.1              1
   D b 1.0.4.0/24         10.155.101.1             20
   D b 1.0.4.0/24         10.155.101.1             20
   DAb 1.0.4.0/24         10.155.101.1             20
[admin@MikroTik] >
```

Instead of typing `/ip route` path before each command, the path can be typed only once to move into this particular branch of the menu hierarchy. Thus, the example above could also be executed like this:

```
[admin@MikroTik] > /ip route
[admin@MikroTik] /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active;
C - connect, S - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn
 #     DST-ADDRESS        GATEWAY            DISTANCE
 0  XS 4.4.4.4            10.155.101.1
   D o 0.0.0.0/0          10.155.101.1            110
 1  AS 0.0.0.0/0          10.155.101.1              1
   D b 1.0.4.0/24         10.155.101.1             20
   D b 1.0.4.0/24         10.155.101.1             20
   DAb 1.0.4.0/24         10.155.101.1             20
[admin@MikroTik] >
```

Each word in the path can be separated by **space** (as in the example above) or by "/"

```
[admin@MikroTik] > /ip/route/
[admin@MikroTik] /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active;
C - connect, S - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn
 #     DST-ADDRESS        GATEWAY            DISTANCE
 0  XS 4.4.4.4            10.155.101.1
   D o 0.0.0.0/0          10.155.101.1            110
 1  AS 0.0.0.0/0          10.155.101.1              1
   D b 1.0.4.0/24         10.155.101.1             20
   D b 1.0.4.0/24         10.155.101.1             20
   DAb 1.0.4.0/24         10.155.101.1             20
[admin@MikroTik] >
```

Notice that the prompt changes in order to reflect where you are located in the menu hierarchy at the moment. To move to the top level again, type " / "

```
[admin@MikroTik] > ip route
[admin@MikroTik] /ip/route> /
[admin@MikroTik] >
```

To move up one command level, type " .. "

```
[admin@MikroTik] /ip/route> ..
[admin@MikroTik] /ip>
```

You can also use **/** and **..** to execute commands from other menu levels without changing the current level:

```
[admin@MikroTik] /ip/route> /ping 10.0.0.1
10.0.0.1 ping timeout
2 packets transmitted, 0 packets received, 100% packet loss
[admin@MikroTik] /ip/firewall/nat> .. service-port print
Flags: X - disabled, I - invalid
#   NAME                                                                PORTS
0   ftp                                                                 21
1   tftp                                                                69
2   irc                                                                 6667
3   h323
4   sip
5   pptp
[admin@MikroTik] /ip/firewall/nat>
```

Item Names and Numbers

Many of the command levels operate with arrays of items: interfaces, routes, users, etc. Such arrays are displayed in similarly-looking lists. All items in the list have an item number followed by flags and parameter values.

To change the properties of an item, you have to use the set command and specify the name or number of the item.

Item Names

Some lists have items with specific names assigned to each of them. Examples are interface or user levels. There you can use item names instead of item numbers.

You do not have to use the print command before accessing items by their names, which, as opposed to numbers, are not assigned by the console internally, but are properties of the items. Thus, they would not change on their own. However, there are all kinds of obscure situations possible when several users are changing the router's configuration at the same time. Generally, item names are more "stable" than the numbers, and also more informative, so you should prefer them to numbers when writing console scripts.

Item Numbers

Item numbers are assigned by the print command and are not constant - it is possible that two successive print commands will order items differently. But the results of the last print commands are memorized and, thus, once assigned, item numbers can be used even after add, remove and move operations (since version 3, move operation does not renumber items). Item numbers are assigned on a per session basis, they will remain the same until you quit the console or until the next print command is executed. Also, numbers are assigned separately for every item list, so `/ip address print` will not change the numbering of the interface list.

You can specify multiple items as targets to some commands. Almost everywhere, where you can write the number of items, you can also write a list of numbers.

```
[admin@MikroTik] > interface print
Flags: X - disabled, D - dynamic, R - running
  #    NAME                TYPE             MTU
  0  R ether1               ether            1500
  1  R ether2               ether            1500
  2  R ether3               ether            1500
  3  R ether4               ether            1500
[admin@MikroTik] > interface set 0,1,2 mtu=1460
[admin@MikroTik] > interface print
Flags: X - disabled, D - dynamic, R - running
  #    NAME                TYPE             MTU
  0  R ether1               ether            1460
  1  R ether2               ether            1460
  2  R ether3               ether            1460
  3  R ether4               ether            1500
[admin@MikroTik] >
```

General Commands

There are some commands that are common to nearly all menu levels, namely: **print, set, remove, add, find, get, export, enable, disable, comment, move**. These commands have similar behavior throughout different menu levels.

Property	Description
add	This command usually has all the same arguments as a set, except the item number argument. It adds a new item with the values you have specified, usually at the end of the item list, in places where the order of items is relevant. There are some required properties that you have to supply, such as the interface for a new address, while other properties are set to defaults unless you explicitly specify them. Common Parameters • copy-from - Copies an existing item. It takes default values of a new item's properties from another item. If you do not want to make an exact copy, you can specify new values for some properties. When copying items that have names, you will usually have to give a new name to a copy • place-before - places a new item before an existing item with a specified position. Thus, you do not need to use the move command after adding an item to the list • disabled - controls disabled/enabled state of the newly added item(-s) • comment - holds the description of a newly created item Return Values • add command returns the internal number of items it has added
edit	This command is associated with the set command. It can be used to edit values of properties that contain a large amount of text, such as scripts, but it works with all editable properties. Depending on the capabilities of the terminal, either a full-screen editor or a single line editor is launched to edit the value of the specified property.
find	The find command has the same arguments as a set, plus the flag arguments like disabled or active that take values yes or no depending on the value of the respective flag. To see all flags and their names, look at the top of the print command's output. The find command returns internal numbers of all items that have the same values of arguments as specified.

move	Changes the order of items in the list. Parameters: • the first argument specifies the item(-s) being moved. • the second argument specifies the item before which to place all items being moved (they are placed at the end of the list if the second argument is omitted).
print	Shows all information that's accessible from a particular command level. Thus, `/system clock print` shows the system date and time, `/ip route print` shows all routes etc. If there\'s a list of items in the current level and they are not read-only, i.e. you can change /remove them (example of read-only item list is `/system history`, which shows a history of executed actions), then print command also assigns numbers that are used by all commands that operate with items in this list. Common Parameters: • append - • brief - forces the print command to use tabular output form • count-only - shows the number of items • detail - forces the print command to use property=value output form • file - prints the contents of the specific sub-menu into a file on the router. • follow - • follow-only - • follow-strict - • from - show only specified items, in the same order in which they are given. • interval - updates the output from the print command for every interval of seconds. • oid - prints the OID value for properties that are accessible from SNMP. • proplist - comma-separated and ordered list of property names that should be included for the returned items. • show-ids - • where - show only items that match specified criteria. The syntax of where the property is similar to the find command. • without-paging - prints the output without stopping after each screenful.
remove	Removes specified item(-s) from a list.
set	Allows you to change values of general parameters or item parameters. The set command has arguments with names corresponding to values you can change. Use ? or double Tab to see a list of all arguments. If there is a list of items in this command level, then the set has one action argument that accepts the number of items (or list of numbers) you wish to set up. This command does not return anything.

Input Modes

It is possible to switch between several input modes:

- **Normal mode** - indicated by normal command prompt.
- **Safe mode** - safe mode is indicated by the word SAFE after the command prompt. In this mode, the configuration is saved to disk only after the safe mode is turned off. Safe mode can be turned on/off with **Ctrl+X or F4.** Read more >>
- **Hot-lock mode** - indicated by additional yellow **>**. Hot-lock mode autocompletes commands and can be turned on/off with **F7**

Quick Typing

There are two features in the console that help entering commands much quicker and easier - the [**Tab**] key completions, and abbreviations of command names. Completions work similarly to the bash shell in UNIX. If you press the [**Tab**] key after a part of a word, the console tries to find the command within the current context that begins with this word. If there is only one match, it is automatically appended, followed by a space:

/inte[**Tab**]_ becomes **/interface _**

If there is more than one match, but they all have a common beginning, which is longer than that what you have typed, then the word is completed to this common part, and no space is appended:

/interface set e[**Tab**]_ becomes **/interface set ether_**

If you've typed just the common part, pressing the tab key once has no effect. However, pressing it for the second time shows all possible completions in compact form:

```
[admin@MikroTik] > interface set e[Tab]_
[admin@MikroTik] > interface set ether[Tab]_
[admin@MikroTik] > interface set ether[Tab]_
ether1 ether5
[admin@MikroTik] > interface set ether_
```

The **[Tab]** key can be used almost in any context where the console might have a clue about possible values - command names, argument names, arguments that have only several possible values (like names of items in some lists or name of the protocol in firewall and NAT rules). You cannot complete numbers, IP addresses, and similar values.

Another way to press fewer keys while typing is to abbreviate command and argument names. You can type only the beginning of the command name, and, if it is not ambiguous, the console will accept it as a full name. So typing:

```
[admin@MikroTik] > pi 10.1 c 3 si 100
```

equals to:

```
[admin@MikroTik] > ping 10.0.0.1 count 3 size 100
```

It is possible to complete not only the beginning, but also any distinctive sub-string of a name: if there is no exact match, the console starts looking for words that have string being completed as first letters of a multiple word name, or that simply contain letters of this string in the same order. If a single such word is found, it is completed at the cursor position. For example:

```
[admin@MikroTik] > interface x[TAB]_
[admin@MikroTik] > interface export _

[admin@MikroTik] > interface mt[TAB]_
[admin@MikroTik] > interface monitor-traffic _
```

Console Search

Console search allows performing keyword search through the list of RouterOS menus and the history. The search prompt is accessible with the **[Ctrl+r]** sh ortcut.

Internal Chat System

RouterOS console has a built-in internal chat system. This allows remotely located admins to talk to each other directly in RouterOS CLI. To start the conversation prefix the intended message with the **#** symbol, anyone who is logged in at the time of sending the message will see it.

```
[admin@MikroTik] > # ready to break internet?
[admin@MikroTik] >
fake_admin: i was born ready
[admin@MikroTik] >
```

```
[fake_admin@MikroTik] >
admin: ready to break internet?
[fake_admin@MikroTik] > # i was born ready
[fake_admin@MikroTik] >
```

List of Keys

Key	Description
Control-C	keyboard interrupt
Control-D	log out (if an input line is empty)
Control-K	clear from the cursor to the end of the line
Control-U	clear from the cursor to the beginning of the line
Control-X or F4	toggle safe mode

F7	toggle hot lock mode mode
Control-R or F3	toggle console search
F6	toggle cellar
F1	show context-sensitive help.
Tab	perform line completion. When pressed a second time, show possible completions.
#	Send a message to an internal chat system
Delete	remove character at the cursor
Control-H or Backspace	removes character before cursor and moves the cursor back one position.
Control-\	split line at cursor. Insert newline at the cursor position. Display second of the two resulting lines.
Control-B or Left	move cursor backward one character
Control-F or Right	move cursor forward one character
Control-P or Up	go to the previous line. If this is the first line of input then recall previous input from history.
Control-N or Down	go to the next line. If this is the last line of input then recall the next input from the history
Control-A or Home	move the cursor to the beginning of the line. If the cursor is already at the beginning of the line, then go to the beginning of the first line of the current input
Control-E or End	move the cursor to the end of the line. If the cursor is already at the end of the line, then move it to the end of the last line of the current input
Control-L or F5	reset terminal and repaint screen

MAC server

MAC server section allows you to configure MAC Telnet Server, MAC WinBox Server and MAC Ping Server on RouterOS device.

MAC Telnet is used to provide access to a router that has no IP address set. It works just like IP telnet. MAC telnet is possible between two MikroTik RouterOS routers only.

MAC WinBox is used to provide Winbox access to the router via MAC address.

MAC Ping is used to allow MAC pings to the router's MAC address.

> **MAC-server** settings are included in the "system" package.

MAC Telnet Server

It is possible to set MAC Telnet access to specific interfaces that are a part of the interface list:

```
[admin@device] /tool mac-server set allowed-interface-list=listBridge
[admin@device] /tool mac-server print
  allowed-interface-list: listBridge
```

In the example above, MAC Telnet is configured for the interface list "listBridge" and, as a result, MAC Telnet will only work via the interfaces that are members of the list (you can add multiple interfaces to the list).

To disable MAC Telnet access, issue the command (set "allowed-interface-list" to "none"):

```
[admin@device] /tool mac-server set allowed-interface-list=none
[admin@device] /tool mac-server print
  allowed-interface-list: none
```

You can check active MAC Telnet sessions (that the device accepted) with the command:

```
[admin@device] > tool mac-server sessions print
Columns: INTERFACE, SRC-ADDRESS, UPTIME
#  INTERFACE  SRC-ADDRESS        UPTIME
0  ether5     64:D1:54:FB:E3:E6  17s
```

MAC Telnet Client

When MAC Telnet Server is enabled, you can use another RouterOS device to connect to the server using the mac-telnet client:

```
[admin@device2] > tool mac-telnet B8:69:F4:7F:F2:E7
Login: admin
Password:
Trying B8:69:F4:7F:F2:E7...
Connected to B8:69:F4:7F:F2:E7



  MMM      MMM       KKK                          TTTTTTTTTTT      KKK
  MMMM    MMMM       KKK                          TTTTTTTTTTT      KKK
  MMM MMMM MMM  III  KKK  KKK  RRRRRR     000000      TTT     III  KKK  KKK
  MMM  MM  MMM  III  KKKKK     RRR  RRR  000  000     TTT     III  KKKKK
  MMM      MMM  III  KKK KKK   RRRRRR    000  000     TTT     III  KKK KKK
  MMM      MMM  III  KKK  KKK  RRR  RRR   000000      TTT     III  KKK  KKK

  MikroTik RouterOS 7.1rc3 (c) 1999-2021       https://www.mikrotik.com/

Press F1 for help

[admin@device] >
```

Change the MAC address accordingly (to your setup) and you should get into the server's CLI (as shown in the example above).

MAC Scan

Mac scan feature discovers all devices, which support MAC telnet protocol on the given network. The command requires you to select an interface that should be scanned:

```
[admin@Sw_Denissm] > tool mac-scan interface=all
MAC-ADDRESS       ADDRESS               AGE
B8:69:F4:7F:F2:E7 192.168.69.1           26
2C:C8:1B:FD:F2:C3 192.168.69.3           56
```

In the example, above, all interfaces are chosen, and the scan will run infinitely unless stopped (by pressing "q").

You can also add a "duration" parameter that will dictate for how long the scan should go on:

```
[admin@Sw_Denissm] > tool mac-scan interface=all duration=1
MAC-ADDRESS       ADDRESS               AGE
B8:69:F4:7F:F2:E7 192.168.69.1           48
2C:C8:1B:FD:F2:C3 192.168.69.3           17
```

In the example above, we set the "duration" parameter to 1 second.

MAC WinBox Server

Same as with MAC Telnet, it is possible to set MAC WinBox access to specific interfaces that are a part of the interface list:

```
[admin@device] > tool mac-server mac-winbox set allowed-interface-list=listBridge
[admin@device] > tool mac-server mac-winbox print
  allowed-interface-list: listBridge
```

In the example above, MAC WinBox access is configured for the interface list "listBridge" and, as a result, MAC WinBox will only work via the interfaces that are members of the list.

To disable MAC WinBox access, issue the command (set "allowed-interface-list" to "none"):

```
[admin@device] > tool mac-server mac-winbox set allowed-interface-list=none
[admin@device] > tool mac-server mac-winbox print
  allowed-interface-list: none
```

MAC Ping Server

MAC Ping Server can be either set to be "disabled" or "enabled":

```
[admin@device] > tool mac-server ping print
  enabled: yes
```

You can enable or disable MAC ping with the help of the commands (**enable=yes** → to enable the feature; **enable=no** → to disable the feature):

```
[admin@device] > tool mac-server ping set enabled=yes
[admin@device] > tool mac-server ping set enabled=no
```

When MAC Ping is enabled, other hosts on the same broadcast domain can use ping tool to ping the mac address. For example, you can issue the following command to check MAC ping results:

```
[admin@device] > /ping 00:0C:42:72:A1:B0
HOST                                  SIZE  TTL TIME  STATUS
00:0C:42:72:A1:B0                     56        0ms
00:0C:42:72:A1:B0                     56        0ms
    sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```

MikroTik mobile app

Summary

The application is available for both Android and iOS operating systems. It is a good way to configure a new device, as it provides a simple and user-friendly setup screen for the most basic settings of your new router. It also features an advanced menu, for the more experienced user.

Downloading app

MikroTik application is available to download on App Store and Google Play, please see our web page Software section for direct download links or scan the QR code.

Use the MikroTik smartphone app to configure your router in the field, or to apply the most basic initial settings for your MikroTik home access point.

Quick start with the application

- Scan QR code and choose your preferred OS;
- Install and connect to your wireless network;
- Open application, by default, the IP address and username will be already entered;
- Click Connect to establish a connection to;
- Choose Quick setup, and the application will guide you through all basic configuration settings in a couple of easy steps.

Quick Set

Summary

Quickset is a simple configuration wizard page that prepares your router in a few clicks. It is the first screen a user sees, when opening the default IP address 192.168.88.1 in a web browser.

Quickset is available for all devices that have some sort of default configuration from factory. Devices that do not have configuration must be configured by hand. The most popular and recommended mode is the HomeAP (or HomeAP dual, depending on the device). This Quickset mode provides the simplest of terminology and the most common options for the home user.

Modes

Depending on the router model, different Quickset modes might be available from the Quickset dropdown menu:

- **CAP**: Controlled Access Point, an AP device, that will be managed by a centralized CAPsMAN server. Only use if you have already set up a CAPsMAN server.
- **CPE**: Client device, which will connect to an Access Point (AP) device. Provides option to scan for AP devices in your area.
- **HomeAP**: The default Access Point config page for most home users. Provides fewer options and simplified terminology.
- **HomeAP dual**: Dual band devices (2GHz/5GHz). The default Access Point config page for most home users. Provides fewer options and simplified terminology.
- **Home Mesh**: Made for making bigger WiFi networks. Enables the CAPsMAN server in the router, and places the local WiFi interfaces under CAPsMAN control. Just boot other MikroTik WiFi APs with the reset button pressed, and they will join this HomeMesh network (see their Quick guide for details)
- **PTP Bridge AP**: When you need to transparently interconnect two remote locations together in the same network, set one device to this mode, and the other device to the next (PTP Bridge CPE) mode.
- **PTP Bridge CPE**: When you need to transparently interconnect two remote locations together in the same network, set one device to this mode, and the other device to the previous (PTP Bridge AP) mode.
- **WISP AP**: Similar to the HomeAP mode, but provides more advanced options and uses industry standard terminology, like SSID and WPA.

HomeAP

This is the mode you should use if you would like to quickly configure a home access point.

Wireless

Set up your wireless network in this section:

- **Network Name**: How will your smartphone see your network? Set any name you like here. In HomeAP dual, you can set the 2GHz (legacy) and 5GHz (modern) networks to same, or different names (see FAQ). Use any name you like, in any format.
- **Frequency**: Normally you can leave "Auto", in this way, the router will scan the environment, and select the least occupied frequency channel (it will do this once). Use a custom selection if you need to experiment.
- **Band**: Normally leave this to defaults (2GHz b/g/n and 5GHz A/N/AC).
- **Use Access List (ACL)**: Enable this if you would like to restrict who can connect to your AP, based on the user's MAC (hardware) address. To use this option, first you need to allow these clients to connect, and then use the below button "Copy to ACL". This will copy the selected client to the access list. After you have built an Access list (ACL), you can enable this option to forbid anyone else to attempt connections to your device. Normally you can leave this alone, as the Wireless password already provides the needed restrictions.
- **WiFi Password**: The most important option here. Sets a secure password that also encrypts your wireless communications, which will be needed to connect to wireless network.

- **WPS accept**: Use this button to grant access to a specific device that supports the WPS connection mode. Useful for printers and other peripherals where typing a password is difficult. First start WPS mode in your client device, then once click the WPS button here to allow said device. Button works for a few seconds and operates on a per-client basis.
- **Guest network**: Useful for house guests who don't need to know your main WiFi password. Set a separate password for them in this option. Important! Guest users will not be able to access other devices in your LAN and other guest devices. This mode enabled Bridge filters to prevent this.
- **Wireless clients**: This table shows the currently connected client devices (their MAC address, if they are in your Access List, their last used IP address, how long are they connected, their signal level in dBm and in a bar graph).

Internet

- **Port**: Select which port is connected to the ISP (internet) modem. Usually Eth1.
- **Address Acquisition**: Select how the ISP is giving you the IP address. Ask your service provider about this and the other options (IP address, Netmask, Gateway).
- **MAC address**: Normally should not be changed, unless your ISP has locked you to a specific MAC address, and you have changed the router to a new one.
- **Firewall router**: This enables a secure firewall for your router and your network. Always make sure this box is selected, so that no access is possible to your devices from the internet port.
- **MAC server / MAC Winbox**: Allows connection with the [Winbox utility https://mt.lv/winbox] from the LAN port side in MAC address mode. Useful for debugging and recovery, when IP mode is not available. Advanced use only.
- **Discovery**: Allows the device to be identified by model name from other RouterOS devices.

Local Network

- **IP address**: Mostly can stay at the default 192.168.88.1 unless your router is behind another router. To avoid IP conflict, change to 192.168.89.1 or similar
- **Netmask**: In most situations can leave 255.255.255.0
- **Bridge all LAN ports**: Allows your devices to communicate to each other, even if, say, your TV is connected via Ethernet LAN cable, but your PC is connected via WiFi.
- **DHCP server**: Normally, you would want automatic IP address configuration in your home network, so leave the DHCP settings ON and on their defaults.
- **NAT**: Turn this off ONLY if your ISP has provided a public IP address for both the router and also the local network. If not, leave NAT on.
- **UPnP**: This option enables automatic port forwarding ("opening ports to the local network" as some call it) for supported programs and devices, like your NAS disks and peer-to-peer utilities. Use with care, as this option can sometimes expose internal devices to the internet without your knowledge. Enable only if specifically needed.

VPN

If you want to access your local network (and your router) from the internet, use a secure VPN tunnel. This option gives you a domain name where to connect to, and enables PPTP and L2TP/IPsec (the second one is recommended). The username is 'vpn' and you can specify your own password. All you need to do is enable it here, and then provide the address, username and password in your laptop or phone, and when connected to the VPN, you will have a securely encrypted connection to your home network. Also, useful when travelling - you will be able to browse the internet through a secure line, as if connecting from your home. This also helps to avoid geographical restrictions that are set up in some countries.

System

- **Check for updates**: Always make sure your device is up-to-date with this button. Checks if an updated RouterOS release is available, and installs it.
- **Password**: Sets the password for the device config page itself. Make sure nobody can access your router config page and change the settings.

FAQ

Q: How is Quickset different from the Webfig tab, where a bunch of new menus appear?

A: QuickSet is for new users who only need their device up and running in no time. It provides the most commonly used options in one place. If you need more options, do not use any Quickset settings at all, click on "Webfig" to open the advanced configuration interface. The full functionality is unlocked.

Q: Can I use Quickset and Webfig together? While settings that are not conflicting can be configured this way, it is not recommended to mix up these menus.

A: If you are going to use Quickset, use only Quickset and vice versa. What is difference between Router and Bridge mode? Bridge mode adds all interfaces to the bridge, allowing to forward Layer2 packets (acts as a hub/switch). In Router mode, packets are forwarded in Layer3 by using IP addresses and IP routes (acts as a router).

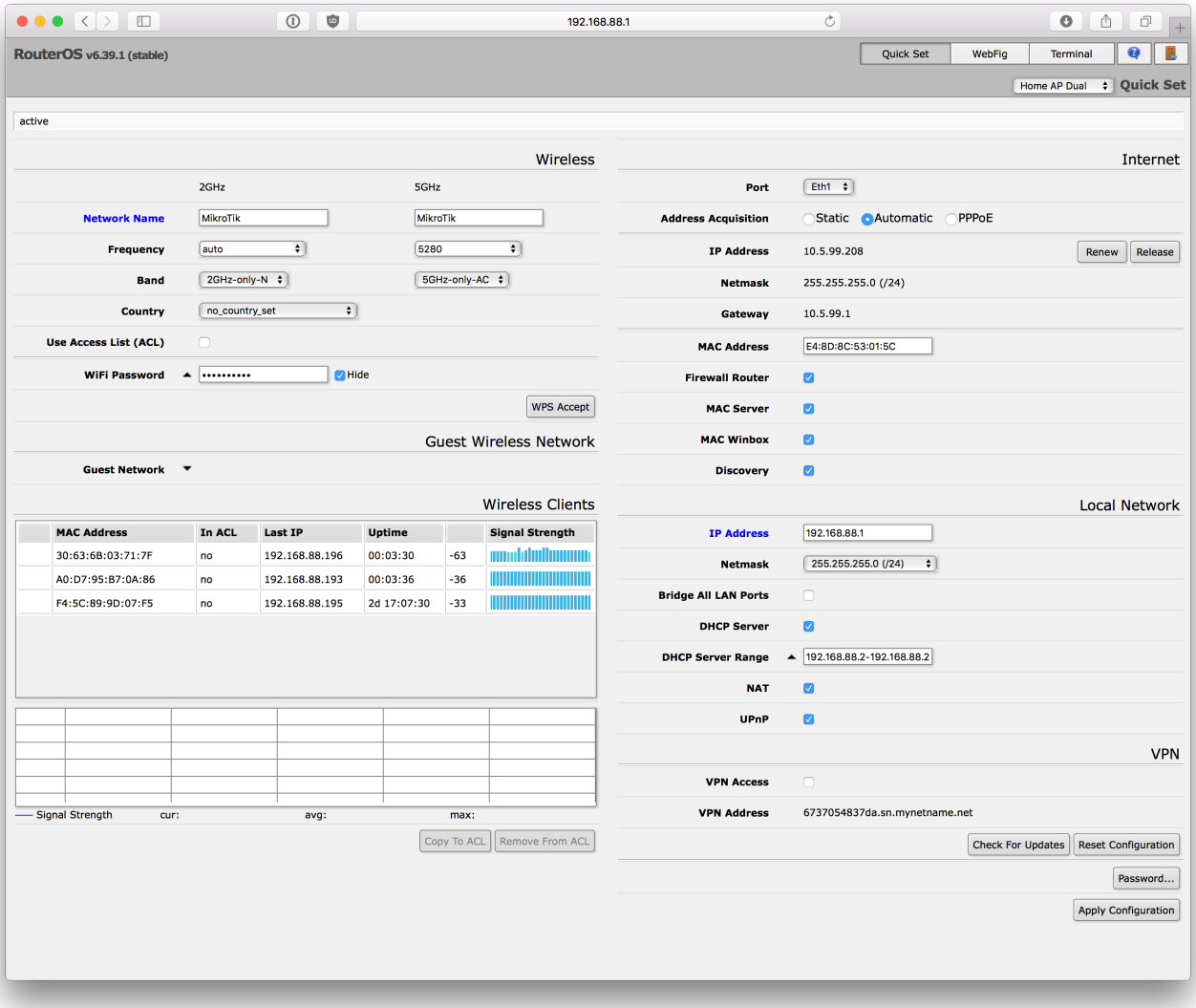
Q: In HomeAP mode, should the 2GHz and 5GHz network names be the same, or different?

A: If you prefer that all your client devices, like TV, phones, game consoles, would automatically select the best preferred network, set the names identically. If you would like to force a client device to use the faster 5GHz 802.11ac connection, set the names unique.

Q: Can I create an AP without security settings - no password or connect to such AP while using QuickSet?

A: QuickSet uses WPA2 pre-shared key by default. It means that the minimal password length is 8 symbols and the device can only connect to WPA2 secured AP or serve as AP itself. For configurations with no security settings, you need to configure them manually using WinBox, Webfig, or console.

QuickSet interface:



REST API

Overview

Watch our video about this feature.

The term "REST API" generally refers to an API accessed via HTTP protocol at a predefined set of resource-oriented URLs.
Starting from **RouterOS v7.1beta4**, it is implemented as a JSON wrapper interface of the console API. It allows to create, read, update and delete resources and call arbitrary console commands.

To start using REST API, the `www-ssl` or `www` (starting with **RouterOS v7.9**) service must be configured and running. When the `www-ssl` service (HTTPS access) is enabled, the REST service can be accessed by connecting to `https://<routers_IP>/rest`. When `www` service (HTTP access) is enabled the REST service can be accessed by connecting to `http://<routers_IP>/rest`.

> We do not advise enabling HTTP access (`www` service). The main risk is that authentication credentials can be read with passive eavesdropping. You can use it only when performing tests (not in a production environment) and when you are certain that nobody can listen in (inspect your traffic)!

The easiest way to start is to use cURL, wget, or any other HTTP client even RouterOS fetch tool.

```
$ curl -k -u admin: https://10.155.101.214/rest/system/resource
[{"architecture-name":"tile","board-name":"CCR1016-12S-1S+",
"build-time":"Dec/04/2020 14:19:51","cpu":"tilegx","cpu-count":"16",
"cpu-frequency":"1200","cpu-load":"1","free-hdd-space":"83439616",
"free-memory":"1503133696","platform":"MikroTik",
"total-hdd-space":"134217728","total-memory":"2046820352",
"uptime":"2d20h12m20s","version":"7.1beta4 (development)"}]
```

Authentication

Authentication to the REST API is performed via HTTP Basic Auth. Provide your Username and password are the same as for the console user (by default "admin" with no password).

> You have to set up certificates to use secure HTTPS, if self-signed certs are used, then CA must be imported to the trusted root. However, for testing purposes, it is possible to connect insecurely (for cUrl use -k flag, for wget use --no-check-certificate).

JSON format

Server broadly follows ECMA-404 standard, with following notes:

- In JSON replies all object values are encoded as strings, even if the underlying data is a number or a boolean.
- The server also accepts numbers in octal format (begins with 0) and hexadecimal format (begins with 0x). If the numbers are sent in a string format, they are assumed to be in decimal format.

- Numbers with exponents are not supported.

HTTP Methods

Below is a table summarising supported HTTP methods

HTTP Verv	CRUD	ROS	Description
GET	Read	print	To get the records.
PATCH	Update/Modify	set	To update a single record.
PUT	Create	add	To create a new record.
DELETE	Delete	remove	To delete a single record.
POST			Universal method to get access to all console commands.

GET

This method allows getting the list of all records or a single record from the specified menu encoded in the URL.
For example, get all IP addresses (equivalent to the '`ip/address/print`' command from the CLI):

```
$ curl -k -u admin: https://10.155.101.214/rest/ip/address
[{".id":"*1","actual-interface":"ether2","address":"10.0.0.111/24","disabled":"false",
"dynamic":"false","interface":"ether2","invalid":"false","network":"10.0.0.0"},
{".id":"*2","actual-interface":"ether3","address":"10.0.0.109/24","disabled":"true",
"dynamic":"false","interface":"ether3","invalid":"false","network":"10.0.0.0"}]
```

To return a single record, append the ID at the end of the URL:

```
$ curl -k -u admin: https://10.155.101.214/rest/ip/address/*1
{".id":"*1","actual-interface":"ether2","address":"10.0.0.111/24","disabled":"false",
"dynamic":"false","interface":"ether2","invalid":"false","network":"10.0.0.0"}
```

If table contains named parameters, then name instread of ID can be used, for example, get ether1:

```
$ curl -k -u admin: https://10.155.101.214/rest/interface/ether1
```

It is also possible to filter the output, for example, return only valid addresses that belong to the 10.155.101.0 network:

```
$ curl -k -u admin: "https://10.155.101.214/rest/ip/address?network=10.155.101.0&dynamic=true"
[{".id":"*8","actual-interface":"sfp12","address":"10.155.101.214/24","disabled":"false",
"dynamic":"true","interface":"sfp12","invalid":"false","network":"10.155.101.0"}]
```

Another example returns only addresses on the "dummy" interface and with the comment "test":

```
$ curl -k -u admin: 'https://10.155.101.214/rest/ip/address?comment=test&interface=dummy'
[{".id":"*3","actual-interface":"dummy","address":"192.168.99.2/24","comment":"test",
"disabled":"false","dynamic":"false","interface":"dummy","invalid":"false","network":"192.168.99.0"}]
```

If you want to return only specific properties, you can use the '`.proplist`', followed by the '=' and a list of comma-separated properties. For example, to show only the address and if it's disabled:

```
$ curl -k -u admin: https://10.155.101.214/rest/ip/address?.proplist=address,disabled
[{"address":"10.0.0.111/24","disabled":"false"},{"address":"10.0.0.109/24","disabled":"true"}]
```

PATCH

This method is used to update a single record. Set PATCH call body as a JSON object which contains fields and values of the properties to be updated. For example, add a comment:

```
$ curl -k -u admin: -X PATCH https://10.155.101.214/rest/ip/address/*3 \
  --data '{"comment": "test"}' -H "content-type: application/json"
{".id":"*3","actual-interface":"dummy","address":"192.168.99.2/24","comment":"test",
"disabled":"false","dynamic":"false","interface":"dummy","invalid":"false","network":"192.168.99.0"}
```

In case of a successful update, the server returns the updated object with all its parameters.

PUT

A method is used to create new records in the menu encoded in the URL. The body should be set as a JSON object containing parameters applied to the newly created record.

In case of success, the server returns the created object with all its parameters.

Only one resource can be created in a single request.

For example, add an IP address to a dummy interface:

```
$ curl -k -u admin: -X PUT https://10.155.101.214/rest/ip/address \
  --data '{"address": "192.168.111.111", "interface": "dummy"}' -H "content-type: application/json"
{".id":"*A","actual-interface":"dummy","address":"192.168.111.111/32","disabled":"false",
"dynamic":"false","interface":"dummy","invalid":"false","network":"192.168.111.111"}
```

DELETE

This method is used to delete the record with a specified ID from the menu encoded in the URL. If the deletion has been succeeded, the server responds with an empty response. For example, call to delete the record twice, on second call router will return 404 error:

```
$ curl -k -u admin: -X DELETE https://10.155.101.214/rest/ip/address/*9
$ curl -k -u admin: -X DELETE https://10.155.101.214/rest/ip/address/*9
{"error":404,"message":"Not Found"}
```

POST

All the API features are available through the `POST` method. The command word is encoded in the header and optional parameters are passed in the JSON object with the corresponding fields and values. For example, to change the password of the active user, send

```
POST https://router/rest/password
{"old-password":"old","new-password":"N3w", "confirm-new-password":"N3w"}
```

REST response is structured similar to API response:

- If the response contains '`!re`' sentences (records), the JSON reply will contain a list of objects.
- If the '`!done`' sentence contains data, the JSON reply will contain an object with the data.
- If there are no records or data in the '`!done`' sentence, the response will hold an empty list.

There are two special keys: `.proplist` and `.query`, which are used with the `print` command word. Read more about APIs responses, prop lists, and queries in the API documentation.

Proplist

The '`.proplist`' key is used to create `.proplist` attribute word. The values can be a single string with comma-separated values:

```
POST https://router/rest/interface/print
{".proplist":"name,type"}
```

or a list of strings:

```
POST https://router/rest/interface/print
{".proplist":["name","type"]}
```

For example, return address and interface properties from the ip/address list:

```
$ curl -k -u admin: -X POST https://10.155.101.214/rest/ip/address/print\
  --data '{"_proplist": ["address","interface"]}' -H "content-type: application/json"
[{"address":"192.168.99.2/24","interface":"dummy"},
{"address":"172.16.5.1/24","interface":"sfpplus1"},
{"address":"172.16.6.1/24","interface":"sfp2"},
{"address":"172.16.7.1/24","interface":"sfp3"},
{"address":"10.155.101.214/24","interface":"sfp12"},
{"address":"192.168.111.111/32","interface":"dummy"}]
```

Query

The '`.query`' key is used to create a query stack. The value is a list of query words. For example this POST request :

```
POST https://router/rest/interface/print
{".query":["type=ether","type=vlan","#|!"]}
```

is equivalent to this API sentence

```
/interface/print
?type=ether
?type=vlan
?#|!
```

For example, let's combine '*query*' and '*proplist*', to return '.id', 'address', and 'interface' properties for all dynamic records and records with the network 192.168.111.111

```
$ curl -k -u admin: -X POST https://10.155.101.214/rest/ip/address/print \
  --data '{".proplist": [".id","address","interface"], ".query": ["network=192.168.111.111","dynamic=true","
#|"]}'\
  -H "content-type: application/json"
[{".id":"*8","address":"10.155.101.214/24","interface":"sfp12"},
{".id":"*A","address":"192.168.111.111/32","interface":"dummy"}]
```

Timeout

If the command runs indefinitely, it will timeout and the connection will be closed with an error. The current timeout interval is 60 seconds. To avoid timeout errors, add a parameter that would sufficiently limit the command execution time.

> ⚠ Timeout is not affected by the parameters passed to the commands. If the command is set to run for an hour, it will terminate early and return an error message.

For example, let's see what we get when the ping command exceeds the timeout and how to prevent this by adding a count parameter:

```
$ curl -k -u admin: -X POST https://10.155.101.214/rest/ping \
  --data '{"address":"10.155.101.1"}' \
  -H "content-type: application/json"
{"detail":"Session closed","error":400,"message":"Bad Request"}

$ curl -k -u admin: -X POST https://10.155.101.214/rest/ping \
  --data '{"address":"10.155.101.1","count":"4"}' \
  -H "content-type: application/json"
[{"avg-rtt":"453us","host":"10.155.101.1","max-rtt":"453us","min-rtt":"453us","packet-loss":"0","received":"1","
sent":"1","seq":"0","size":"56","time":"453us","ttl":"64"},
{"avg-rtt":"417us","host":"10.155.101.1","max-rtt":"453us","min-rtt":"382us","packet-loss":"0","received":"2","
sent":"2","seq":"1","size":"56","time":"382us","ttl":"64"},
{"avg-rtt":"495us","host":"10.155.101.1","max-rtt":"650us","min-rtt":"382us","packet-loss":"0","received":"3","
sent":"3","seq":"2","size":"56","time":"650us","ttl":"64"},
{"avg-rtt":"461us","host":"10.155.101.1","max-rtt":"650us","min-rtt":"359us","packet-loss":"0","received":"4","
sent":"4","seq":"3","size":"56","time":"359us","ttl":"64"}]
```

Another example is a bandwidth test tool, which can be limited by providing run duration:

```
$ curl -k -u admin: -X POST 'https://10.155.101.214/rest/tool/bandwidth-test' \
  --data '{"address":"10.155.101.1","duration":"2s"}' \
  -H "content-type: application/json"
[{".section":"0","connection-count":"20","direction":"receive","lost-packets":"0",
"random-data":"false","rx-10-second-average":"0","rx-current":"0","rx-size":"1500",
"rx-total-average":"0",
"status":"connecting"},
{".section":"1","connection-count":"20","direction":"receive","duration":"1s",
"lost-packets":"0","random-data":"false","rx-10-second-average":"0","rx-current":"0",
"rx-size":"1500","rx-total-average":"0",
"status":"running"},
{".section":"2","connection-count":"20","direction":"receive","duration":"2s",
"lost-packets":"581175","random-data":"false","rx-10-second-average":"854372352",
"rx-current":"854372352","rx-size":"1500","rx-total-average":"854372352",
"status":"running"},
{".section":"3","connection-count":"20","direction":"receive","duration":"3s",
"lost-packets":"9014","random-data":"false","rx-10-second-average":"891979008",
"rx-current":"929585664","rx-size":"1500","rx-total-average":"891979008",
"status":"done testing"}]
```

Errors

The success or failure of the API calls is indicated in the HTTP status code. In case of failure (status code 400 or larger), the body of the response contains a JSON object with the error code, a description of the error, and optional error details. For example, trying to delete an interface will return

```
{"error":406,"message":"Not Acceptable","detail":"no such command or directory (remove)"}
```

RoMON

Summary

RoMON stands for "Router Management Overlay Network". RoMON works by establishing an independent MAC layer peer discovery and data forwarding network. RoMON packets are encapsulated with EtherType 0x88bf and DST-MAC 01:80:c2:00:88:bf and its network operate independently of L2 or L3 forwarding configuration. When RoMON is enabled, any received RoMON packets will not be displayed by sniffer or torch tools.

Each router on the RoMON network is assigned its RoMON ID. RoMON ID can be selected from the port MAC address or specified by the user.

RoMON protocol does not provide encryption services. Encryption is provided at the "application" level, by e.g. using ssh or by using a secure WinBox.

Secrets

RoMON protocol secrets are used for message authentication, integrity check and replay prevention by means of hashing message contents with MD5.

For each interface, if the interface-specific secret list is empty, a global secret list is used. When sending out, messages are hashed with the first secret in list if list is not empty and first is not "empty secret" (empty string = ""), otherwise, messages are sent unhashed. When received, unhashed messages are only accepted if a secret list is empty or contains "empty secret", hashed messages are accepted if they are hashed with any of the secrets in list.

This design allows for the incremental introduction and/or change of secrets in-network without RoMON service interruption and can happen over RoMON itself, e.g.:

- initially, all routers are without secrets;
- configure each router one by one with secrets="","mysecret" - this will make all routers still send unprotected frames, but they all will be ready to accept frames protected with secret "mysecret";
- configure each router one by one with secrets="mysecret","" - this will make all routers use secret "mysecret", but also still accept unprotected frames (from routers that have not yet been changed);
- configure each router with secrets="mysecret" - this will make all routers use secret "mysecret" and also only accept frames protected with "mysecret";

Changing of secret in a network should be performed in a similar fashion where for some time both secrets are in use in network.

Peer discovery

In order to discover all routers on RoMON network RoMON discover command must be used:

```
[admin@MikroTik] > /tool/romon/discover
Flags: A - active
Columns: ADDRESS, COST, HOPS, PATH, L2MTU, IDENTITY, VERSION, BOARD
   ADDRESS            COS  H  PATH               L2MT  IDENTITY   VERSION    BOARD
A  6C:3B:6B:48:0E:8B  200  1  6C:3B:6B:48:0E:8B  1500  hEX        6.47beta7  RB750Gr3
A  6C:3B:6B:ED:83:69  200  1  6C:3B:6B:ED:83:69  1500  CCR1009    6.47beta7  CCR1009-7G-1C-1S+
A  B8:69:F4:B3:1B:D2  200  1  B8:69:F4:B3:1B:D2  1500  4K11       6.47beta7  RB4011iGS+5HacQ2HnD
A  CC:2D:E0:26:22:4D  200  1  CC:2D:E0:26:22:4D  1500  CCR1036    6.47beta7  CCR1036-8G-2S+
A  CC:2D:E0:8D:01:88  200  1  CC:2D:E0:8D:01:88  1500  CRS328     6.47beta7  CRS328-24P-4S+
A  E4:8D:8C:1C:D3:0E  200  1  E4:8D:8C:1C:D3:0E  1500  MikroTik   6.47beta7  RB2011iLS
A  E4:8D:8C:49:49:DB  200  1  E4:8D:8C:49:49:DB  1500  hAP        6.47beta7  RB962UiGS-5HacT2HnT
```

Configuration Examples

In order for a device to participate in the RoMON network, the RoMON feature must be enabled and ports that participate in the RoMON network must be specified.

```
/tool romon set enabled=yes secrets=testing
```

Ports that participate in the RoMON network are configured in **the RoMON port** menu. Port list is a list of entries that match either specific port or all ports and specifies if matching port(s) is forbidden to participate in the RoMON network and in case port is allowed to participate in RoMON network entry also specifies the port cost. Note that all specific port entries have higher priority than the wildcard entry with **interface=all**.

For example, the following list specifies that all ports participate in RoMON network with cost 100 and ether7 interface with cost 200:

```
[admin@MikroTik] > /tool/romon/port/print
Flags: * - default
Columns: INTERFACE, FORBID, COST
#     INTERF  FO  COS
0  *  all     no  100
1     ether7  no  200
```

By default one wildcard entry with **forbid=no** and **cost=100** is created.

Applications

Multiple applications can be run over the RoMON network.

In order to test the reachability of specific router on RoMON network RoMON ping command can be used:

```
[admin@MikroTik] > /tool/romon/ping id=6C:3B:6B:48:0E:8B count=5
  SEQ HOST                                     TIME  STATUS
    0 6C:3B:6B:48:0E:8B                        1ms
    1 6C:3B:6B:48:0E:8B                        0ms
    2 6C:3B:6B:48:0E:8B                        1ms
    3 6C:3B:6B:48:0E:8B                        0ms
    4 6C:3B:6B:48:0E:8B                        1ms
    sent=5 received=5 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=1ms
```

In order to establish a secure terminal connection to router on RoMON network RoMON SSH command can be used:

```
[admin@MikroTik] > /tool/romon/ssh 6C:3B:6B:48:0E:8B
```

Run RoMON in WinBox by using CLI

In order to establish the RoMON session directly by using the command line on a computer, you must specify RoMON agents and desired routers addresses. RoMON agent must be saved on Managed routers list in WinBox in order to make a successful connection:

```
winbox.exe --romon 192.168.88.1 6C:3B:6B:48:0E:8B admin ""
```

Serial Console

Overview

The Serial Console and Serial Terminal are tools, used to communicate with devices and other systems that are interconnected via the serial port. The serial terminal may be used to monitor and configure many devices - including modems, network devices (including MikroTik routers), and any device that can be connected to a serial (asynchronous) port.

The Serial Console feature is for configuring direct-access configuration facilities (monitor/keyboard and serial port) that are mostly used for initial or recovery configuration. A special null-modem cable is needed to connect two hosts (like two PCs, or two routers; not modems). Note that a terminal emulation program (e.g., HyperTerminal on Windows or minicom on Linux) is required to access the serial console from another computer. Default settings of the router's serial port are 115200 bits/s (for x86 default is 9600 bits/s), 8 data bits, 1 stop bit, no parity, hardware (RTS/CTS) flow control.

Several customers have described situations where the Serial Terminal (managing side) feature would be useful:

- on a mountaintop, where a MikroTik wireless installation sits next to equipment (including switches and Cisco routers) that can not be managed in-band (by telnet through an IP network)
- monitoring weather-reporting equipment through a serial port
- connection to a high-speed microwave modem that needed to be monitored and managed by a serial connection

With the serial-terminal feature of the MikroTik, up to 132 (and, maybe, even more) devices can be monitored and controlled.

Serial Console Connections

Serial communications between devices are done with RS232, it is one of the oldest and most widely spread communication methods in the computer world. It was used for communication with the modems or other peripheral devices DTE/DCE. In the modern world, the main use of serial communication is DTE/DTE communication (Data Terminal Equipment) e.g. using a null-modem cable. There are several types of null modem cables and some of them may not work with RouterBoards at all.

Null Modem Without Handshake

This cable does not utilize handshake pins at all:

Side1 (DB9f)	Side2 (DB9f)	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	GND

It allows data-only traffic on the cross-connected Rx/Tx lines. Hardware flow control is not possible with this type of cable. The only way to perform flow control is with software flow control using the XOFF and XON characters.

Null Modem With Loopback Handshake

The problem with the first cable is when connected to a device on which hardware flow control is enabled software may hang when checking modem signal lines.

Null modem cable with loop back handshake fixes the problem, its main purpose is to fool well-defined software into thinking there is handshaking available:

Side1 (DB9f)	Side2 (DB9f)	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	GND
1+4+6	-	DTR → CD + DSR
-	1+4+6	DTR → CD + DSR
7+8	-	RTS → CTS
-	7+8	RTS → CTS

Hardware flow control is not possible with this cable. Also if remote software does not send its own ready signal to DTR output communication will hang.

Null Modem With Partial Handshake

This cable can be used when flow control enabled without being incompatible with the original way flow control was used with DTE/DCE communication.

This type of cable is not recommended for use with RouterOS.

Side1 (DB9f)	Side2 (DB9f)	Function
1	7+8	RTS2 → CTS2 + CD1
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	GND
6	4	DSR ← DTR
7+8	1	RTS1 → CTS1 + CD2

Null Modem With Full Handshake

Used with special software and should not be used with RouterOS.

Side1 (DB9f)	Side2 (DB9f)	Function
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	GND
6	4	DSR ← DTR
7	8	RTS → CTS
8	7	CTS ← RTS

Null Modem Compatibility

Summary tables below will allow you to choose the proper cable for your application.

	No handshake	Loopback handshake	Partial handshake	Full handshake
RouterBoards with limited port functionality	Y	Y	N*	N
RouterBoards with full functionality	Y	Y	Y	N

* - may work only when hardware flow control is disabled

	No handshake	Loopback handshake	Partial handshake	Full handshake
Software flow control only	Y	Y*	Y**	Y**
Low-speed DTE/DCE compatible hardware flow control	N	Y	Y*	N
High-speed DTE/DCE compatible hardware flow control	N	Y	Y**	N
High speed communication using special software	N	N	Y*	Y

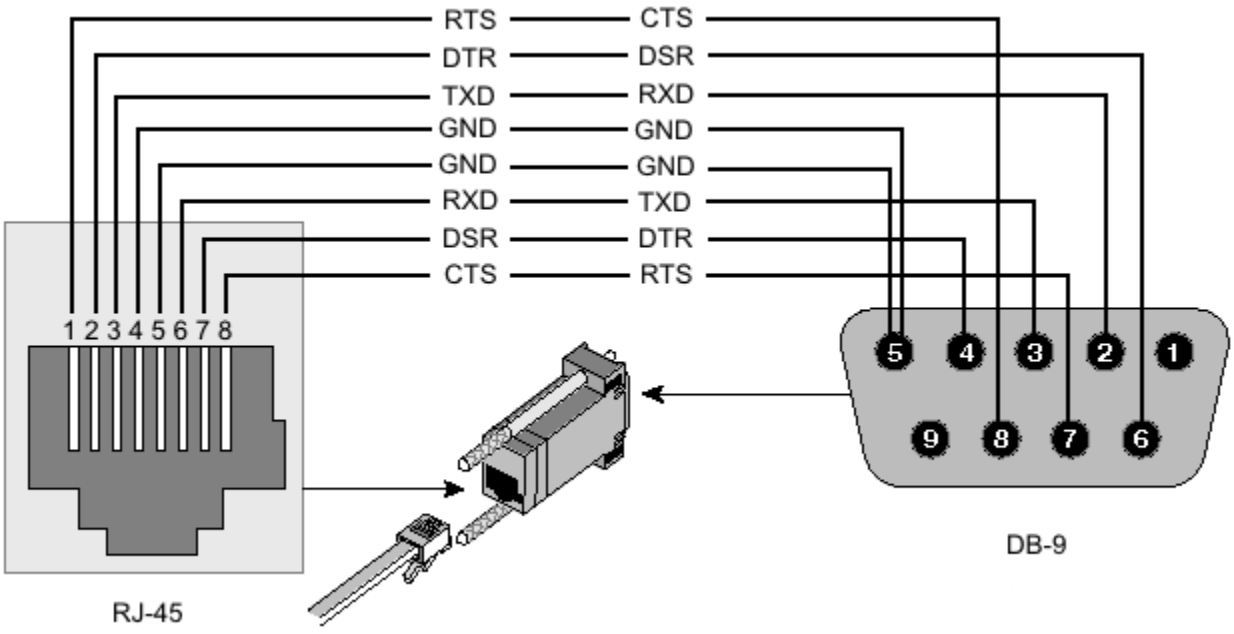
* - will work as an alternative

** - will work but not recommended

RJ45 Type Serial Port

This type of port is used on RouterBOARD 2011, 3011, 4011, CCR1072, CCR1036 r2, CCR2xxx and CRS series devices, sometimes called "Cisco style" serial port.

RJ45 to DB9 Cable Pinout:



Signal	Console Port (DTE) RJ-45	RJ-45 Rolled Cable RJ-45 Pin	Adapter DB-9 Pin	Adapter DB-25 Pin	Signal
RTS	1	8	8	5	CTS
DTR	2	7	6	6	DSR
TxD	3	6	2	3	RxD
Ground	4	5	5	7	Ground
Ground	5	4	5	7	Ground
RxD	6	3	3	2	TxD
DSR	7	2	4	20	DTR
CTS	8	1	7	4	RTS

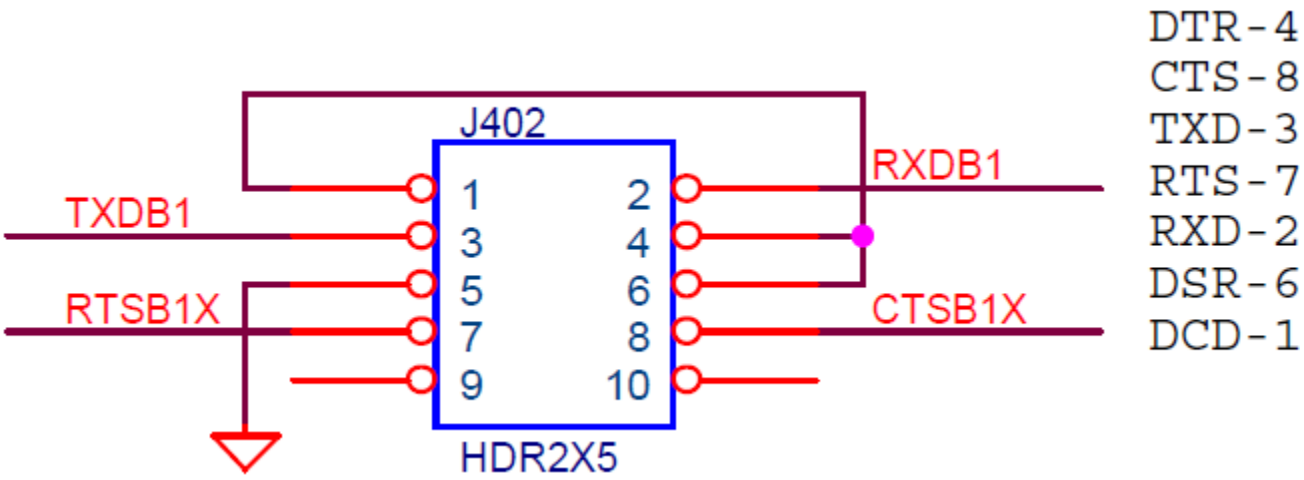
RB M33G Additional Serial Header

For RBM33G additional serial header can be attached on GPIO pins U3_RXD, GND, U3_TXD, and 3V3

CCR Serial Header

The Cloud Core Router series devices have a serial header on the PCB board, called J402 or 100

Here is the pin-out of that connector:



Serial Terminal Usage

RouterOS allows to communicate with devices and other systems that are connected to the router via the serial port using a `/system serial-terminal` command. All keyboard input will be forwarded to the serial port and all data from the port is output to the connected device.

First, you have to have a free serial port, if the device has only one serial port (like all RouterBoards, WRAP/ALIX boards, etc.) you will have to disable the system console on this serial port to be able to use it as **Serial Terminal** for connection to other equipment (switches, modems, etc):

```
/system console disable 0
```

Be sure to just disable the console rather than removing it, as RouterOS will recreate the console after the next reboot when you really remove it.

Note that there are some caveats you should be aware of! Take your time understanding those limits to avoid strange things to happen when connecting a device to a serial port on a RouterBoard:

- By re-configuring port Serial0 on a RouterBoard as seen above, you will lose your serial console access to RouterOS. This means, that if you cannot access your RouterBoard over the network anymore, you might even have to reset the whole configuration of it to gain access again.
- When rebooting a RouterBoard the boot loader (RouterBOOT) will always use the serial console (Serial0 on RouterBoards) to send out some startup messages and offer access to the RouterBOOT menu.

  Having text coming out of the serial port to the connected device might confuse your attached device. Furthermore, in the standard config, you can enter the RouterBOOT menu by pressing **ANY** key. So if your serial device sends any character to the serial port of your RouterBoard during boot time, the RouterBoard will enter the RouterBOOT menu and will **NOT** boot RouterOS unless you manually intervene!

  You can reconfigure RouterBOOT to enter the RouterBOOT menu only when a **DEL** character is received - use this to reduce the chance to get a router that's stuck when rebooting!

  Or if newer versions are used "Silent boot" feature can be used to suppress any output on the serial interface, including removal of booting sounds.

Next, you will have to configure your serial port according to the serial port settings of the connected device. Using the following command you will set your serial port to 19200 Baud 8N1. What settings you need to use depends on the device you connect:

```
/port set serial0 baud-rate=19200 data-bits=8 parity=none stop-bits=1
```

You can also try to let RouterOS guess the needed baud rate by setting

```
/port set serial0 baud-rate=auto
```

Now's the time to connect your device if not already done. Usually, you will have to use a null modem cable (the same thing as a cross-over-cable for Ethernet). Now we're ready to go:

```
/system serial-terminal serial0
```

This will give you access to the device you connected to port Serial0. ***Ctrl-A*** is the prefix key, which means that you will enter a small "menu". If you need to send the ***Ctrl-A*** character to a remote device, press ***Ctrl-A*** twice.

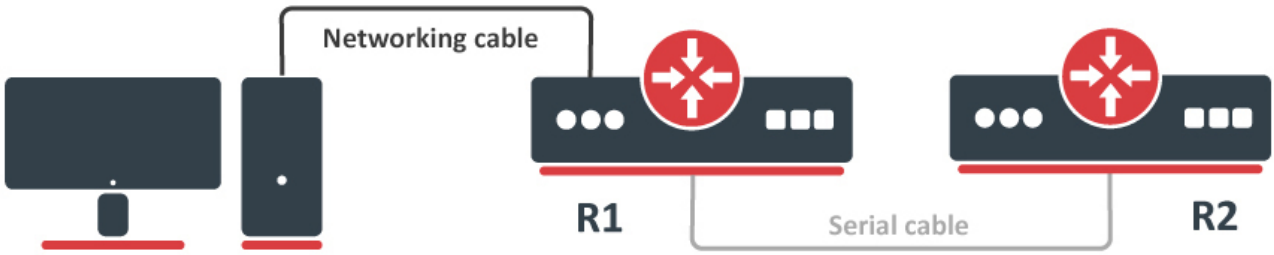
If you want to exit the connection to the serial device type ***Ctrl-A***, then ***Q***. This will return you to your RouterOS console.

Do not connect to devices at an incorrect speed and avoid dumping binary data.

Special Login

Special login can be used to access another device (like a switch, for example) that is connected through a serial cable by opening a telnet/ssh session that will get you directly on this device (without having to login to RouterOS first).

For demonstration we will use two RouterBoards and one PC.



Routers R1 and R2 are connected with serial cable and PC is connected to R1 via ethernet. Lets say we want to access router R2 via serial cable from our PC. To do this you have to set up serial interface proxy on R1. It can be done by feature called **special-login**.

> By default console is bound to serial port.

First task is to unbind console from serial simply by disabling entry in /system console menu:

```
[admin@MikroTik] /system console> print
Flags: X - disabled, U - used, F - free
 #   PORT                                                                TERM
 0 X serial0                                                             vt102
```

Next step is to add new user, in this case *serial*, and bind it to the serial port

```
[admin@MikroTik] > /user add name=serial group=full
[admin@MikroTik] > /special-login add user=serial port=serial0 disabled=no
[admin@MikroTik] > /special-login print
Flags: X - disabled
 #   USER                                                                PORT
 0   serial                                                              serial0
```

Now we are ready to access R2 from our PC.

```
maris@bumba:/$ ssh serial@10.1.101.146

[Ctrl-A is the prefix key]
R2 4.0beta4
R2 Login:

[admin@R2] >
```

To exit special login mode press Ctrl+A and Q

```
[admin@MikroTik] >
[Q - quit connection]      [B - send break]
[A - send Ctrl-A prefix]   [R - autoconfigure rate]


Connection to 10.1.101.146 closed.
```

> After router reboot and serial cable attached router may stuck at Bootloader main menu

To fix this problem you need to allow access bootloader main menu from <any> key to <delete>:

- enter bootloader menu
- press 'k' for boot key options
- press '2' to change key to <delete>

```
What do you want to configure?
d - boot delay
k - boot key
s - serial console
n - silent boot
o - boot device
u - cpu mode
f - cpu frequency
r - reset booter configuration
e - format nand
g - upgrade firmware
i - board info
p - boot protocol
b - booter options
t - call debug code
l - erase license
x - exit setup
your choice: k - boot key

Select key which will enter setup on boot:
 * 1 - any key
   2 - <Delete> key only

your choice: 2
```

SSH

SSH Server

RouterOS has built in SSH server that is enabled by default and is listening for incoming connections on port TCP/22. It is possible to change the port and disable the server under Services menu.

Properties

Sub-menu: `/ip ssh`

Property	Description
allow-none-crypto (*yes*/*no*; Default: **no**)	Whether to allow connection if cryptographic algorithms are set to none.
always-allow-password-login (*yes* / *no*; Default: **no**)	Whether to allow password login at the same time when public key authorization is configured for a user.
forwarding-enabled (*both* / *local* / *no* / *remote*; Default: **no**)	Allows to control which SSH forwarding method to allow: • no - SSH forwarding is disabled; • local - Allow SSH clients to originate connections from the server(router), this setting controls also dynamic forwarding; • remote - Allow SSH clients to listen on the server(router) and forward incoming connections; • both - Allow both local and remote forwarding methods.
host-key-size (*1024* / *1536* / *2048* / *4096* / *8192*; Default: **2048**)	RSA key size when host key is being regenerated.
host-key-type (*ed25519* / *rsa*; Default: **rsa**)	Select host key type
strong-crypto (*yes* / *no*; Default: **no**)	Use stronger encryption, HMAC algorithms, use bigger DH primes and disallow weaker ones: • use 256 and 192 bit encryption instead of 128 bits; • disable null encryption; • use sha256 for hashing instead of sha1; • disable md5; • use 2048bit prime for Diffie-Hellman exchange instead of 1024bit.

Commands

Property	Description
export-host-key (*key-file-prefix*)	Export public and private RSA/Ed25519 to files. Command takes one parameter: • **key-file-prefix** - used prefix for generated files, for example, prefix 'my' will generate files 'my_rsa', 'my_rsa.pub' etc.

import-host-key (*private-key-file*)	Import and replace private RSA key from specified file. Command takes one parameter: • **private-key-file** - name of the private RSA key file
regenerate-host-key ()	Generated new and replace current set of private keys (RSA/Ed25519) on the router. Be aware that previously imported keys might stop working.

⚠ Exporting the SSH host key requires "sensitive" user policy.

Enabling PKI authentication

Example of importing public key for user *admin*

Generate SSH keys on the client device (the device you will connect from). Upload the public SSH key to the router and import it.

```
/user ssh-keys import public-key-file=id_rsa.pub user=admin
```

SSH Client

Sub-menu: `/system ssh`

Simple log-in to remote host

It is able to connect to remote host and initiate ssh session. IP address supports both IPv4 and IPv6.

```
/system ssh 192.168.88.1
/system ssh 2001:db8:add:1337::beef
```

In this case user name provided to remote host is one that has logged into the router. If other value is required, then *user=<username>* has to be used.

```
/system ssh 192.168.88.1 user=lala
/system ssh 2001:db8:add:1337::beef user=lala
```

Log-in from certain IP address of the router

For testing or security reasons it may be required to log in to other host using certain source address of the connection. In this case *src-address=<ip address>* argument has to be used. Note that IP address in this case supports both, IPv4 and IPv6.

```
/system ssh 192.168.88.1 src-address=192.168.89.2
/system ssh 2001:db8:add:1337::beef src-address=2001:db8:bad:1000::2
```

in this case, ssh client will try to bind to address specified and then initiate ssh connection to remote host.

Log-in using RSA public/private key

Example of importing private key for user *admin*

First, export currently generated SSH keys to a file:

```
/ip ssh export-host-key key-file-prefix=admin
```

Two files *admin_rsa* and *admin_rsa.pub* will be generated. The pub file needs to be trusted on the SSH server side (how to enable SSH PKI on RouterOS) The private key has to be added for the particular user.

```
/user ssh-keys private import user=admin private-key-file=admin_rsa
```

Only user with full rights on the router can change 'user' attribute value under */user ssh-keys private*

After the public key is installed and trusted on the SSH server, a PKI SSH session can be created.

```
/system ssh 192.168.1.1
```

Executing remote commands

To execute remote command it has to be supplied at the end of log-in line

```
/system ssh 192.168.88.1 "/ip address print"
/system ssh 192.168.88.1 command="/ip address print"
/system ssh 2001:db8:add:1337::beef "/ip address print"
/system ssh 2001:db8:add:1337::beef command="/ip address print"
```

If the server does not support pseudo-tty (ssh -T or ssh host command), like MikroTik ssh server, then it is not possible to send multiline commands via SSH

For example, sending command `"/ip address \n add address=1.1.1.1/24"` to MikroTik router will fail.

If you wish to execute remote commands via **scripts** or **scheduler**, use command **ssh-exec**.

SSH exec

Sub-menu: `/system ssh-exec`

Command *ssh-exec* is a non-interactive ssh command, thus allowing to execute commands remotely on a device via scripts and scheduler.

Retrieve information

The command will return two values:

- **exit-code**: returns 0 if the command execution succeeded
- **output**: returns the output of remotely executed command

Example: Code below will retrieve interface status of ether1 from device 10.10.10.1 and output the result to "Log"

```
:local Status ([/system ssh-exec address=10.10.10.1 user=remote command=":put ([/interface ethernet monitor
[find where name=ether1] once as-value]->\"status\")" as-value]->"output")
:log info $Status
```

For security reasons, plain text password input is not allowed. To ensure safe execution of the command remotely, use SSH PKI authentication for users on both sides.

The user group and script policy executing the command requires **test** permission

TR-069

TR069-client implements CPE WAN Management Protocol (CWMP) for remote device management, which is standardized by the Broadband Forum (BBF). CWMP works over IP network using HTTP(S) to communicate with an Auto Configuration Server (ACS), which can monitor, configure attributes and update the firmware of a remote device. Typically used by ISPs to manage CPEs, but also can be used for Network Infrastructure Device management.

Requires tr069-client package.

Configuration Settings

Sub-menu: `/tr069-client`

TR069-client menu parameters:

Writable Settings

Client configuration settings.

Property	Description
enabled	enable/disable CWMP protocol
acs-url	URL of ACS. Examples: "https://example.com:8080/path/", "https://192.168.1.100/"
username	HTTP authentication username (used by CPE to "login" into ACS)
password	HTTP authentication password (used by CPE to "login" into ACS)
periodic-inform-enabled	enable/disable CPE periodical session initiation. Timer is started after every successful session. When session is started by periodic interval then Inform RPC contains "2 PERIODIC" event. Maps to "Device.ManagementServer.PeriodicInformEnable" Parameter
periodic-inform-interval	timer interval of periodic inform. Maps to "Device.ManagementServer.PeriodicInformInterval"
client-certificate	certificate of client/CPE, which can be used by ACS for extra authentication

Read-only Settings

Read only parameters to monitor state of the client.

Property	Description
status	informative status of CWMP. • disabled - protocol disabled, • waiting-URL - protocol enabled, but ACS URL not configured, • running - CWMP is configured correctly and will communicate with ACS on events
last-session-error	user-friendly error description indicating why the previous session didn't finish successfully
retry-count	consecutive unsuccessful session count. If > 0, then last-session-error should indicate error. Resets to 0 on a successful session, disabled protocol or reboot

Commands

Command	Description
reset-tr069-config	completely resets and forgets tr069-client configuration and state (without affecting other ROS configurations). Use when CWMP goes into unresponsive/hanged state and should be restored without re-installation of the RouterOS.

CWMP Session

CWMP client usually starts communication(Session) with ACS on different events - first boot, reboot, periodic interval, remote request, value change etc. In each session, CPE and ACS can call RPCs to be "executed" on the other side. CPE always starts with Inform RPC, which contains connection reason, device info and some Parameter values depending on configuration. When CPE has nothing more to say, then ACS executes its RPCs (which most of the time are Parameter management RPCs).

Parameters and Data Models

Parameters are simple name+value pairs and each vendor can decide which Parameters to support in its devices. A combination of all supported Parameters is called Data Model (DM). BBF defines three root Data Models(TR-098, TR-181:1, TR-181:2) on which vendors should base their supported Parameters. **RouterOS Data Model is based on "TR-181 Issue 2 Amendment 11"**, which is the newest DM and recommended by BBF.

RouterOS TR069 client supported parameter reference document:

File	Modified
HTML File current.html v7.13 - RouterOS TR069 client supported parameter reference document	Feb 28, 2024 by Confluence Helper

Download RPC

RouterOS Update (1 Firmware Upgrade Image)

CWMP standard defines that CPE's firmware can be updated using Download RPC with FileType="1 Firmware Upgrade Image" and single URL of a downloadable file (HTTP and HTTPS are supported). Standard also states that downloaded file can be any type and vendor specific process can be applied to finish firmware update. Because MikroTik's update is package based (and also for extra flexibility), an XML file is used to describe firmware upgrade/downgrade. For now, XML configuration supports providing multiple URLs of files, which will be downloaded and applied similarly as regular RouterOS update through firmware/package file upload.

An example of RouterOS bundle package and tr069-client package update (don't forget to also update tr069-client package). An XML file should be put on some HTTP server, which is accessible from CPE for download. Also, downloadable RouterOS package files should be accessible the same way (can be on any HTTP server). Using ACS execute Download RPC with URL pointing to XML file (e.g. "https://example.com/path/upgrade.xml") with contents:

```
<upgrade version="1" type="links">
   <config/>
   <links>
       <link>
          <url>https://example.com/routeros-mipsbe-X.Y.Z.npk</url>
       </link>
       <link>
          <url>https://example.com/tr069-client-X.Y.Z-mipsbe.npk</url>
       </link>
   </links>
</upgrade>
```

CPE will download XML, parse/validate its contents, download files from provided URLs and try to upgrade. The result will be reported with TransferComplete RPC.

> **Note**
>
> Always make firmware updates incremental - first, update locally tested device and make sure that CWMP communication is resumed with a new version and required ROS functionality works. Secondly, repeat steps by updating groups of CPEs incrementally. We do not recommend updating all remote devices at once.
>
> **Warning:** Use HTTPS in production for firmware management

Configuration Change (3 Vendor Configuration File)

The same Download RPC can be used to perform complete configuration overwrite (as intended by standard) OR configuration alteration (when URL's filename extension is ".alter").

Alter configuration

RouterOS has a lot of configuration attributes and not everything can be ported to CWMP Parameters, that's why RouterOS provides a possibility to execute its powerful scripting language to configure any attribute. A configuration alteration (which is really a regular script execution) can be performed using Download RPC FileType="3 Vendor Configuration File" with downloadable file extension ".alter". This powerful feature can be used to configure any ROS attributes which are not available through CWMP Parameters.

Overwrite all configurations

Full ROS configuration overwrite can be performed using Download RPC FileType="3 Vendor Configuration File" with any URL file name (except with ".alter" extension).

Warning: Provided configuration file(script) must be "smart" enough to apply configuration correctly right after reboot. This is especially important when using uploaded configuration file with Upload RPC, because it only contains values export. Some things that should be added manually:

- delay at beginning, for interfaces to show up;
- hidden passwords for users;
- certificates.

RouterOS default configuration change (X MIKROTIK Factory Configuration File)

This vendor specific FileType allows the change of the RouterOS default configuration script that is executed when **/system reset-configuration** command is executed (or the other means when router configuration is being reset).

> **Note**
>
> If the default configuration script is changed it will not be displayed by **/system default-configuration print** as it is the case if that script is altered via Netinstall tool. That command will always show the default script set up by MikroTik
>
> **Warning:** Use this with caution as the failure of uploaded script may render device inoperable and/or inaccessible by the ACS

FactoryReset RPC

This is CWMP standard RPC, which performs RouterOS configuration factory-reset. The reset process is performed in the same way as executing the command:

```
/system reset-configuration skip-backup=yes
```

Note that the default factory configuration can be different for each device (see [1]) and execution of this command removes all configurations and executes internally stored default-configuration script.

Best Practices Guide for preparing CPE with custom factory settings for TR069 https://wiki.mikrotik.com/wiki/Tr069-best-practices

Upload RPC

Upload current configuration (1 Vendor Configuration File)

The result of this is file uploaded to the ACS same as the output of **/export** command in the RouterOS

Upload log file (2 Vendor Log File)

The result of this is file uploaded to the ACS is similar to the output of **/log print** command in the RouterOS

Upload default configuration (X MIKROTIK Factory Configuration File)

The result of this is file uploaded to the ACS that has contents of the current set default configuration script that will be executed if **/system reset-configuration** command is executed. It may differ from one returned using **/system default-configuration print**.

Security

- HTTP should only be used when testing initial setup in the secured/private network because Man-in-the-middle attacker could read/change configuration parameters. **In the production environment, HTTPS is a MUST**.
- CWMP's incoming connection validation by design is safe because CPE will not communicate with any other device except previously configured ACS. Connection Request only signals CPE to start a new connection + new session with previously configured ACS.

Tested ACSs

Ordering is alphabetical. MikroTik does not imply any one vendor superiority of another. If some ACS is missing you can notify us of the existence of it, and it might be added to the list.

Commercial

We have tested and verified to be working the following commercial ACS solutions:

- AVSystem
- Axiros
- Friendly Tech

Open Source

- GenieACS

Note: these ACS systems below seem to be not maintained and thus is not suggested as useful options

- FreeACS
- LibreACS

WebFig

Introduction

WebFig is a web-based RouterOS utility that allows you to monitor, configure and troubleshoot the router. It is designed as an alternative of WinBox, both have similar layouts and both have access to almost any feature of RouterOS.

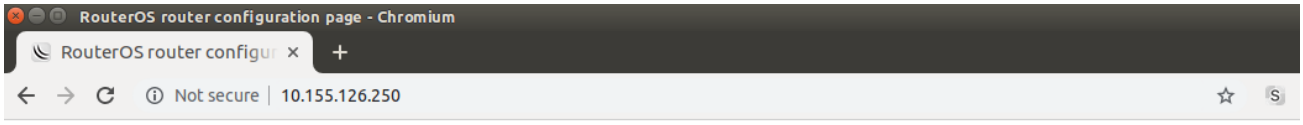
As Webfig is platform-independent, it can be used to configure a router directly from various devices without the need for software developed for specific platforms. In other words, there is no need to install additional software.

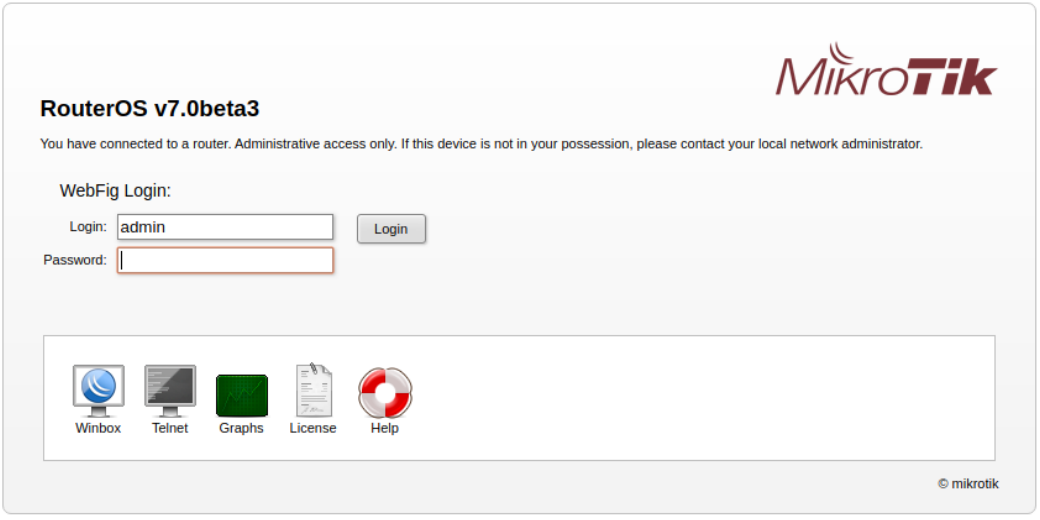
WebFig allows performing three basic actions:

- Configuration - view and edit current configuration;
- Monitoring - display the current status of the router, routing information, interface stats, logs, etc;
- Troubleshooting - RouterOS has built-in many troubleshooting tools (like ping, traceroute, packet sniffers, traffic generators, etc) and all of them can be used with WebFig

Connecting to a Router

As we already know from the First Time Configuration section, the device by default has username **admin** and **no password** configured. Simply open a Web browser and in the search bar type device IP address which by default is **192.168.88.1.** Be sure your device has IP address from the same network, for example, 192.168.88.2 otherwise Layer3 communication will not work.





In our example, we will use IP address 10.155.126.250 to connect to the device via WebFig.

Enable HTTPS

For HTTPS to work properly, you need to specify a valid certificate that WebFig can use. You can use a certificate that is issued by a trusted Certificate Authority (CA) or you can create your own root CA and generate self-signed certificates.

WebFig supports wildcard certificates. You can generate such a certificate by specifying a wildcard in the common-name property, for example, *common-name=*.mikrotik.com*.

To generate your own certificates and enable HTTPS access, you must configure the following:

Create your own root CA on your router and sign it

```
[admin@MikroTik] > certificate add name=local-cert common-name=local-cert key-usage=key-cert-sign,crl-sign
[admin@MikroTik] > certificate sign local-cert
  progress: done
```

In case you already have set up your own CA or you are using a service that signs certificates for you, then you need to create and sign the certificate remotely and import the certificate on the router later. In case you are importing a certificate, then make sure you mark the certificate as trusted.

Create a new certificate for WebFig (non-root certificate)

```
[admin@MikroTik] > certificate add name=webfig common-name=192.168.88.1
[admin@MikroTik] > certificate sign webfig
  progress: done
[admin@MikroTik] > certificate print
Flags: K - private-key; A - authority; T - trusted
Columns:NAME        COMMON-NAME     FINGERPRINT
0  KAT  local-cert  local-cert      9b6363d033c4b2e6893c340675cfb8d1e330977526dba347a440fabffd983c5d
1  KAT  webfig      192.168.88.1    9f84ac2979bea65dccd02652056e5559bcdf866f8da5f924139d99453402bd02
```

Enable **www-ssl** and specify to use the newly created certificate for WebFig

```
[admin@MikroTik] > ip service
set www-ssl certificate=webfig disabled=no
```

You can now visit https://192.168.88.1 and securely configure your router.

By default, browsers will not trust self-signed certificates, you will need to add the certificate as trusted on the first time you visit the page in your browser. Another approach is to export the root CA certificate and import it as a trusted root certificate on your computer, this way all certificates signed by this router will be considered as valid and will make it easier to manage certificates in your network.

Most Internet browsers have their own certificate trust chain and work independently of the operating system's certificate trust chain, this means that you may have to add your own root CA's certificate as a trusted certificate in your browser settings since trusting the certificate in your operating system's settings might not have any effect when using your Internet browser.

Skins

WebFig **Design Skin** is a handy tool to make the interface more user-friendly. It is not a security tool. If the user has sufficient rights it is possible to access hidden features by other means.

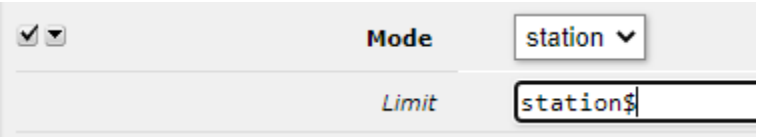
Designing skins

If the user has sufficient permissions (the group has the policy to edit permissions) **Design Skin** button becomes available. Pressing that toggle button will open interface editing options.

To prevent the user from accessing the **Design Skin** menu, disable Policy "policy" under the user group configuration.

Possible operations are:

- Hide menu - this will hide all items from the menu and its submenus;
- Hide submenu - only certain submenu will be hidden;
- Hide tabs - if submenu details have several tabs, it is possible to hide them this way;
- Rename menus and items - make certain features more obvious or translate them into your language;
- Add a note to the item (in detail view) - to add comments on the field;
- Make item read-only (in detail view) - for user safety very sensitive fields can be made read only;
- Hide flags (in detail view) - while it is only possible to hide a flag in detail view, this flag will not be visible in list view and in detailed view;
- Add limits for the field - (in detail view) where it is the list of times that are comma or newline separated list of allowed values:
    - number interval '..' example: 1..10 will allow values from 1 to 10 for fields with numbers, for example, MTU size.
    - field prefix (Text fields, MAC address, set fields, combo-boxes). If it is required to limit prefix length *$* should be added to the end. For example, limiting the wireless interface to "station" only, "Add limit" will contain "station$"



- Add *Tab* - will add a gray ribbon with an editable label that will separate the fields. Ribbon will be added before the field it is added to;
- Add *Separator* - will add a low height horizontal separator before the field it is added to.

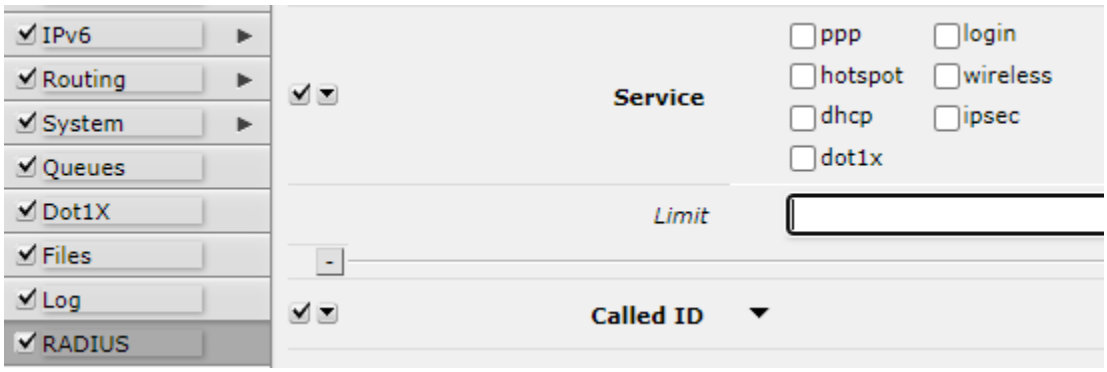
Note: Number interval cannot be set to extend limitations set by RouterOS for that field

Note: Set fields are arguments that consist of a set of check-boxes, for example, setting up policies for user groups, RADIUS "Service"

Note: Limitations set for combo-boxes will also limit the values selectable from the dropdown

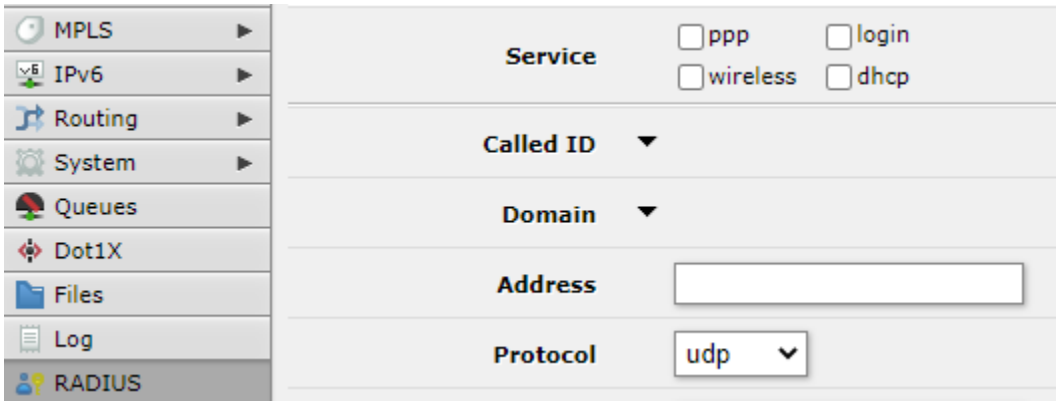
Skin design examples

If you need to limit the user for some services



Add a limit to the RADIUS Service.

The result will be only those services, that are pointed in the "Limit" field.



Using skins

To use skins you have to assign the skin to the group. When that is done, users of that group will automatically use the selected skin as their default when logging into WebFig or WinBox.

```
/user/group/set your_group_name skin=your_skin
```

If it is required to use created skin on another router you can copy files to the skins folder on the other router. On the new router, it is required to add copied skin to the user group to use it.

WinBox

Summary

WinBox is a small utility that allows the administration of MikroTik RouterOS using a fast and simple GUI. It is a native Win32/Win64 binary but can be run on **Linux** and **macOS (OSX)** using Wine. All WinBox interface functions are as close as possible mirroring the console functions, that is why there are no WinBox sections in the manual. Some advanced and system critical configurations are not possible from the WinBox, like MAC address change on an interface.

From WinBox v3.14, the following security features are used:

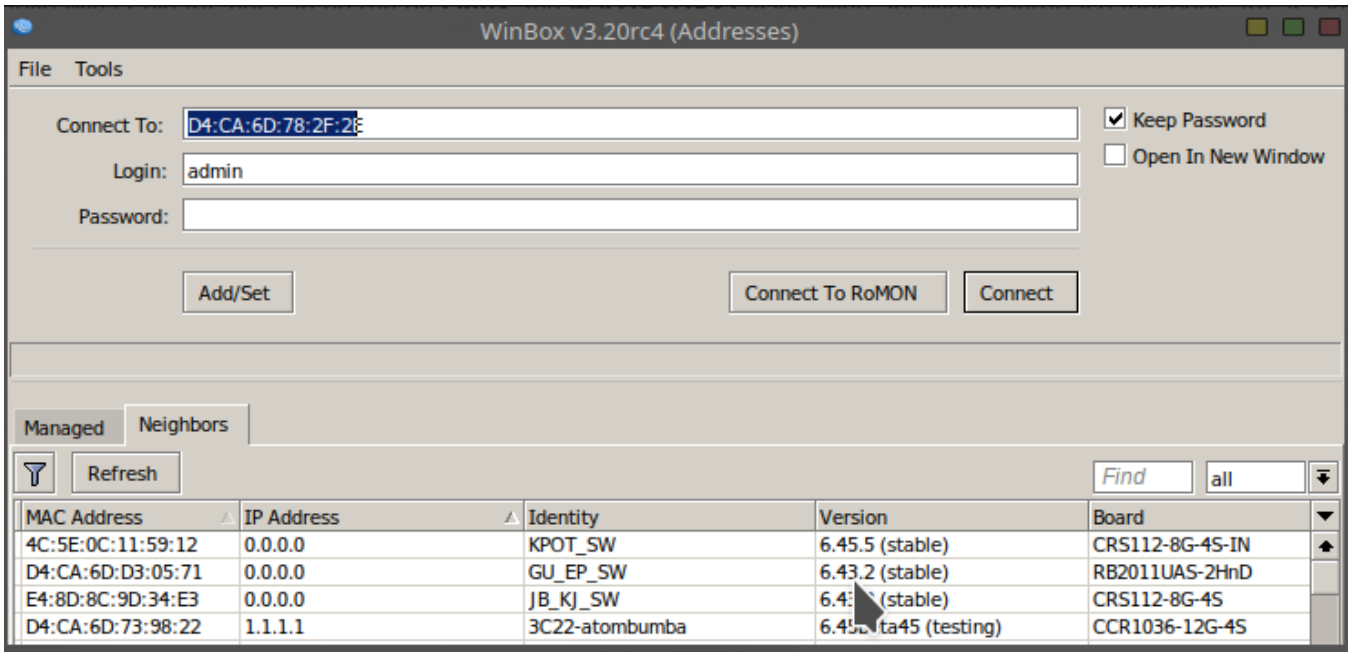
- WinBox.exe is signed with an Extended Validation certificate, issued by SIA Mikrotīkls (MikroTik).
- WinBox uses ECSRP for key exchange and authentication (requires a new WinBox version).
- Both sides verify that the other side knows the password (no man in the middle attack is possible).
- WinBox in RoMON mode requires that the agent is the latest version to be able to connect to the latest version routers.
- WinBox uses AES128-CBC-SHA as an encryption algorithm (requires WinBox version 3.14 or above).

Starting WinBox

WinBox loader can be downloaded from the MikroTik download page. When WinBox.exe is downloaded, double click on it, and the WinBox loader window will pop up. There are two WinBox loader modes: simple which is enabled by default and advanced.

Simple mode

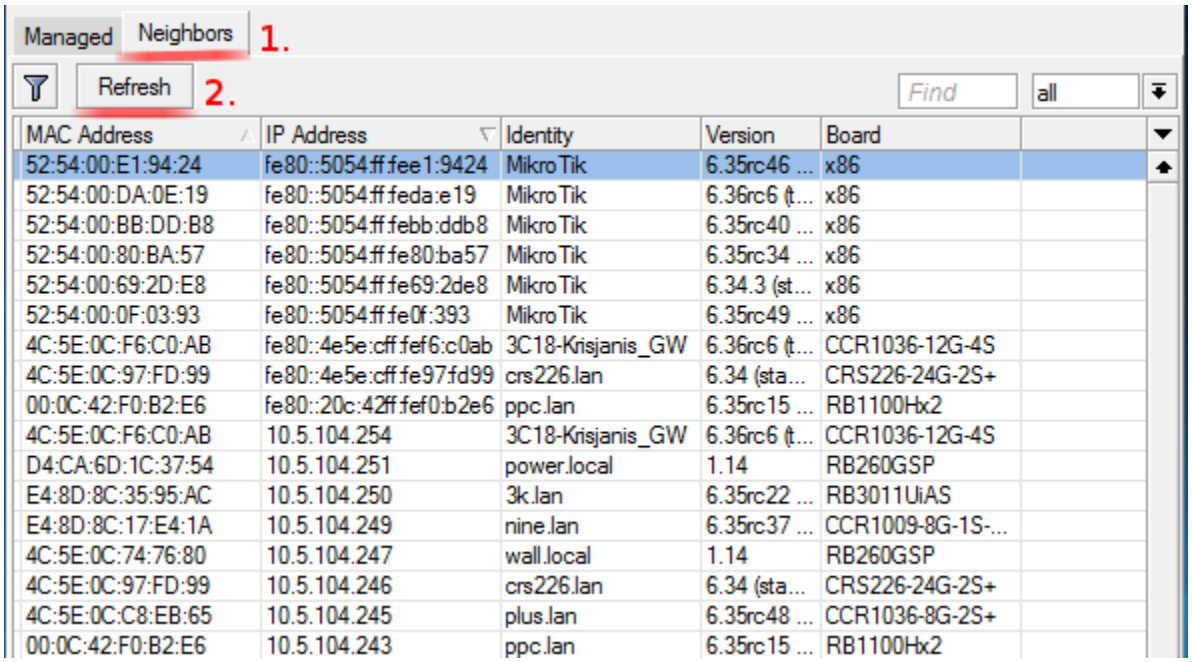
When you open WinBox loader for the first time simple mode layout will be used:



To connect to the router enter the IP or MAC address of the router, specify username and password (if any) and click on the **Connect** button. You can also enter the port number after the IP address, separating them with a colon, like this 192.168.88.1:9999. The port can be changed in the RouterOS **services** menu.

> ⚠ It is recommended to use an IP address whenever possible. MAC session uses network broadcasts and is not 100% reliable.

You can also use neighbor discovery, to list available routers use the **Neighbors** tab:



From the list of discovered routers, you can click on the IP or MAC address column to connect to that router. If you click on IP address then IP will be used to connect, but if you click on MAC Address then the MAC address will be used to connect to the router.

> ⓘ Neighbor discovery will show also devices that are not compatible with WinBox, like Cisco routers or any other device that uses CDP (Cisco Discovery Protocol). If you will try to connect to a SwOS device, then the connection will be established through a web browser

Buttons/check-boxes and Other Fields

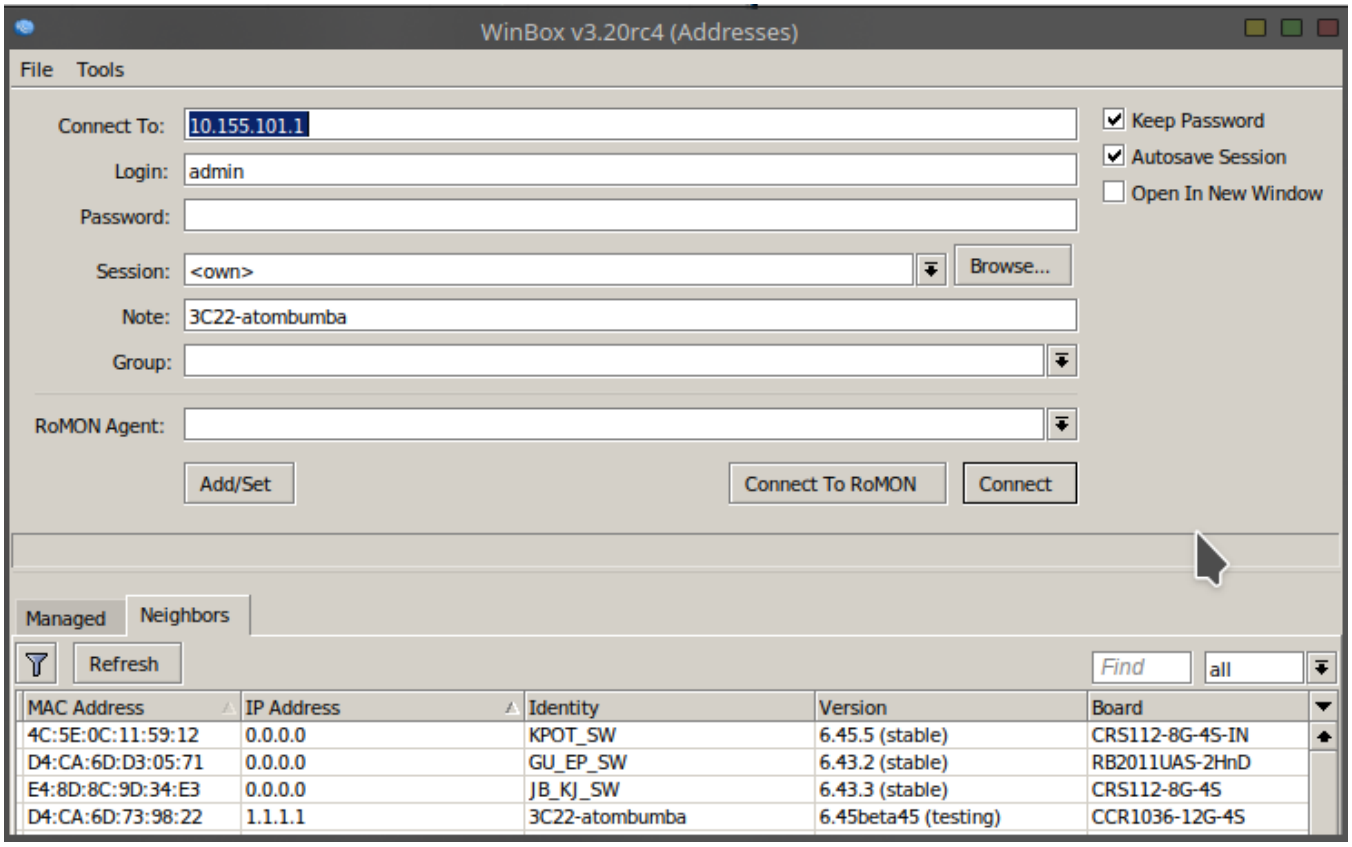
- **Connect** - Connect to the router
- **Connect To RoMON** - Connect to RoMON Agent
- **Add/set** - Save/Edit any of the saved router entries in the **Managed** tab.
- **Open In New Window** - Leaves loader open in the background and opens new windows for each device to which connection is made.
- **Connect To:** - destination IP or MAC address of the router
- **Login** - username used for authentication
- **Password** - password used for authentication
- **Keep Password** - if unchecked, the password is not saved to the list

Menu Items

- **File**
  - **New** - Create a new managed router list in a specified location
  - **Open** - Open managed router list file
  - **Save As** - Save current managed router list to file
  - **Exit** - Exit WinBox loader
- **Tools**
  - **Advanced Mode** - Enables/Disables advanced mode view
  - **Import** - Imports saved session file
  - **Export** - Exports saved session file
  - **Move Session Folder** - Change path where session files are stored
  - **Clear cache** - Clear WinBox cache
  - **Check For Updates** - Check for updates for WinBox loader

Advanced mode

Additional WinBox loader parameters are revealed when an **advanced mode** is enabled with *Tools → Advanced Mode:*



Buttons/check-boxes and Other Fields

Buttons/check-boxes

- **Browse** - Browse file directory for some specific session
- **Keep Password** - if unchecked, the password is not saved to the list
- **Secure mode** - if checked, WinBox will use DH-1984 for key exchange and modified and hardened RC4-drop3072 encryption to secure the session.
- **Autosave session** - Saves sessions automatically for devices to which connections are made.

Fields:

- **Session** - Saved router session.
- **Note** - Note that is assigned to save router entry.
- **Group** - Group to which saved router entry is assigned.
- **RoMON Agent** - Select RoMON Agent from the available device list

Managed routers list is encrypted, but it can still be loaded with another WinBox, **IF** the master password is not set for it!

Command Line

It is possible to use the command line to pass connect to, user and password parameters automatically:

```
winbox.exe [<connect-to> [<login> [<password>]]]
```

For example (with no password):

```
winbox.exe 10.5.101.1 admin ""
```

Will connect to router 10.5.101.1 with user "admin" without a password.

It is possible to use the command line to pass connect to, user, and password parameters automatically to connect to the router through RoMON. In this case, RoMON Agent must be saved on the Managed routers list so WinBox would know the user and password for this device:

```
winbox.exe --romon [<romon-agent> [<connect-to> [<login> [<password>]]]]
```

For example (with no password):

```
winbox.exe --romon 10.5.101.1 D4:CA:6D:E1:B5:7D admin ""
```

Will connect to router D4:CA:6D:E1:B5:7D, through 10.5.101.1 via RoMON Agent with user "admin" without a password.

IPv6 connectivity

WinBox supports IPv6 connectivity. To connect to the router's IPv6 address, it must be placed in square braces the same as in web browsers when connecting to the IPv6 server. Example:

db8::1

when connecting to the link-local address interface index must be entered after the %:

:a00:27ff:fe70:e88c%2

Port number is set after the square brace when it is necessary to connect WinBox to other port than the default:

:a00:27ff:fe70:e88c%2:8299

WinBox neighbor discovery is capable of discovering IPv6 enabled routers. There are two entries for each IPv6 enabled router, one entry is with IPv4 address and another one with IPv6 link-local address. You can easily choose which one you want to connect to.

Run WinBox on macOS

Starting with macOS 10.15 Catalina, Apple has removed support for 32bit applications, meaning it is no longer possible to use regular Wine and regular WinBox in this OS. Wine has made available a 64bit version for macOS, and MikroTik has released a special WinBox64.exe version as well.

To run WinBox64 the following steps are required.

1. Install latest Wine from the Wine macOS builds page (wine-devel-7.X-osx64.tar.xz) and make sure you have downloaded the WinBox64.exe executable from the MikroTik download page.
2. Launch WinBox64.exe with "open file with" > Wine64.app

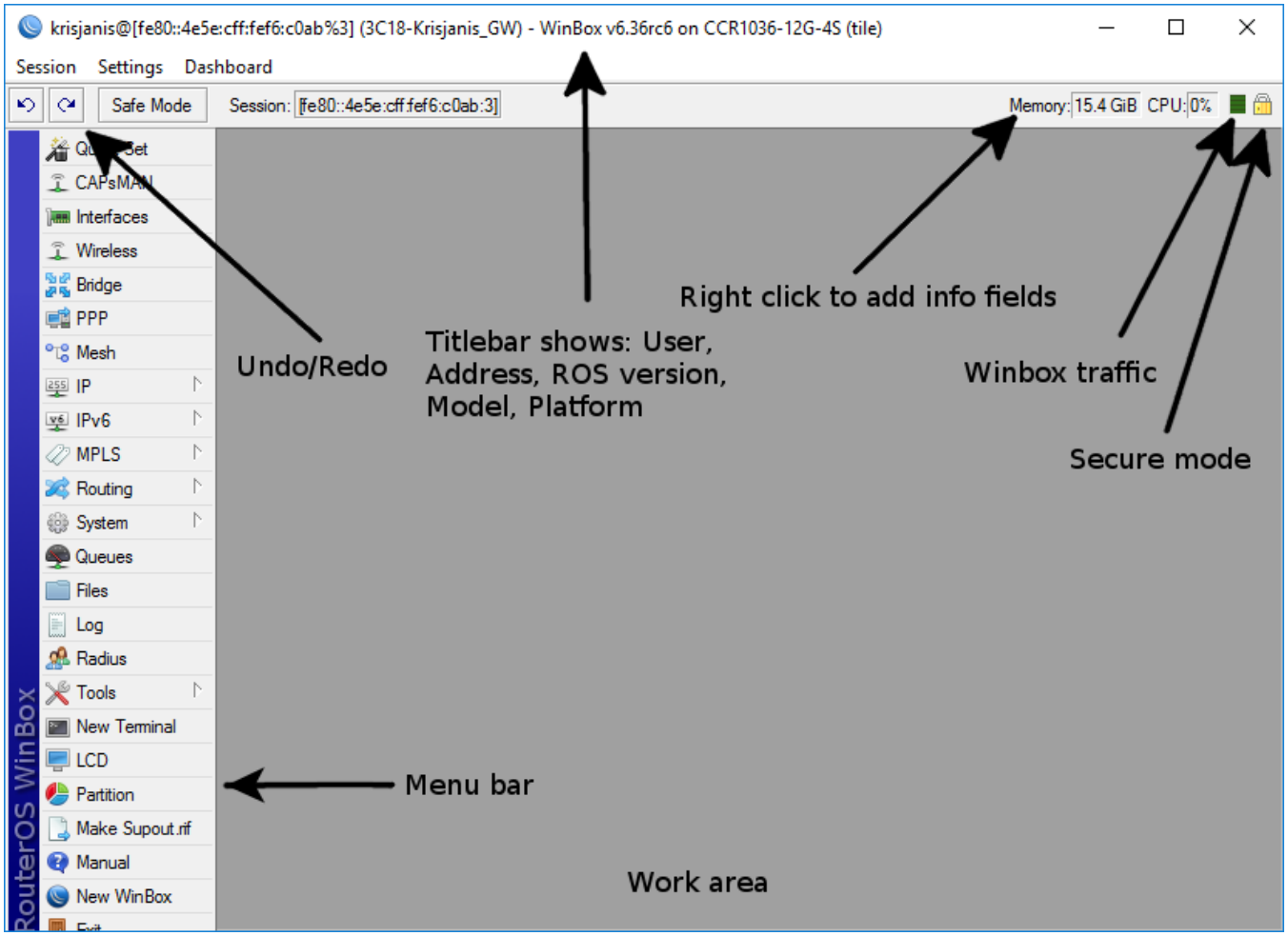
Run WinBox on Linux

It is possible to run WinBox on Linux by using Wine emulation software. Make sure that the Microsoft font pack is installed, otherwise, you may see distortions.

Interface Overview

WinBox interface has been designed to be intuitive for most of the users. The interface consists of:

- The main toolbar at the top where users can add various info fields, like CPU and memory usage.
- The menu bar on the left - list of all available menus and sub-menus. This list changes depending on what packages are installed. For example, if the IPv6 package is disabled, then the **IPv6** menu and all its sub-menus will not be displayed.
- Work area - an area where all menu windows are opened.



The title bar shows information to identify with which router WinBox session is opened. Information is displayed in the following format:

```
[username]@[Router's IP or MAC] ( [RouterID] ) - WinBox [ROS version] on [RB model] ([platform])
```

From screenshot above we can see that user **krisjanis** is logged into router with IPv4/IPv6 address **[fe80::4e5e:cff:fef6:c0ab%3]**. Router's ID is **3C18-Krisjanis_GW**, currently installed RouterOS version is **v6.36rc6**, RouterBoard is **CCR1036-12G-4S** and platform is **tile**.

On the Main toolbar's left side is located:

- **undo**
- **redo**

- **Safe Mode**
- Currently loaded session

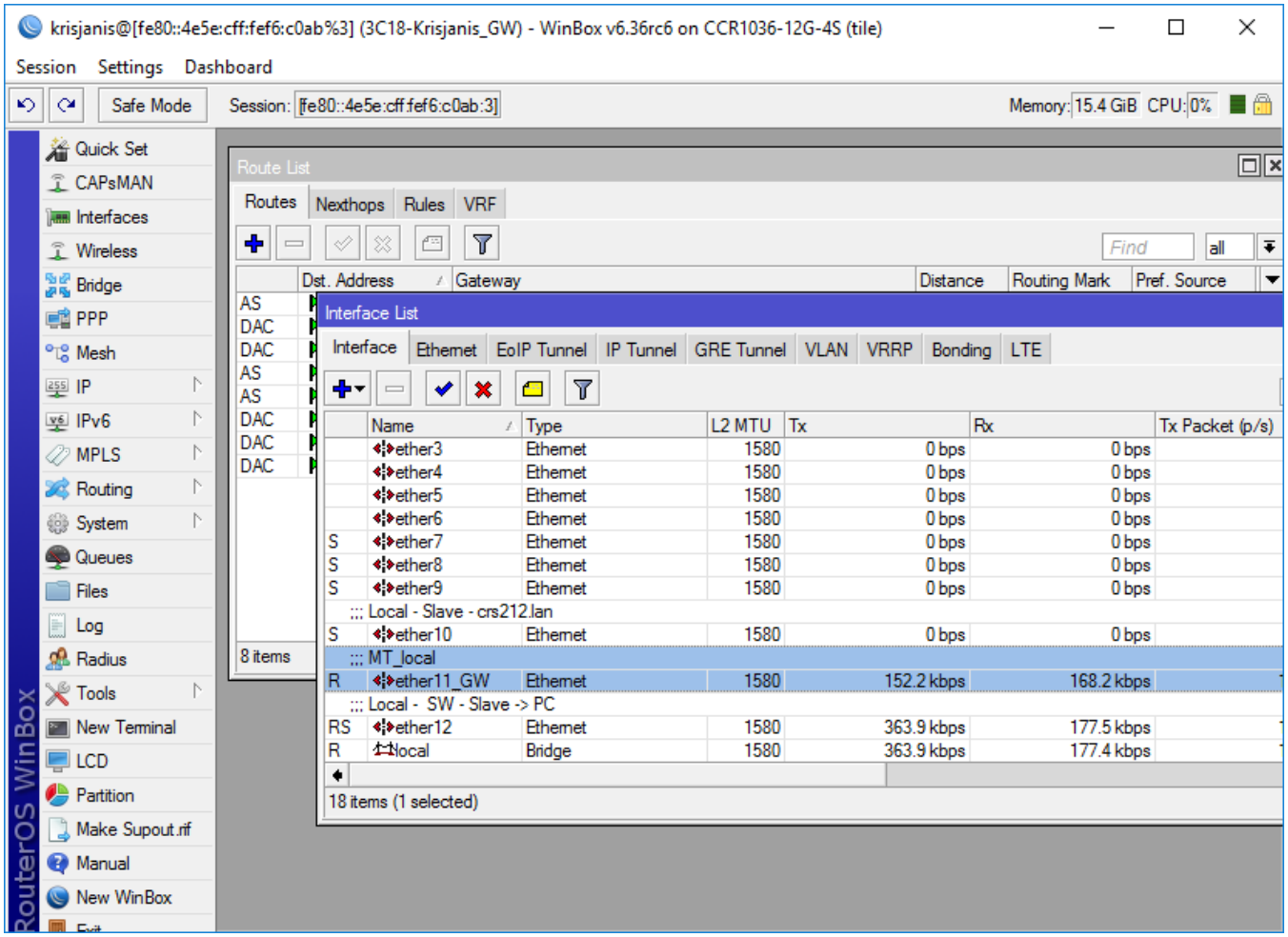
More about Safe mode and undoing performed actions read in this article.

On the right side is located:

- an indicator that shows whether the WinBox session uses encryption
- WinBox traffic indicator displayed as a green bar,
- Custom info fields that can be added by the user by right-clicking on the toolbar and picking available info fields from the list

Work Area and Child Windows

WinBox has an MDI interface meaning that all menu configuration (child) widows are attached to the main (parent) WinBox window and is showed in the work area.



Child windows can not be dragged out of the working area. Notice in the screenshot above that the **Interface** window is dragged out of the visible working area and a horizontal scroll bar appeared at the bottom. If any window is outside visible work area boundaries the vertical or/and horizontal scrollbars will appear.

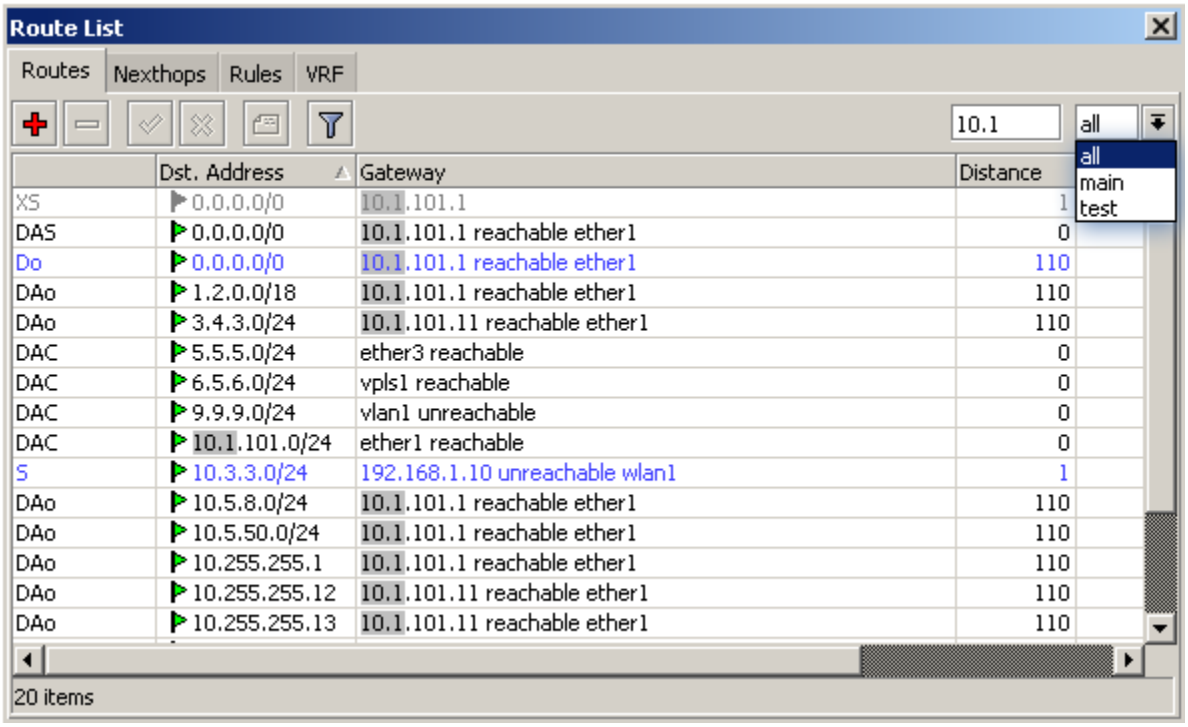
Child window menu bar

Each child window has its own toolbar. Most of the windows have the same set of toolbar buttons:

-  **Add** - add a new item to the list

- **Remove** - remove the selected item from the list
- **Enable** - enable selected item (the same as **enable** command from console)
- **Disable** - disable selected item (the same as **disable** command from console)
- **Comment** - add or edit a comment
- **Sort** - allows to sort out items depending on various parameters. Read more >>

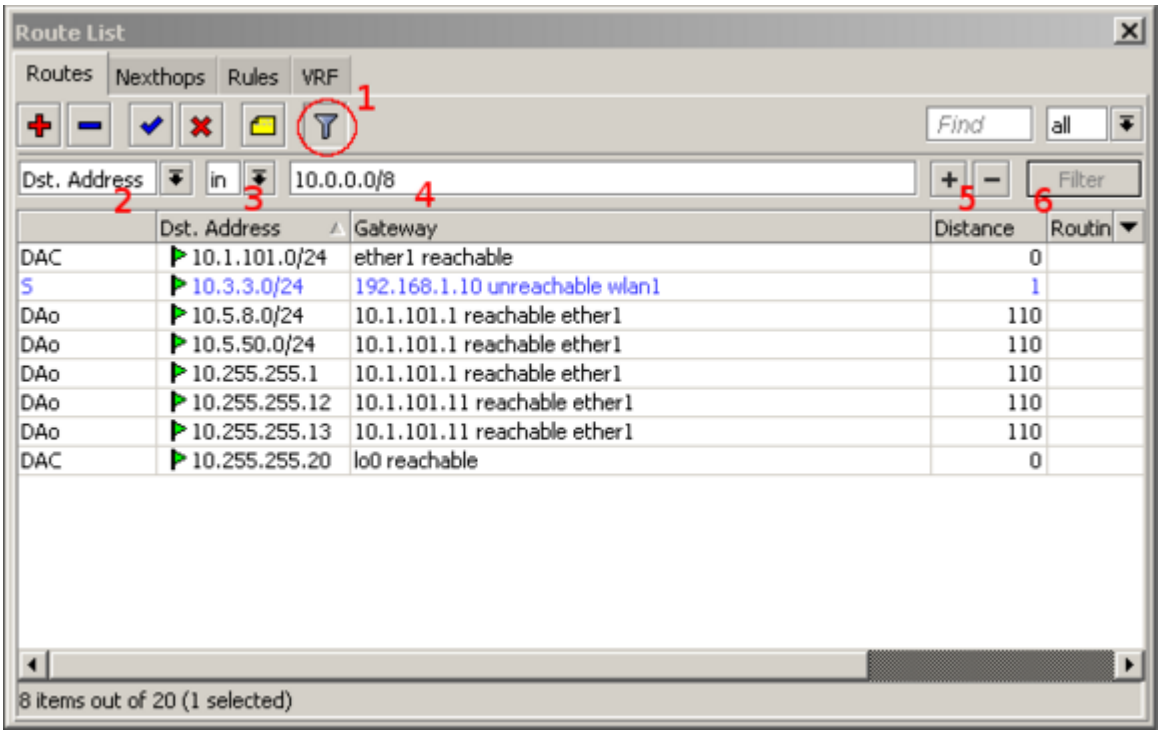
Almost all windows have a quick search input field on the right side of the toolbar. Any text entered in this field is searched through all the items and highlighted as illustrated in the screenshot below



Notice that on the right side next to the quick find input filed there is a drop-down box. For the currently opened (IP Route) window, this drop-down box allows to quickly sort out items by routing tables. For example, if the **main** is selected, then only routes from the main routing table will be listed.
A similar drop-down box is also in all firewall windows to quickly sort out rules by chains.

Sorting out displayed items

Almost every window has a **Sort** button. When clicking on this button several options appear as illustrated in the screenshot below



The example shows how to quickly filter out routes that are in the 10.0.0.0/8 range

1. Press **Sort** button
2. Choose **Dst.Address** from the first drop-down box.
3. Choose **in** form the second drop-down box. "in" means that filter will check if DST address value is in range of the specified network.
4. Enter the network against which values will be compared (in our example enter "10.0.0.0/8")
5. These buttons are to add or remove another filter to the stack.
6. Press the **Filter** button to apply our filter.

As you can see from the screenshot WinBox sorted out only routes that are within the 10.0.0.0/8 range.

Comparison operators (Number **3** in the screenshot) may be different for each window. For example "IP Route" window has only two **is** and **in**. Other windows may have operators such as "is not", "contains", "contains not".

WinBox allows building a stack of filters. For example, if there is a need to filter by destination address and gateway, then

- set the first filter as described in the example above,
- press **[+]** button to add another filter bar in the stack.
- set up a second filter to filter by the gateway
- press the **Filter** button to apply filters.

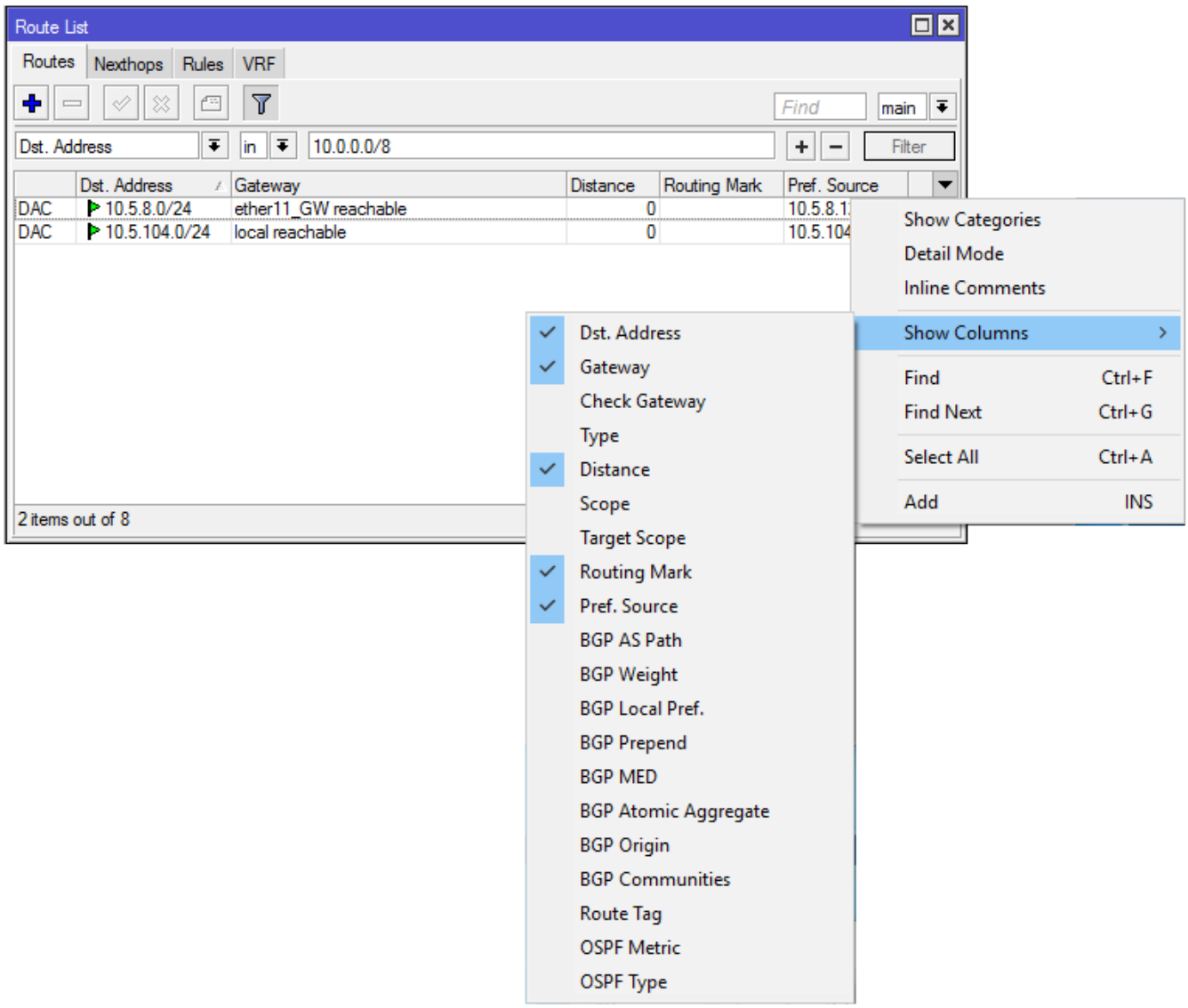
You can also remove unnecessary filters from the stack by pressing the **[-]** button.

Customizing list of displayed columns

By default, WinBox shows the most commonly used parameters. However sometimes it is needed to see other parameters, for example, "BGP AS Path" or other BGP attributes to monitor if routes are selected properly.

WinBox allows to customize displayed columns for each individual window. For example to add BGP AS path column:

- Click on the little arrow button (**1**) on the right side of the column titles or right mouse click on the route list.
- From popped up menu move to **Show Columns** (**2**) and from the sub-menu pick the desired column, in our case click on **BGP AS Path** (**3**)

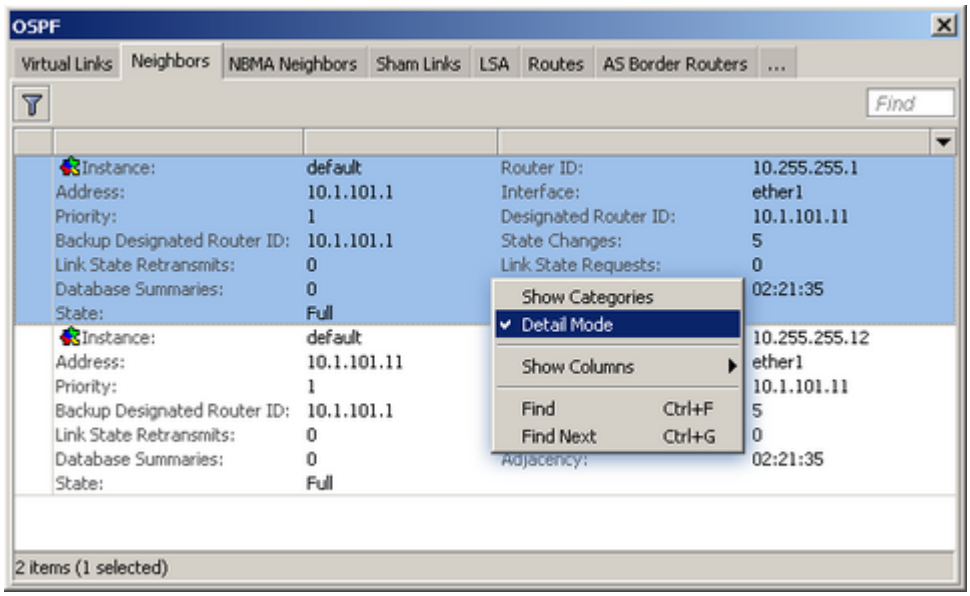


Changes made to window layout are saved and next time when WinBox is opened the same column order and size are applied.

Detail mode

It is also possible to enable **Detail mode**. In this mode all parameters are displayed in columns, the first column is the parameter name, the second column is the parameter's value.

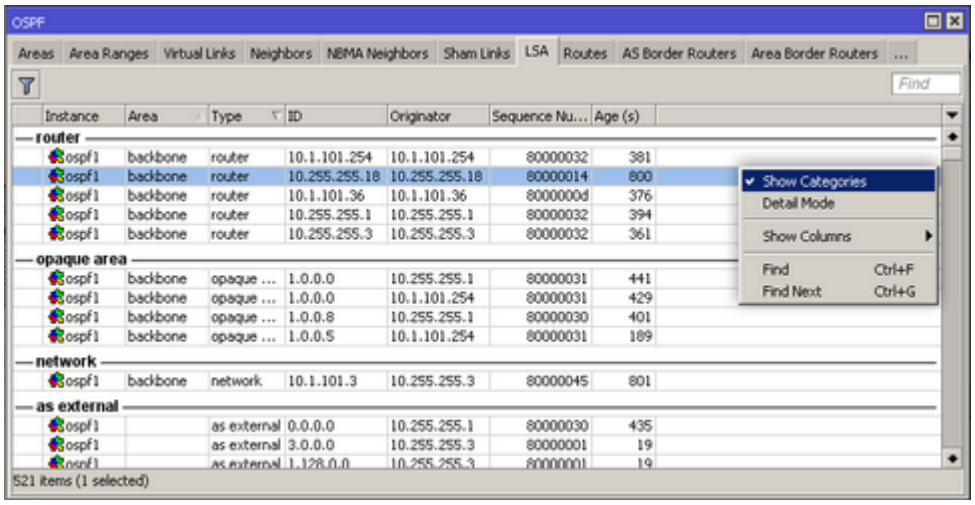
To enable detail mode right mouse click on the item list and from the popup menu pick **Detail mode**



Category view

It is possible to list items by categories. In this mode, all items will be grouped alphabetically or by another category. For example, items may be categorized alphabetically if sorted by name, items can also be categorized by type like in the screenshot below.

To enable Category view, right mouse click on the item list and from the popup menu pick **Show Categories**



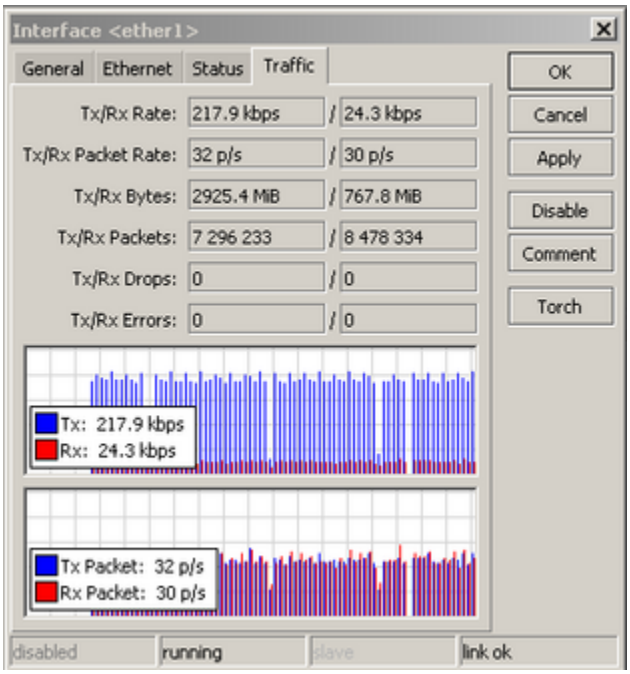
Drag & Drop

It is possible to upload and download files to/from the router using WinBox drag & drop functionality. You can also download the file by pressing the right mouse button on it and selecting "Download".

> Drag & Drop works if WinBox is running on Linux using wine4. Drag and drop between two WinBox windows may fail.

Traffic monitoring

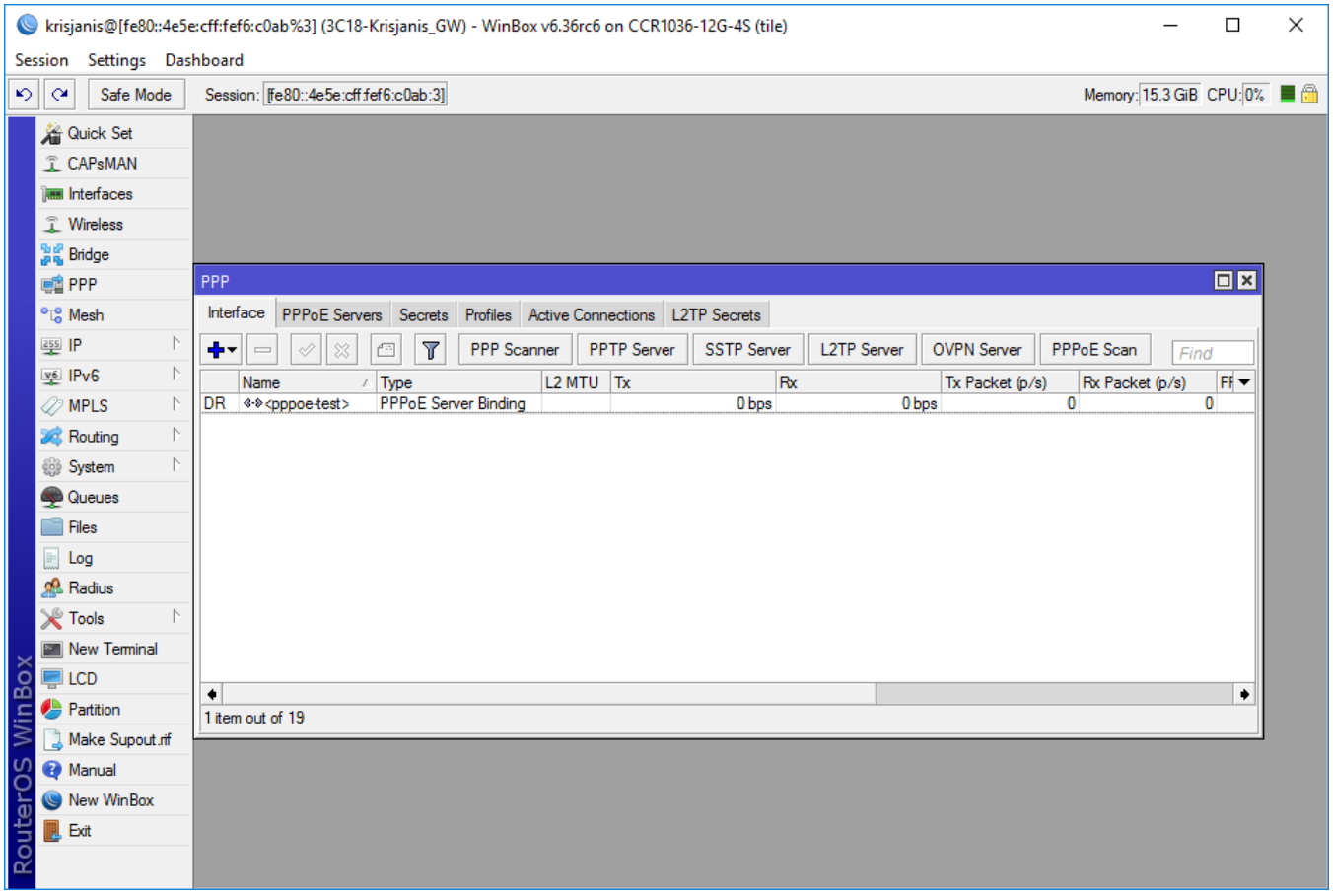
WinBox can be used as a tool to monitor the traffic of every interface, queue, or firewall rule in real-time. The screenshot below shows Ethernet traffic monitoring graphs.



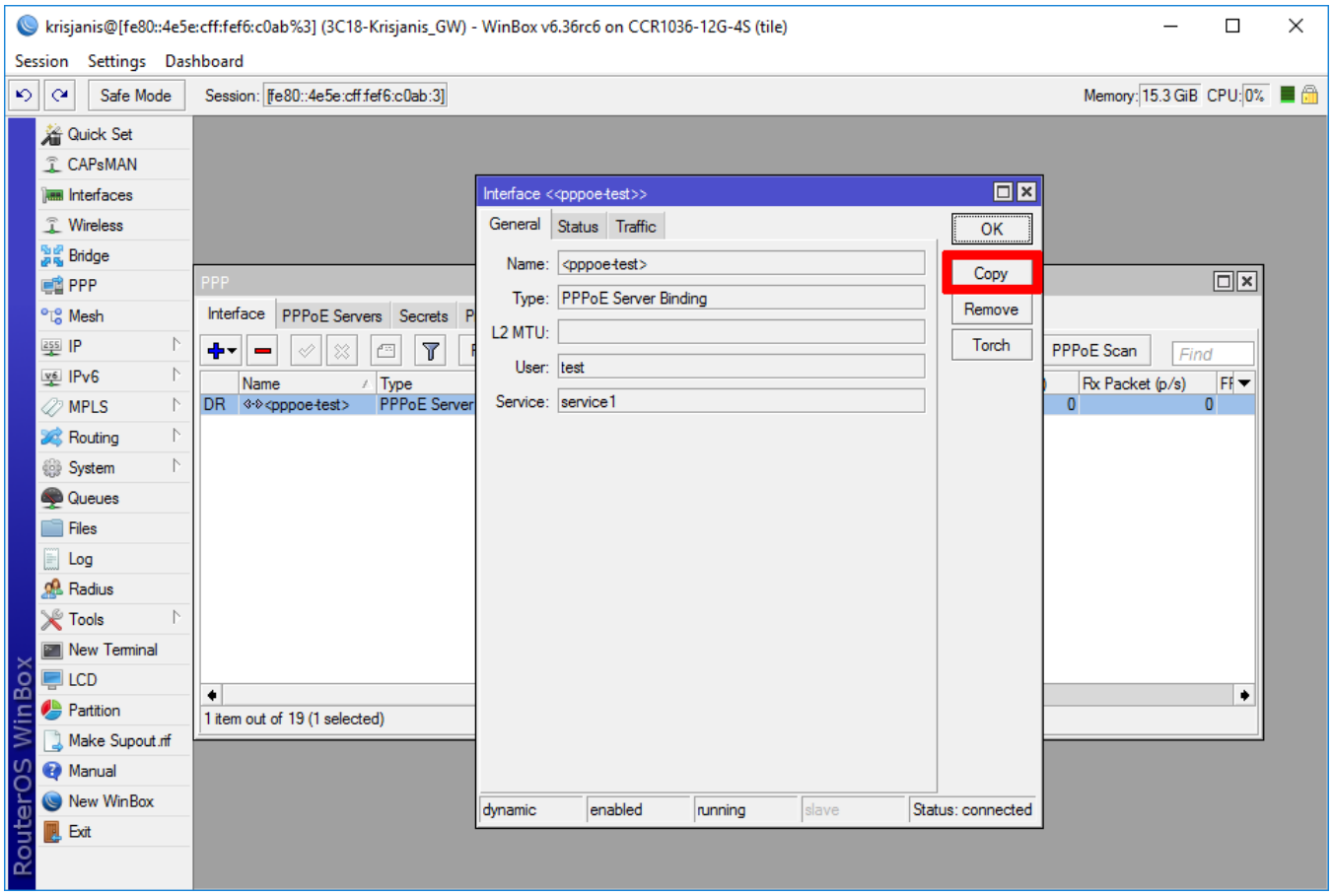
Item copy

This shows how easy it is to copy an item in WinBox. In this example, we will use the COPY button to make a Dynamic PPPoE server interface into a Static interface.

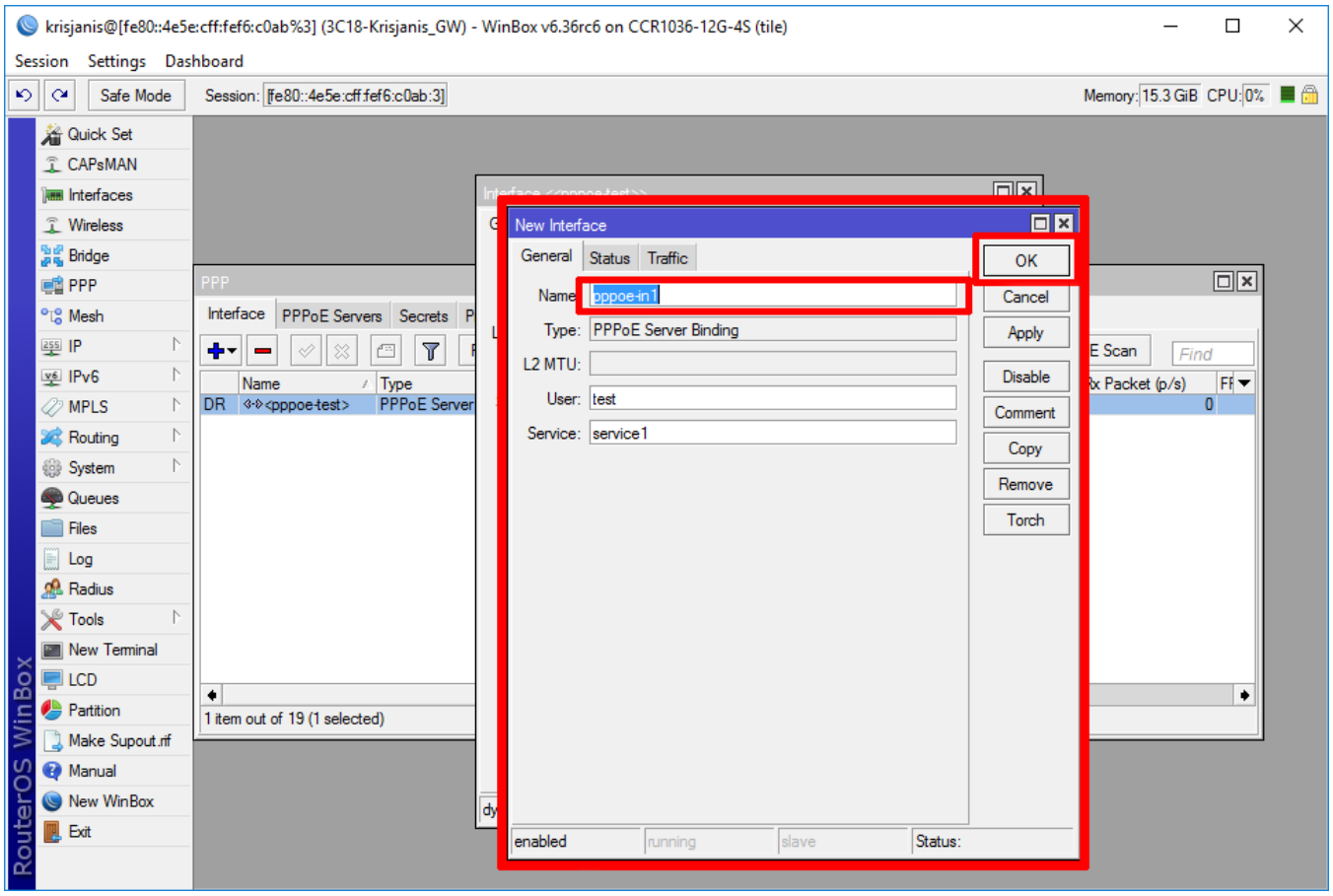
This image shows us the initial state, as you see DR indicates "D" which means Dynamic:



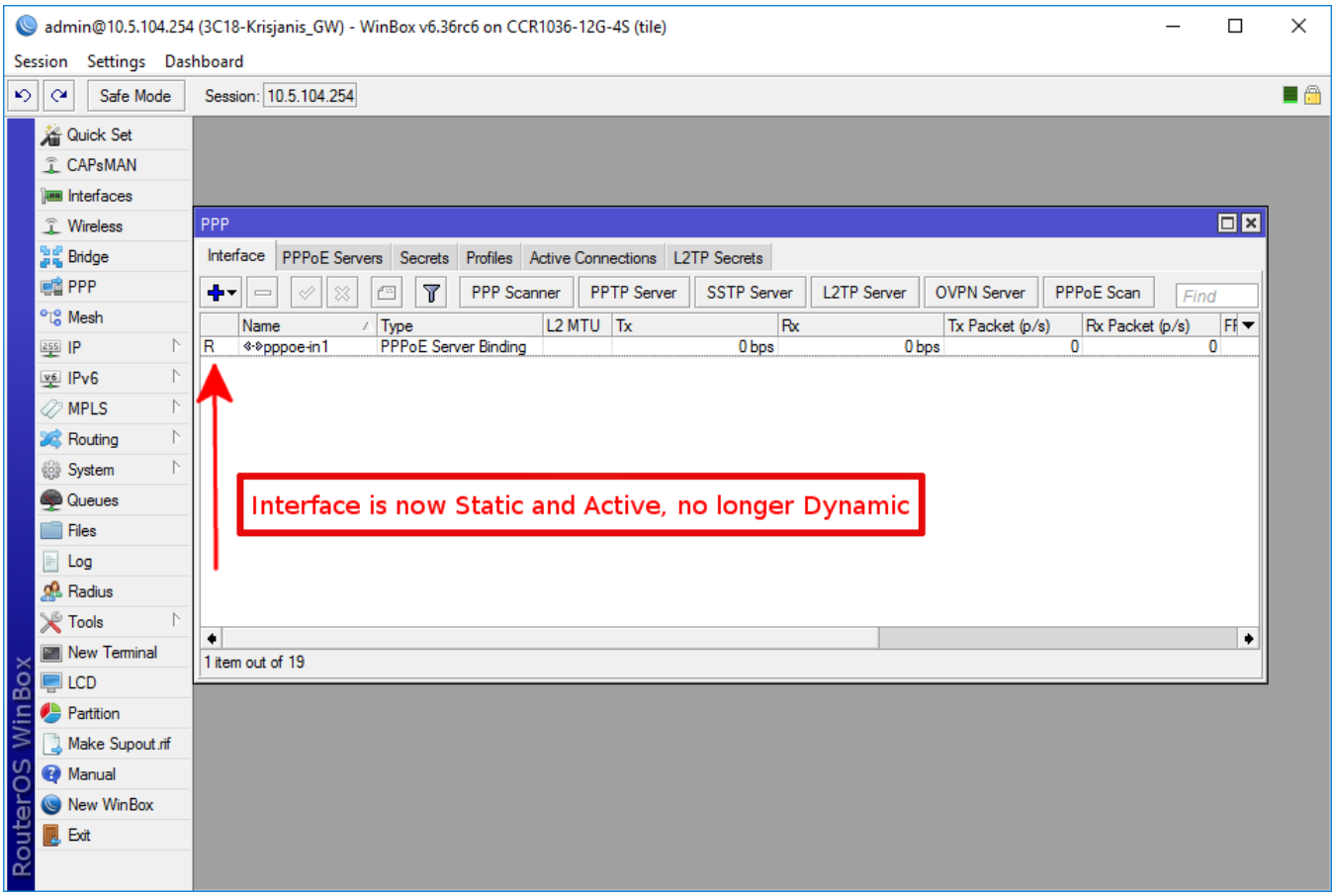
Double-Click on the interface and click on COPY:



A new interface window will appear, a new name will be created automatically (in this case pppoe-in1)



After this Down/Up event this interface will be Static:



Transferring Settings

- Managed router transfer - In the File menu, use Save As and Open functions to save the managed router list to file and open it up again on a new workstation.
- Router sessions transfer - In the Tools menu, use Export and Import functions to save existing sessions to file and import them again on a new workstation.

Troubleshooting

WinBox cannot connect to the router's IP address, devices do not show up in the Neighbors list

Make sure that the Windows firewall is set to allow WinBox connections through Private and/or Public network interfaces in the Windows firewall, it can be changed in *Control Panel\System and Security\Windows Defender Firewall\Allowed applications* or disable the Windows firewall.

I get an error '(port 20561) timed out' when connecting to routers mac address

Windows (7/8) does not allow mac connection if file and print sharing is disabled.

I can't find my device in WinBox IPv4 Neighbors list or MAC connection fails with "ERROR could not connect to XX-XX-XX-XX-XX-XX"

Most of the network drivers will not enable IP stack unless your host device has an IP configuration. Set IPv4 configuration on your host device.

Sometimes the device will be discovered due to caching, but MAC connection will still fail with "ERROR: could not connect to XX:XX:XX:XX:XX:XX

> ⚠ WinBox MAC-ADDRESS connection requires MTU value set to 1500, unfragmented. Other values can perform poorly - loss of connectivity can occur.

FlashFig

Description

FlashFig is an application for mass router configuration. It can be used by MikroTik distributors, ISPs, or any other companies who need to apply RouterOS configuration to many routers in the shortest possible time.

FlashFig applies MikroTik RouterOS **configuration** to any RouterBOARD within **3 seconds**. You can perform FlashFig on a batch of routers, the only thing you need is to **connect** RouterBOARD to a Layer 2 network running FlashFig and to **power** a FlashFig-enabled RouterBOARD up.

FlashFig only runs on a Windows computer and is available from the downloads page.

All RouterBOARDs support FlashFig mode. It works between a Windows computer running FlashFig and a RouterBOARD in the same broadcast domain (direct Layer 2 Ethernet network connection is required).

FlashFig support is enabled on every new RouterBOARD manufactured since March 2010 by default from the factory. For older models, FlashFig can be enabled via RouterBOOT or from MikroTik RouterOS console - */system routerboard settings set boot-device=flash-boot-once-then-nand* or */system routerboard settings set boot-device=flash-boot*.

After FlashFig is used once on a brand new RouterBOARD, it is disabled on further boots to avoid unwanted reconfiguration at a later time. To use FlashFig a second time on the same router, you need to enable **flash-boot** in Bootloader settings (this setting will revert to NAND after a successful configuration change OR once any user logs into the board).

If RouterOS *reset-configuration* command is used later (or configuration reset using the Reset button), FlashFig configuration is loaded. To permanently overwrite, use the Netinstall process and check *Apply default configuration* or use *-r* flag in Linux-based command line.

You view FlashFig video tutorial on MikroTik YouTube channel.

FlashFig Example

This is a step-by-step example of how to use the FlashFig process to apply a chosen MikroTik RouterOS configuration to a 'factory fresh' RouterBOARD.

Requirements

The Windows computer must be equipped with the following ports and contain the following files:

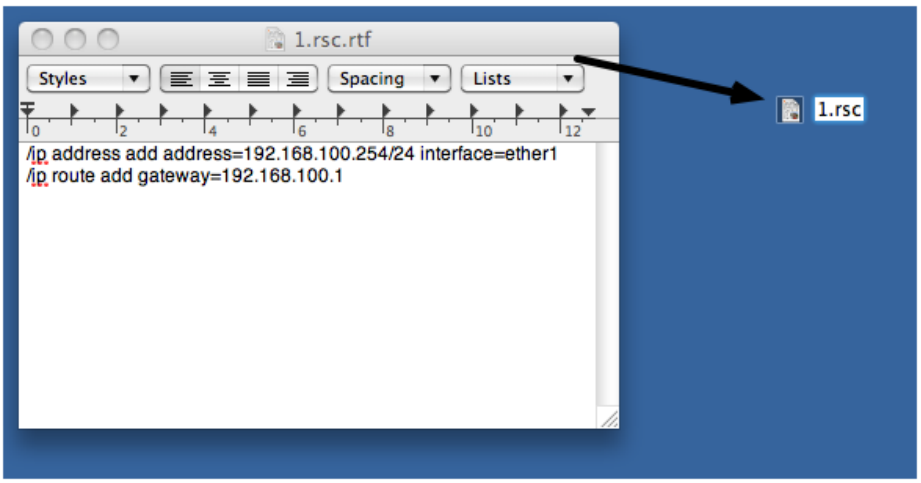
- A working Ethernet port;
- Valid .rsc file(s) with MikroTik RouterOS configuration similar to an export/import file. In addition to regular configuration commands, it is also possible to re-apply the factory passwords by using the read-only variables *$defconfPassword* and *$defconfWifiPassword* (starting from RouterOS 7.10beta8);
- Always use the latest FlashFig program available from the downloads page;
- The RouterBOARD has to be in flash-boot mode, if this is the very first boot, nothing needs to be done

> Be aware of the text editor's treatment of CR/LF characters and test that the config has no errors when normally applied onto an identical version of RouterOS before applying via FlashFig as run-time errors will not be visible!

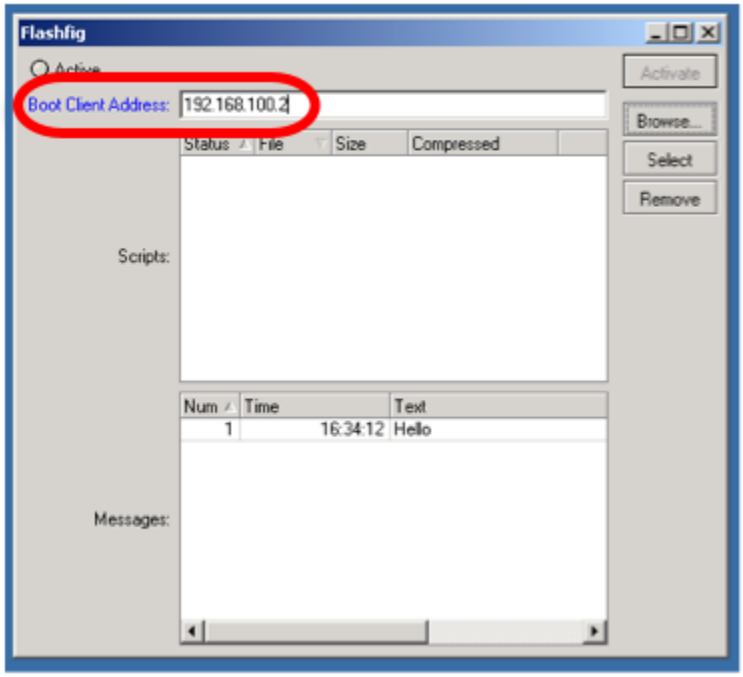
Pre-Configuration

Windows Computer

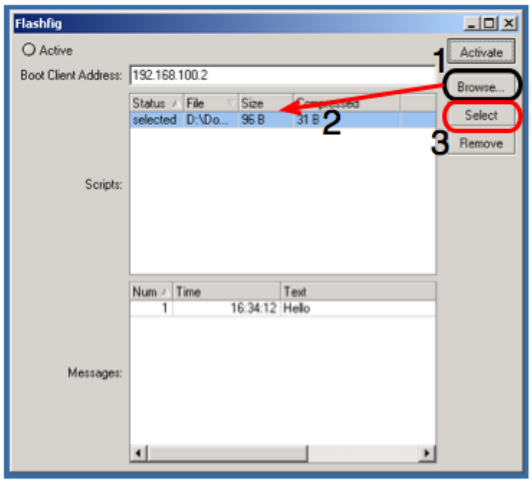
- Run FlashFig;
- Prepare **.rsc** file, **.rsc** file is regular/import file, it accepts valid MikroTik RouterOS CLI commands. You can create .rsc file with any text editor program (Notepad, Notepad++, Texteditor, TextEdit, Microsoft Word, OpenOffice Writer)



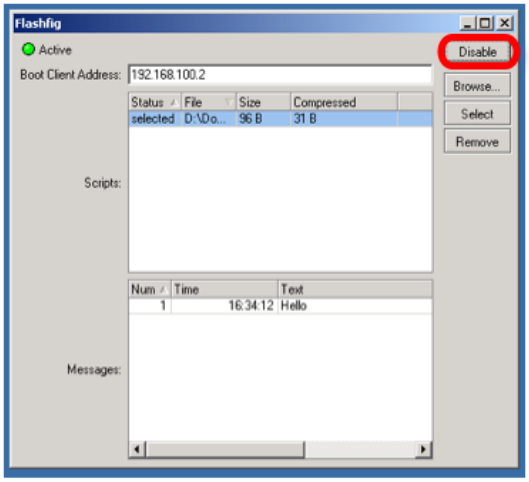
- Assign **Boot Client Address**, which should be an address within *the same subnet as that configured on the computer's Ethernet interface*,



- **Browse** for **.rsc** MikroTik RouterOS configuration file to apply to the RouterBOARD, highlight the file and **Select** to approve it,



- Activate FlashFig server, now it is ready to FlashFig. Note, any RouterBOARD will be FlashFig'ed within the network when they are powered on with boot-device configured to **flash-boot** or **flash-boot-once-then-nand**,



RouterBOARD

- FlashFig mode is enabled on every RouterBOARD from the factory by default, which means **no configuration** is required on RouterBOARD.
- If FlashFig is not enabled on your router, access the RouterBOARD with WinBox/Console and change the boot-device to *flash-boot* or *flash-boot-once-then-nand*:

```
system/routerboard/settings/set boot-device=flash-boot
```

Or use a more preferable option, for a single boot flash-boot:

```
system/routerboard/settings/set boot-device=flash-boot-once-then-nand
```

Your router is now ready for FlashFig.

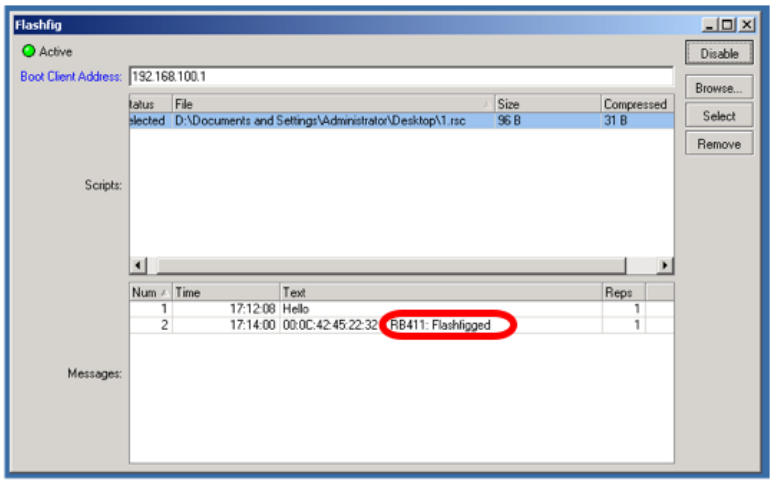
Connect

Connect the ***Boot*** port of RouterBOARD and FlashFig computer to the same Local Area Network.

Run FlashFig

- Plug-in power for RouterBOARD

- Check the status on FlashFig program,



Messages log shows "FlashFigged" and RouterBOARD should repeatedly make the morse code sound for the character "/" ("_.._." and flash the LED - it is now safe to unplug / power down the router.

- FlashFig **configuration** was applied to the RouterBOARD and it is **ready** to be used in production with this new config.

Troubleshoot

FlashFig can not find a router

If between a PC and a router there is another device (a router/switch), ensure that for this device:

- DHCP server is disabled;
- if used ports are in a bridge, set bridge *protocol-mode* to *none*;
- HW-offload for used ports is disabled.

FlashFig finds a router, flashing is not done (no TFTP request)

Ensure that the computer on which FlashFig is running has only one network interface active.

FlashFig is done, but a configuration is not applied

If all procedures went successfully, but RouterOS configuration from .rsc file is not applied, add startup delay to *.rsc configuration file. The reason might be, that the configuration script is executed before all interfaces boots up.

Not enough flash space, ignoring

FlashFig configuration maximum file size is up to 4000 bytes, otherwise program will return an error as above.

Authentication, Authorization, Accounting

In This Section:

Certificates

Overview


```
/certificate
```

The general menu is used to manage certificates, add templates, issue certificates, and manage CRL and SCEP Clients.

Certificate Template

Certificate templates are used to prepare a desired certificate for signing.

Certificate template is deleted right after a certificate is signed or a certificate request command is executed

```
/certificate
add name=CA-Template common-name=CAtemp key-usage=key-cert-sign,crl-sign
add name=Server common-name=server
add name=Client common-name=client
```

To print out certificates:

```
[admin@4k11] /certificate> print detail
Flags: K - private-key; L - crl; C - smart-card-key; A - authority; I - issued, R - revoked; E - expired; T -
trusted
 0         name="CA-Template" key-type=rsa common-name="CAtemp" key-size=2048 subject-alt-name="" days-
valid=365 key-usage=key-cert-sign,crl-sign

 1         name="Server" key-type=rsa common-name="server" key-size=2048 subject-alt-name="" days-valid=365
           key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-
client

 2         name="Client" key-type=rsa common-name="client" key-size=2048 subject-alt-name="" days-valid=365
           key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-
client
```

Certificate properties

Property	Description
common-name (*string*)	Certificate common name
copy-from (*name*)	Certificate name from which to copy general settings

country (*string*)	Certificate issuer country
days-valid (days Default: 365)	Days certificate will be valid after signing
digest-algorithm (*md5 | sha1 | sha256 | sha384 | sha512* Default: **sha256**)	Certificate public key algorithm
key-size (1024 | 1536 | 2048 | 4096 | 8192 | prime256v1 | secp384r1 | secp521r1 Default: **2048**)	Certificate public key size
key-usage (code-sign | crl-sign | decipher-only | dvcs | encipher-only key-cert-sign | ocsp-sign | tls-client | content-commitment | data-encipherment | digital-signature | email-protect | key-agreement | key-encipherment | timestamp | tls-server **Default: digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-client)**	Certificate usage
locality (*string*)	Certificate issuer locality
name (*string*)	Certificate name
organization (*string*)	Certificate issuer organization
state (*string*)	Certificate issuer state
subject-alt-name (*DNS: | IP: | email:*)	Certificate subject alternative name
trusted (*no| yes Default:*)	
unit (*string*)	Certificate issuer organizational unit

Certificate read-only properties

After a certificate is signed, most of a certificate template properties are converted to read-only (except *name* and *trusted*)

Property	Description
serial-number	Certificate serial number
fingerprint	
akid	Certificate authority ID
skid	Certificate subject ID
invalid-before	Date and time before which a certificate expired
invalid-after	Date and time after which a certificate expired
expires-after	
key-type	
ca	Certificate authority common name

⚠ If the CA certificate is removed, all issued certificates in the chain are also removed.

Sign Certificate

Certificates should be signed. In the following example, we will sign certificates and add CRL URL for the server certificate:

```
/certificate
sign CA-Template
sign Client
sign Server ca-crl-host=192.168.88.1 name=ServerCA
```

Let`s check is the certificates are signed:

```
[admin@MikroTik] /certificate> print
Flags: K - private-key; L - crl; A - authority; T - trusted
Columns: NAME, COMMON-name, FINGERPRINT
#        NAME         COMMON  FINGERPRINT
0  K AT  CA-Template  CAtemp  0c7aaa7607a4dde1bbf33deaae6be7bac9fe4064ba47d64e8a73dcefad6cfc38
1  K AT  Client       client  b3ff25ecb166ea41e15733a7493003f3ea66310c10390c33e98fe32364c3659f
2  KLAT  ServerCA     server  152b88c9d81f4b765a59e2302e01efd1fbf11ceeed6e59f4974e87787a5bb980
```


The time of the key signing process depends on the key size of a specific certificate. With values of 4k and higher, it might take a substantial time to sign this specific certificate on less powerful CPU-based devices.

Export Certificate

It is possible to export client certificates with keys and CA certificates:

```
/certificate
export-certificate CA-Template
export-certificate ServerCA export-passphrase=yourpassphrase
export-certificate Client export-passphrase=yourpassphrase
```

Exported certificates are available under the */file* section:

```
[admin@MikroTik] > file print
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME                          TYPE        SIZE  CREATION-TIME
0  skins                         directory         jan/19/2019 00:00:04
1  flash                         directory         jan/19/2019 01:00:00
2  flash/rw                      directory         jan/19/2019 01:00:00
3  flash/rw/disk                 directory         jan/19/2019 01:00:00
4  pub                           directory         jan/19/2019 02:42:16
5  cert_export_CA-Template.crt   .crt file   1119  jan/19/2019 04:15:21
6  cert_export_ServerCA.crt      .crt file   1229  jan/19/2019 04:15:42
7  cert_export_ServerCA.key      .key file   1858  jan/19/2019 04:15:42
8  cert_export_Client.crt        .crt file   1164  jan/19/2019 04:15:55
9  cert_export_Client.key        .key file   1858  jan/19/2019 04:15:55
```


Exporting certificates requires "sensitive" user policy.

Import Certificate

To import certificates, certificates must uploaded to a device using one of the file upload methods.

Certificates must be imported as a file.

Property	Description
name (*string* Default: file-name_number)	A certificate name that will be shown in the certificate manager
file-name (*string*)	A file name that will be imported

passphrase (*string* Default: none)	File passphrase if there is such

```
[admin@MikroTik] > /certificate/import file-name=certificate_file_name name=name_example
passphrase=file_passphrase
     certificates-imported: 2
     private-keys-imported: 1
            files-imported: 1
       decryption-failures: 0
  keys-with-no-certificate: 0

[admin@MikroTik] > /certificate/print
Flags: K - PRIVATE-KEY; T - TRUSTED
Columns: NAME, COMMON-NAME
#    NAME            COMMON-NAME
0 KT name_example    cert
1  T name_example_1  ca
```

Let's Encrypt certificates

Watch our video about this feature.

RouterOS v7 has Let's Encrypt (letsencrypt) certificate support for the 'www-ssl' service. To enable the Let's Encrypt certificate service with automatic certificate renewal, use the 'enable-ssl-certificate' command:

```
/certificate enable-ssl-certificate dns-name=my.domain.com
```

Note that the DNS name must point to the router and port TCP/80 must be available from the WAN. If the dns-name is not specified, it will default to the automatically generated */ip cloud* name (ie. http://example.sn.mynetname.net)

Different acme servers

Support has been added starting from 7.15beta7, you can use not only Let's Encrypt certificate service, but any other you like.

Server properties

Property	Description
directory-url (*string*)	ACME directory url.
eab-hmac-key (*string*)	HMAC key for ACME External Account Binding (optional).
eab-kid (*string*)	Key identifier (optional).

Example:

```
/certificate/enable-ssl-certificate directory-url=https://acme.zerossl.com/v2/DV90 dns-name=mydomain.abc eab-
hmac-key=4ac7xuxAdV4mIncwIIEhLjExsFZ4v1rWgDkX4SKXD25pMVtF85GZJYSF8UKXUOjzSr2g3-v4lhL57NHFaQ42Ff eab-
kid=GHWaP2_Ghx73vcU8ricAKU
```

SCEP

SCEP is using HTTP protocol and base64 encoded GET requests. Most of the requests are without authentication and cipher, however, important ones can be protected if necessary (ciphered or signed using a received public key).

SCEP client in RouterOS will:

- get CA certificate from CA server or RA (if used);
- user should compare the fingerprint of the CA certificate or if it comes from the right server;
- generate a self-signed certificate with a temporary key;
- send a certificate request to the server;
- if the server responds with status x, then the client keeps requesting until the server sends an error or approval.

The SCEP server supports the issuance of one certificate only. RouterOS supports also renew and next-ca options:

- renew - the possibility to renew the old certificate automatically with the same CA.
- next-ca - possibility to change the current CA certificate to the new one.

The client polls the server for any changes, if the server advertises that the next-ca is available, then the client may request the next CA or wait until CA almost expires and then request the next-ca.

The RouterOS client by default will try to use POST, AES, and SHA256 if the server advertises that. If the above algorithms are not supported, then the client will try to use 3DES, DES and SHA1, MD5.

SCEP certificates are renewed when 3/4 of their validity time has passed.

Dot1X

Summary

Dot1X is implementation of IEEE 802.1X standard in RouterOS. Main purpose is to provide port-based network access control using EAP over LAN also known as EAPOL. 802.1X consists of a supplicant (client), an authenticator (server) and an authentication server (RADIUS server). Both authenticator and supplicant sides are supported in RouterOS, as well as authentication server when User Manager package is installed. Supported EAP methods for supplicant are EAP-TLS, EAP-TTLS, EAP-MSCHAPv2 and PEAPv0/EAP-MSCHAPv2.

Feature is not supported on SMIPS devices (hAP lite, hAP lite TC and hAP mini).

Client

Supplicant configuration settings.

Sub-menu: `/interface dot1x client`

Property	Description
anon-identity (*string*; Default:)	Identity for outer layer EAP authentication. Used only with `eap-ttls` and `eap-peap` methods. If not set, value from `identity` parameter will be used for outer layer EAP authentication.
client-certificate (*string*; Default:)	Name of a certificate listed in System/Certificates. Necessary when `eap-tls` method is used.
comment (*string*; Default:)	Short description of the entry.
disabled (*yes* | *no*; Default: **no**)	Whether client is enabled or not.
eap-methods (*eap-tls* | *eap-ttls* | *eap-peap* | *eap-mschapv2*; Default:)	Ordered list of EAP methods used for authentication.
identity (*string*; Default:)	Supplicant identity used for EAP authentication.
interface (*string*; Default:)	Name of the interface the client will run on.
password (*string*; Default:)	Cleartext password for supplicant.

Read only properties

Property	Description

status (*authenticated* / *authenticating* / *disabled*)	Possible statuses: • authenticated - the client has successfully authenticated; • authenticated without server - access to the port is granted without communication with server; • authenticating - the server is reached and authentication process is ongoing; • connecting - initial stage of the authentication process; • disabled - the client is disabled; • error - an internal error has occurred; • interface is down - the parent interface is not running; • rejected - the server denied the authentication.

Server

A RouterOS dot1x server acts as an authenticator. An interface where dot1x server is enabled will block all traffic except for EAPOL packets which is used for the authentication. After client is successfully authenticated, the interface will accept all received traffic on the port. If the interface is connected to a shared medium with multiple hosts, the traffic will be accepted from all hosts when at least one client is successfully authenticated. However, it is possible to configure dynamic switch rules to accept only the authenticated user source MAC address and drop all other source MAC addresses. In case of failed authentication, it is possible to accept the traffic with a dedicated port VLAN ID.

⚠ When a dot1x server is created on a bridge port, the bridge should be running (R/M)STP, otherwise EAP packets from the client will not be correctly accepted. Bridge interface is created with `protocol-mode=rstp` by default. If the bridge port should not send any BPDUs or any received BPDUs should be ignored, use `edge=yes` configuration on bridge ports.

Sub-menu: `/interface dot1x server`

Property	Description
accounting (*yes* / *no*; Default: **yes**)	Whether to send RADIUS accounting requests to authentication server.
auth-timeout (*time*; Default: **1m**)	Total time available for EAP authentication.
auth-types (*dot1x* / *mac-auth*; Default: **dot1x**)	Used authentication type on a server interface. When both options are selected at the same time, the server will prefer `dot1x` authentication type and only after 3 `retrans-timeout` periods, the authentication type will fall back to `mac-auth`. In order for `mac-auth` authentication type to work, the server interface should receive at least one frame containing a client's device source MAC address.
comment (*string*; Default:)	Short description of the entry.
disabled (*yes* / *no*; Default: **no**)	Whether server config is enabled or not.
guest-vlan-id (*integer: 1..4094*; Default: **!guest-vlan-id**)	Assigned VLAN when end devices does not support `dot1x` authentication and no `mac-auth` fall back is configured. The setting will apply after 3 `retrans-timeout` periods. Once dot1x enabled client is created and successful re-authentication happened, the port is removed from the guest VLAN. This setting is available since RouterOS 7.2 version and has an effect when bridge `vlan-filtering` is enabled. By default, guest VLAN is disabled.
interface (*string*; Default:)	Name of the interface or interface list the server will run on.
interim-update (*time*; Default: **0s**)	Interval between scheduled RADIUS Interim-Update messages.
mac-auth-mode (*mac-as-username* / *mac-as-username-and-password*; Default: **mac-as-username**)	Allows to control User-Name and User-Password RADIUS attributes when using MAC authentication.
radius-mac-format (*XX-XX-XX-XX-XX-XX* / *XX:XX:XX:XX:XX:XX* / *XXXXXXXXXXXX* / *xx-xx-xx-xx-xx-xx* / *xx:xx:xx:xx:xx:xx* / *xxxxxxxxxxxx*; Default: **XX:XX:XX:XX:XX:XX**)	Controls how the MAC address of the client is encoded in the User-Name and User-Password attributes when using MAC authentication.

reauth-timeout (*time*; Default: **!reauth-timeout**)	Enables server port re-authentication. When enabled with `dot1x` authentication type, server will try to re-authenticate a client by sending EAP-Request Identity to the client. When enabled with `mac-auth` authentication type, server will try to re-authenticate client with RADIUS server by using the last seen MAC address. This setting is available since RouterOS 7.2 version. By default, re-authentication is disabled.
reject-vlan-id (i*nteger: 1..4094*; Default: **!reject-vlan-id**)	Assigned VLAN when authentication failed and a RADIUS server responded with an Access-Reject message. This property will not apply if the RADIUS server is not responding at all, the client authentication will simply timeout and the service will be unavailable. This property only has an effect when bridge `vlan-filtering` is enabled. By default, reject VLAN is disabled.
retrans-timeout (*time*; Default: **30s**)	Time interval between message re-transmissions if no response is received from supplicant.
server-fail-vlan-id (i*nteger: 1..4094*; Default: **!server-fail-vlan-id**)	Assigned VLAN when RADIUS server is not responding and request timeout has elapsed. This setting is available since RouterOS 7.2 version and has an effect when bridge `vlan-filtering` is enabled. By default, server-fail VLAN is disabled.

Currently authenticated clients are listed in the active menu (read only properties).

Sub-menu: `/interface dot1x server active`

Property	Description
auth-info (*string*)	Authentication information: • dot1x • dot1x (guest vlan) • dot1x (reject vlan) • dot1x (server fail vlan) • mac-auth • mac-auth (reject vlan) • mac-auth (server fail vlan)
client-mac (*mac-address*)	MAC Address of the supplicant.
interface (*string*)	Name of the interface.
session-id (*string*)	Unique session identifier.
username (*string*)	Identity of the supplicant.
vlan-id (*string*)	Untagged VLAN ID that is assigned to the interface. VLAN ID filtering must be enabled on bridge.

Statuses of all active dot1x server interfaces are listed in the state menu (read only properties).

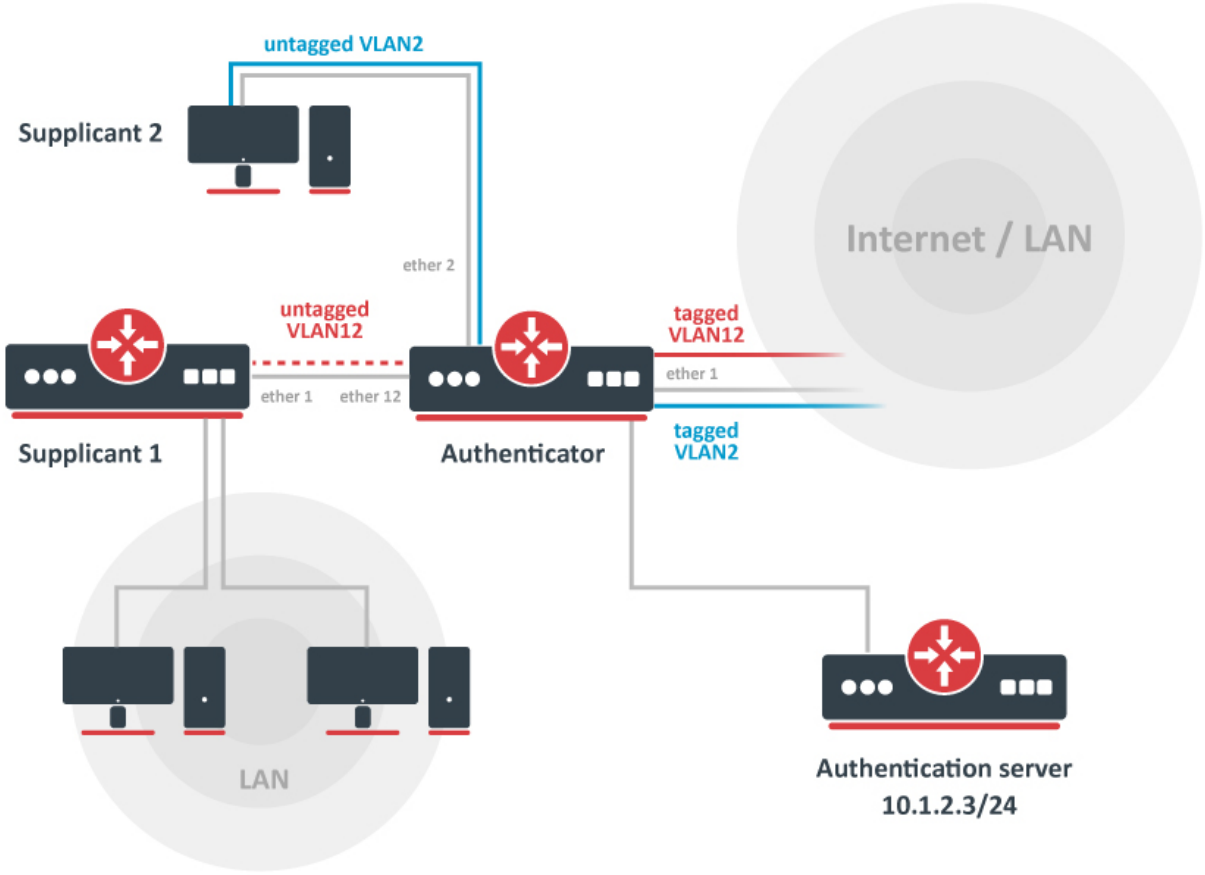
Sub-menu: `/interface dot1x server state`

Property	Description
interface (*string*)	Name of the interface.
status (*string*)	Possible interface statuses: • authorized - access to interface is granted; • iface-down - interface is not running; • rejected-holding - access was rejected by the RADIUS server; • un-authorized - access to interface is not granted.

Examples

Below are described the most common configuration examples for dot1x server and client.

RouterOS Authenticator configuration



Start off by adding a new RADIUS client. The authentication server (RADIUS) does not necessary have to be in the same LAN as authenticator, but it must be reachable from the authenticator, so any firewall limitations must be considered.

```
/radius
add address=10.1.2.3 secret=radiussecret service=dot1x
```

⚠ If RADIUS communication is done over public network, it is advised to use RadSec for RADIUS communication. More information: RADIUS

Add new dot1x server instances.

```
/interface dot1x server
add interface=ether2 interim-update=30s comment=accounted
add interface=ether12 accounting=no comment=notaccounted
```

Port based VLAN ID assignment

It is possible to assign an authenticated interface to a specific VLAN ID using bridge VLAN filtering. This can be done using RADIUS Tunnel-Type, Tunnel-Medium-Type and Tunnel-Private-Group-ID attributes. Note that only devices with hardware offloaded VLAN filtering will be able to do this in switch chip.

First of all, make sure the interface is added to a bridge which has VLAN filtering enabled.

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether12
```

It is necessary to add static VLAN configuration for tagged VLAN traffic to be sent over ether1 interface.

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=2
add bridge=bridge1 tagged=ether1 vlan-ids=12
```

With enabled RADIUS debug logs it is possible to see complete RADIUS message packets with all attributes. In our example, Tunnel attributes are received in Access-Accept message from RADIUS server:

```
09:51:45 radius,debug,packet received Access-Accept with id 64 from 10.1.2.3:1812
09:51:45 radius,debug,packet     Tunnel-Type = 13
09:51:45 radius,debug,packet     Tunnel-Medium-Type = 6
09:51:45 radius,debug,packet     Tunnel-Private-Group-ID = "12"
(..)
09:51:45 radius,debug,packet     User-Name = "dot1x-user"
```

The VLAN ID is now present in active session list and untagged ports are added to previously created static VLAN configuration.

```
/interface dot1x server active print
 0 interface=ether12 username="dot1x-user" user-mac=00:0C:42:EB:71:F6 session-id="86b00006" vlan=12
```

```
/interface bridge vlan print detail
Flags: X - disabled, D - dynamic
 0 D bridge=bridge1 vlan-ids=1 tagged="" untagged="" current-tagged="" current-untagged=bridge1,ether3

 1   bridge=bridge1 vlan-ids=2 tagged=ether1 untagged="" current-tagged=ether1 current-untagged=ether2

 2   bridge=bridge1 vlan-ids=12 tagged=ether1 untagged="" current-tagged=ether1 current-untagged=ether12
```

Dynamic switch rule configuration

In some network configurations, additional access rules are needed for a particular supplicant to restrict or allow certain network services. This can be done using a Mikrotik-Switching-Filter attribute, please see the RADIUS vendor dictionary. When a client is successfully authenticated by an authentication server, the server can pass back the Mikrotik-Switching-Filter attribute. Based on the received information, the authenticator will create dynamic access rules on a switch port where the client resides. These rules will be active as long as the client session is active and the interface is running. There are certain order and restrictions regarding correct switch rule implementation:

- The `mac-protocol`, `src-mac-address` (available only since RouterOS 7.2 version), `src-address` (IPv4/mask, available only since RouterOS 7.2 version), `dst-address` (IPv4/mask), `protocol` (IPv4) `src-port` (L4, available only since RouterOS 7.2 version), `dst-port` (L4) conditional parameters are supported
- Hexadecimal or decimal representation can be used for `mac-protocol` and `protocol` parameters (e.g. `protocol 17` or `protocol 0x11`)
- The `src-port` and `dst-port` support single or range values (e.g. `src-port 10` or `src-port 10-20`)
- The `src-mac-address` support "xx:xx:xx:xx:xx:xx" or "xxxxxxxxxxxx" formats, and switch rule without any source MAC address can be set with "none" keyword (e.g. `src-mac-address none`)
- The `src-mac-address` (if not already set by the attribute), `switch` and `ports` conditional parametrs are automatically set for each rule
- Each rule should end with an action property, supported values are either **drop** or **allow**. If no action property is set, the default **allow** value will be used.
- Multiple rules are supported for a single supplicant and they must be separated by a comma ","

Below are some examples of Mikrotik-Switching-Filter attributes and dynamic switch rules they create:

```
# Drop ARP frames (EtherType: 0x0806 or 2054)
Mikrotik-Switching-Filter = "mac-protocol 2054 action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid, D - dynamic
 0  D ;;; dot1x dynamic
      switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF mac-protocol=arp copy-
to-cpu=no redirect-to-cpu=no mirror=no new-dst-ports=""

# Allow UDP (IP protocol: 0x11 or 17) destination port 100 and drop all other packets
Mikrotik-Switching-Filter = "protocol 17 dst-port 100 action allow, action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid, D - dynamic
 0  D ;;; dot1x dynamic
      switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF protocol=udp dst-
port=100 copy-to-cpu=no redirect-to-cpu=no mirror=no

 1  D ;;; dot1x dynamic
      switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no new-dst-ports=""

# Allow only authenticated source MAC address, drop all other packets
Mikrotik-Switching-Filter = "action allow, src-mac-address none action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid; D - dynamic
 0  D ;;; dot1x dynamic
      switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:01:6D:EB/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no

 1  D ;;; dot1x dynamic
      switch=switch1 ports=ether1 copy-to-cpu=no redirect-to-cpu=no mirror=no new-dst-ports=""
```

In our example, Supplicant2 on ether2 is only allowed to access the 192.168.50.0/24 network with UDP destination port 50, all other traffic should be dropped. First, make sure that hardware offloading is working on bridge ports, otherwise switch rules might not work properly.

```
/interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                  BRIDGE                  HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-
COST    HORIZON
 0   H ether1                     bridge1                 yes    1     0x80         10
10       none
 1   H ether2                     bridge1                 yes    1     0x80         10
10       none
 2   H ether12                    bridge1                 yes    1     0x80         10
10       none
```

With enabled RADIUS debug logs it is possible to see complete RADIUS message packets with all attributes. In our example, Mikrotik-Switching-Filter attribute is received in Access-Accept message from Radius server:

```
02:35:38 radius,debug,packet received Access-Accept with id 121 from 10.1.2.3:1812
(..)
02:35:38 radius,debug,packet     MT-Switching-Filter = "mac-protocol 2048 dst-address 192.168.50.0/24 dst-
port 50 protocol 17 action allow,action drop"
```

The dynamic switch rules are now present under the switch menu:

```
/interface ethernet switch rule print
Flags: X - disabled, I - invalid, D - dynamic
 0  D ;;; dot1x dynamic
      switch=switch1 ports=ether2 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF mac-protocol=ip dst-
address=192.168.50.0/24 protocol=udp dst-port=50 copy-to-cpu=no redirect-to-cpu=no mirror=no

 1  D ;;; dot1x dynamic
      switch=switch1 ports=ether2 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no new-dst-ports=""
```


Dynamic switch rules will only apply to RouterBoards with switch rule support - CRS3xx, CRS5xx series switches, CCR2116, CCR2216, and devices with QCA8337, Atheros8327 and Atheros8316 switch chips. CRS1xx/2xx series switches do no support this functionality. Take into consideration the maximum number of rules for each device, see CRS3xx, CRS5xx, CCR2116, CCR2216 table and basic switch chip table

RouterOS Supplicant configuration

CA certificates are required for `eap-tls, eap-ttls` and `eap-peap` authentication methods. Additionally a client certificate is required for `eap-tls` method. For this example we have already imported a P12 certificate bundle with self signed client and CA certificates. For more information how to import certificates in RouterOS, please visit System/Certificates.

```
/certificate print
Flags: K - private-key, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked, E - expired, T
- trusted
 #         NAME                                              COMMON-
NAME                                       SUBJECT-ALT-NAME
FINGERPRINT
 0 K  A  T dot1x-client                                      ez_dot1x-
client                                   IP:10.1.2.34
 1  L A  T dot1x CA                                         ca
```

Simply add a new dot1x client instance that will initiate authentication process.

```
/interface dot1x client
add anon-identity=anonymous client-certificate=dot1x-client eap-methods=eap-tls identity=dot1x-user
interface=ether1 password=dot1xtest
```

If authentication was successful, the interface should have status authenticated.

```
/interface dot1x client print
Flags: I - inactive, X - disabled
 0   interface=ether1 eap-methods=eap-peap identity="dot1x-user" password="dot1xtest" anon-identity="
anonymous" client-certificate=dot1x-client status="authenticated"
```

HotSpot (Captive portal)

Introduction

The MikroTik HotSpot Gateway provides authentication for clients before access to public networks.

Hotspot (captive portal) - uses web-proxy and it is capable of using only the default routing table, at the moment. Making the PCC(per connection-classifier) not a valid method, due to the, multiple routing tables used.

HotSpot Gateway features:

- different authentication methods of clients, using a local client database on the router, or remote RADIUS server;
- users accounting in a local database on the router, or on remote RADIUS server;
- a walled-garden system, access to some web pages without authorization;
- login page modification, where you can put information about the company;
- automatic and transparent change any IP address of a client to a valid address;
- HotSpot can inform DHCP clients that they are behind a captive portal (RFC7710);

A hotspot can work reliably only when IPv4 is used. Hotspot relies on Firewall NAT rules which currently are not supported for IPv6.

Example

```
[admin@MikroTik] /ip hotspot> setup
Select interface to run HotSpot on

hotspot interface: ether3
Set HotSpot address for interface

local address of network: 10.5.50.1/24
masquerade network: yes
Set pool for HotSpot addresses

address pool of network: 10.5.50.2-10.5.50.254
Select hotspot SSL certificate

select certificate: none
Select SMTP server

ip address of smtp server: 0.0.0.0
Setup DNS configuration

dns servers: 10.1.101.1
DNS name of local hotspot server

dns name: myhotspot
Create local hotspot user

name of local hotspot user: admin
password for the user:
[admin@MikroTik] /ip hotspot>
```

Verify HotSpot configuration:

```
[admin@MikroTik] /ip hotspot> print
Flags: X - disabled, I - invalid, S - HTTPS
# NAME INTERFACE ADDRESS-POOL PROFILE IDLE-TIMEOUT
0 hotspot1 ether3 hs-pool-3 hsprof1 5m
[admin@MikroTik] /ip hotspot>
[admin@MikroTik] /ip pool> print
# NAME RANGES
0 hs-pool-3 10.5.50.2-10.5.50.254
[admin@MikroTik] /ip pool> /ip dhcp-server
[admin@MikroTik] /ip dhcp-server> print
Flags: X - disabled, I - invalid
# NAME INTERFACE RELAY ADDRESS-POOL LEASE-TIME ADD-ARP
0 dhcp1 ether3 hs-pool-3 1h
[admin@MikroTik] /ip dhcp-server> /ip firewall nat
[admin@MikroTik] /ip firewall nat> print
Flags: X - disabled, I - invalid, D - dynamic
0 X ;;; place hotspot rules here
chain=unused-hs-chain action=passthrough

1 ;;; masquerade hotspot network
chain=srcnat action=masquerade src-address=10.5.50.0/24
[admin@MikroTik] /ip firewall nat>
```

Parameters asked during the setup process

Parameter	Description
hotspot interface (*string*; Default: **allow**)	Interface name on which to run HotSpot. To run HotSpot on a bridge interface, make sure public interfaces are not included in the bridge ports.
local address of network (*IP*; Default: **10.5.50.1/24**)	HotSpot gateway address

masquerade network (*yes* | *no*; Default: **yes**)	Whether to masquerade HotSpot network, when **yes** rule is added to */ip firewall nat* with *action=masquerade*
address pool of network (*string*; Default: **yes**)	Address pool for HotSpot network, which is used to change user IP address to a valid address. Useful if providing network access to mobile clients that are not willing to change their networking settings.
select certificate (*none* | *import-other-certificate*; Default:)	Choose SSL certificate, when HTTPS authorization method is required.
ip address of smtp server (*IP*; Default: **0.0.0.0**)	The IP address of the SMTP server, where to redirect HotSpot's network SMTP requests (25 TCP port)
dns servers (*IP*; Default: **0.0.0.0**)	DNS server addresses used for HotSpot clients, configuration taken from */ip dns* menu of the HotSpot gateway
dns name (*string*; Default: **""**)	the domain name of the HotSpot server, a full qualified domain name is required, for example, www.example.com
name of local hotspot user (*string*; Default: **"admin"**)	username of one automatically created HotSpot user, added to */ip hotspot user*
password for the user' (*string*; Default:)	Password for automatically created HotSpot user

IP HotSpot

```
/ip/hotspot
```

The menu is designed to manage the HotSpot servers of the router. It is possible to run HotSpot on Ethernet, wireless, VLAN, and bridge interfaces. One HotSpot server is allowed per interface. When HotSpot is configured on the bridge interface, set HotSpot interface as bridge interface, not as bridge port, do not add public interfaces to bridge ports. You can add HotSpot servers manually to the */ip/hotspot* menu, but it is advised to run */ip/hotspot/setup*, which adds all necessary settings.

Parameters	Description
name (text)	HotSpot server's name or identifier
address-pool (name/ none; default: *none*)	address space used to change HotSpot client *any* IP address to a valid address. Useful for providing public network access to mobile clients that are not willing to change their networking settings
idle-timeout (time /none; default: *5m*)	period of inactivity for unauthorized clients. When there is no traffic from this client (literally client computer should be switched off), once the timeout is reached, a user is dropped from the HotSpot host list, its used address becomes available
keepalive-timeout (time/none; default: *none*)	Value of how long host can stay out of reach to be removed from the HotSpot
login-timeout (time /none; default: *none*)	Period of time after which if a host hasn't been authorized itself with a system the host entry gets deleted from host table. Loop repeats until the host logs in the system. Enable if there are situations where a host cannot log in after being too long in the host table unauthorized.
interface (name of an interface)	Interface to run HotSpot on
addresses-per-mac (integer/unlimited; default: 2)	Number of IP addresses allowed to be bind with the MAC address, when multiple HotSpot clients connected with one MAC-address
profile (name; default: ***default***)	HotSpot server default HotSpot profile, which is located in */ip/hotspot/profile*

Read-only

Parameters	Description

keepalive-timeout (read-only; time)	The exact value of the keepalive-timeout, that is applied to the user. Value shows how long the host can stay out of reach to be removed from the HotSpot

IP HotSpot Active

```
/ip/hotspot/active
```

HotSpot active menu shows all clients authenticated in HotSpot, the menu is informational (read-only) it is not possible to change anything here.

Parameters	Description
server (read-only; name)	HotSpot server name client is logged in
user (read-only; name)	name of the HotSpot user
domain (read-only; text)	the domain of the user (if split from the username), a parameter is used only with RADIUS authentication
address (read-only; IP address)	The IP address of the HotSpot user
mac-address (read-only; MAC-address)	MAC-address of the HotSpot user
login-by (read-only; multiple-choice: cookie **/** http-chap **/** http-pap **/** https **/** mac **/** mac-cookie **/** trial)	the authentication method used by the HotSpot client
uptime (read-only; time)	current session time of the user, it is showing how long the user has been logged in
idle-time (read-only; time)	the amount of time the user has been idle
session-time-left (read-only; time)	the exact value of session-time, that is applied for the user. Value shows how long user is allowed to be online to be logged off automatically by **uptime** reached
idle-timeout (read-only; time)	the exact value of the user's idle-timeout
keepalive-timeout (read-only; time)	the exact value of the keepalive-timeout, that is applied for the user. Value shows how long the host can stay out of reach to be removed from the HotSpot
limit-bytes-in (read-only; integer)	value shows how many bytes received from the client, an option is active when the appropriate parameter is configured for HotSpot user
limit-bytes-out (read-only; integer)	value shows how many bytes send to the client, an option is active when the appropriate parameter is configured for HotSpot user
limit-bytes-total (read-only; integer)	value shows how many bytes total were send/received from the client, an option is active when the appropriate parameter is configured for HotSpot user

IP HotSpot Host

```
/ip/hotspot/host
```

The host table lists all computers connected to the HotSpot server. The host table is informational and it is not possible to change any value there:

Parameters	Description
mac-address (read-only; MAC-address)	HotSpot user MAC-address
address (read-only; IP address)	HotSpot client original IP address
to-address (read-only; IP address)	The new client address assigned by HotSpot might be the same as the original **address**

server (read-only; name)	HotSpot server name client is connected to
bridge-port (read-only; name)	*"/interface bridge port"* the client is connected to, value is unknown when HotSpot is not configured on the bridge
uptime (read-only; time)	value shows how long the user is online (connected to the HotSpot)
idle-time (read-only; time)	time user has been idle
idle-timeout (read-only; time)	value of the client idle-timeout (unauthorized client)
keepalive-timeout (read-only; time)	keepalive-timeout value of the unauthorized client
bytes-in (read-only; integer)	amount of bytes received from an unauthorized client
packet-in (read-only; integer)	amount of packets received from an unauthorized client
bytes-out (read-only; integer)	amount of bytes sent to an unauthorized client
packet-out (read-only; integer)	amount of packets sent to an unauthorized client

IP Binding

```
/ip/hotspot/ip-binding
```

IP-Binding HotSpot menu allows to the setup of static One-to-One NAT translations, allows to bypass specific HotSpot clients without any authentication, and also allows to block specific hosts and subnets from the HotSpot network

Property	Description
address (*IP Range*; Default: **""**)	The original IP address of the client
mac-address (*MAC*; Default: **""**)	MAC address of the client
server (*string* | *all*; Default: **"all"**)	Name of the HotSpot server. • **all** - will be applied to all hotspot servers
to-address (*IP*; Default: **""**)	New IP address of the client, translation occurs on the router (client does not know anything about the translation)
type (*blocked* | *bypassed* | *regular*; Default: **""**)	Type of the IP-binding action • **regular** - performs One-to-One NAT according to the rule, translates the **address** to **to-address** • **bypassed** - performs the translation, but excludes client from login to the HotSpot • **blocked** - translation is not performed and packets from a host are dropped

Cookies

The menu contains all cookies sent to the HotSpot clients, which are authorized by cookie method, all the entries are read-only.

```
/ip/hotspot/cookie
```

Property	Description
domain (*string*)	The domain name (if split from the username)
expires-in (*time*)	How long the cookie is valid

mac-address (*MAC*)	Client's MAC-address
user (*string*)	HotSpot username

Using DHCP option to advertise HotSpot URL

Most devices, such as modern smartphones, do some kind of background checking to see if they are behind a captive portal. They do this by requesting a known webpage and comparing the contents of that page, to what they should be. If contents are different, the device assumes there is a login page and creates a popup with this login page.

This does not always happen, as this "known webpage" could be blocked, whitelisted, or not accessible in internal networks. To improve on this mechanism, RFC 7710 was created, allowing the HotSpot to inform all DHCP clients that they are behind a captive-portal device and that they will need to authenticate to get Internet access, regardless of what webpages they do or do not request.

This DHCP option field is enabled automatically, but only if the router has a DNS name configured and has a valid SSL certificate (so that the login page can be accessed over HTTPS). When these requirements are met, a special DHCP option will be sent, containing a link to `https://<dns-name-of-hotspot>/api`. This link contains information in JSON format, instructing the client device of the captive portal status, and the location of the login page.

Contents of `https://<dns-name-of-hotspot>/api` are as follows:

```
{
"captive": $(if logged-in == 'yes')false$(else)true$(endif),
"user-portal-url": "$(link-login-only)",
$(if session-timeout-secs != 0)
"seconds-remaining": $(session-timeout-secs),
$(endif)
$(if remain-bytes-total)
"bytes-remaining": $(remain-bytes-total),
$(endif)
"can-extend-session": true
}
```

Some devices require venue-info URL as well, so you are free to modify the api.json file to your liking, just like any other hotspot files. It is located in the router files menu.

> **Important**
>
> If you have set up Hotspot before RouterOS v7.3 when RFC 7710 was implemented, you will have to use "Reset HTML" function, or manually add/edit the api.json file to have the above contents, for Hotspot detection to work.

Hotspot customisation

Requirements

- HotSpot captive portal

Introduction

You can create a completely different set of servlet pages for each HotSpot server you have, specifying the directory in the "html-override-directory" property of a HotSpot server profile /ip hotspot profile. The default servlet pages are copied in the directory "hotspot" directory right after you create the server profile. This directory can be accessed by connecting to the router with an FTP client. You can copy this directory and modify the pages as you like using the information from this section of the manual. Note that it is suggested to edit the files manually, as automated HTML editing tools may corrupt the pages by removing variables or other vital parts. After you are finished with content modification you need to upload this modified content to some custom directory on the hotspot router and point previously mentioned property "html-override-directory" value as path to this new custom HTML directory.

Note: If "html-override-directory" value path is missing or empty then the hotspot server will revert to default HTML files.

Available Pages

Main HTML servlet pages, which are shown to the user:

- **redirect.html** - redirects the user to another URL (for example, to the login page)
- **login.html** - login page shown to a user to ask for a username and password. This page may take the following parameters:
  - **username** - username
  - **password** - either plain-text password (in case of PAP authentication) or MD5 hash of chap-id variable, password, and CHAP challenge (in case of CHAP authentication). This value is used as e-mail address for trial users
  - **dst** - original URL requested before the redirect. This will be opened on successful login
  - **popup** - whether to pop-up a status window on successful login
  - **radius<id>** - send the attribute identified with <id> in text string form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
  - **radius<id>u** - send the attribute identified with <id> in unsigned integer form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
  - **radius<id>-<vnd-id>** - send the attribute identified with <id> and vendor ID <vnd-id> in text string form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
  - **radius<id>-<vnd-id>u** - send the attribute identified with <id> and vendor ID <vnd-id> in unsigned integer form to the RADIUS server (in case RADIUS authentication is used; lost otherwise)
- **md5.js** - JavaScript for MD5 password hashing. Used together with http-chap login method
- **alogin.html** - page shown after a client has logged in. It pops-up status page and redirects the browser to the originally requested page (before he /she was redirected to the HotSpot login page)
- **status.html** - status page, shows statistics for the client. It is also able to display advertisements automatically
- **logout.html** - logout page, shown after a user is logged out. Shows final statistics about the finished session. This page may take the following additional parameters:
  - **erase-cookie** - whether to erase cookies from the HotSpot server on logout (makes it impossible to log in with cookie next time from the same browser, might be useful in multiuser environments)
- **error.html** - error page, shown on fatal errors only

Some other pages are available as well, if more control is needed:

- **rlogin.html** - page, which redirects the client from some other URL to the login page, if authorization of the client is required to access that URL
- **rstatus.html** - similar to rlogin.html, only in case if the client is already logged in and the original URL is not known
- **radvert.html** - redirects the client to the scheduled advertisement link
- **flogin.html** - shown instead of login.html, if some error has happened (invalid username or password, for example)
- **fstatus.html** - shown instead of redirect, if a status page is requested, but the client is not logged in
- **flogout.html** - shown instead of redirect, if logout page is requested, but the client is not logged in

Serving Servlet Pages

The HotSpot servlet recognizes 5 different request types:

1. **request for a remote host**
   - if user is logged in and advertisement is due to be displayed, radvert.html is displayed. This page redirects to the scheduled advertisement page
   - if user is logged in and advertisement is not scheduled for this user, the requested page is served
   - if user is not logged in, but the destination host is allowed by the walled garden, then the request is also served

- if user is not logged in, and the destination host is disallowed by the walled garden, rlogin.html is displayed; if rlogin.html is not found, redirect.html is used to redirect to the login page

2. **request for "/" on the HotSpot host**
   - if user is logged in, rstatus.html is displayed; if rstatus.html is not found, redirect.html is used to redirect to the status page
   - if user is not logged in, rlogin.html is displayed; if rlogin.html is not found, redirect.html is used to redirect to the login page

3. **request for "/login" page**
   - if user has successfully logged in (or is already logged in), alogin.html is displayed; if alogin.html is not found, redirect.html is used to redirect to the originally requested page or the status page (in case, the original destination page was not given)
   - if user is not logged in (username was not supplied, no error message appeared), login.html is showed
   - if login procedure has failed (an error message is supplied), flogin.html is displayed; if flogin.html is not found, login.html is used
   - in case of fatal errors, error.html is showed

4. **request for "/status" page**
   - if user is logged in, status.html is displayed
   - if user is not logged in, fstatus.html is displayed; if fstatus.html is not found, redirect.html is used to redirect to the login page

5. **request for '/logout' page**
   - if user is logged in, logout.html is displayed
   - if user is not logged in, flogout.html is displayed; if flogout.html is not found, redirect.html is used to redirect to the login page

Note: If it is not possible to meet a request using the pages stored on the router's FTP server, Error 404 is displayed

There are many ways to customize what the HotSpot authentication pages look like:

- The pages are easily modifiable. They are stored on the router's FTP server in the directory you choose for the respective HotSpot server profile.
- By changing the variables, which client sends to the HotSpot servlet, it is possible to reduce the keyword count to one (username or password; for example, the client's MAC address may be used as the other value) or even to zero (License Agreement; some predefined values general for all users or client's MAC address may be used as username and password)
- Registration may occur on a different server (for example, on a server that is able to charge Credit Cards). Client's MAC address may be passed to it, so that this information doesn't have to be entered manually. After the registration, the server should change RADIUS database enabling client to log in for some amount of time.

To insert a variable in some place in the HTML file, the $(var_name) syntax is used, where the "var_name" is the name of the variable (without quotes). This construction may be used in any HotSpot HTML file accessed as '/', '/login', '/status' or '/logout', as well as any text or HTML (.txt, .htm or .html) file stored on the HotSpot server (with the exception of traffic counters, which are available in status page only, and **error**, **error-orig**, **chap-id**, **chap-challenge** and **popup** variables, which are available in login page only). For example, to show a link to the login page, following construction can be used:

```
<a href="$(link-login)">login</a>
```

Variables

All of the Servlet HTML pages use variables to show user specific values. Variable names appear only in the HTML source of the servlet pages - they are automatically replaced with the respective values by the HotSpot Servlet. For most variables, there is an example of their possible value included in brackets. All the described variables are valid in all servlet pages, but some of them just might be empty at the time they are accessed (for example, there is no uptime before a user has logged in).

List of available variables

Note: Some of the variables use hard-coded http URL, if you are using https, you can construct the link in some other way, for example for $link-status, you can use https://$(hostname)/$(target-dir)status

Common server variables:

- **hostname** - DNS name or IP address (if DNS name is not given) of the HotSpot Servlet ("hotspot.example.net")
- **identity** - RouterOS identity name ("MikroTik")
- **login-by** - authentication method used by user
- **plain-passwd** - a "yes/no" representation of whether HTTP-PAP login method is allowed ("no")
- **server-address** - HotSpot server address ("10.5.50.1:80")
- **ssl-login** - a "yes/no" representation of whether HTTPS method was used to access that servlet page ("no")
- **server-name** - HotSpot server name (set in the /ip hotspot menu, as the name property)

Links:

- **link-login** - link to login page including original URL requested ("http://10.5.50.1/login?dst=http://www.example.com/")
- **link-login-only** - link to login page, not including original URL requested ("http://10.5.50.1/login")
- **link-logout** - link to logout page ("http://10.5.50.1/logout")
- **link-status** - link to status page ("http://10.5.50.1/status")
- **link-orig** - original URL requested ("http://www.example.com/")

General client information:

- **domain** - domain name of the user ("example.com")
- **interface-name** - physical HotSpot interface name (in case of bridged interfaces, this will return the actual bridge port name)
- **ip** - IP address of the client ("10.5.50.2")
- **logged-in** - "yes" if the user is logged in, otherwise - "no" ("yes")
- **mac** - MAC address of the user ("01:23:45:67:89:AB")
- **trial** - a "yes/no" representation of whether the user has access to trial time. If user's trial time has expired, the value is "no"
- **username** - the name of the user ("John")
- **host-ip** - client IP address from /ip hotspot host table
- **vlan-id** - Represents ID of a VLAN interface from which the client is connected

User status information:

- **idle-timeout** - idle timeout ("20m" or "" if none)
- **idle-timeout-secs** - idle timeout in seconds ("88" or "0" if there is such timeout)
- **limit-bytes-in** - byte limit for send ("1000000" or "---" if there is no limit)
- **limit-bytes-out** - byte limit for receive ("1000000" or "---" if there is no limit)
- **refresh-timeout** - status page refresh timeout ("1m30s" or "" if none)
- **refresh-timeout-secs** - status page refresh timeout in seconds ("90s" or "0" if none)
- **session-timeout** - session time left for the user ("5h" or "" if none)
- **session-timeout-secs** - session time left for the user, in seconds ("3475" or "0" if there is such timeout)
- **session-time-left** - session time left for the user ("5h" or "" if none)
- **session-time-left-secs** - session time left for the user, in seconds ("3475" or "0" if there is such timeout)
- **uptime** - current session uptime ("10h2m33s")
- **uptime-secs** - current session uptime in seconds ("125")

Traffic counters, which are available only on the status page:

- **bytes-in** - number of bytes received from the user ("15423")
- **bytes-in-nice** - user-friendly form of number of bytes received from the user ("15423")
- **bytes-out** - number of bytes sent to the user ("11352")
- **bytes-out-nice** - user-friendly form of number of bytes sent to the user ("11352")
- **packets-in** - number of packets received from the user ("251")
- **packets-out** - number of packets sent to the user ("211")
- **remain-bytes-in** - remaining bytes until limit-bytes-in will be reached ("337465" or "---" if there is no limit)
- **remain-bytes-out** - remaining bytes until limit-bytes-out will be reached ("124455" or "---" if there is no limit)

Miscellaneous variables:

- **session-id** - value of 'session-id' parameter in the last request
- **var** - value of 'var' parameter in the last request
- **error** - error message, if something failed ("invalid username or password")
- **error-orig** - original error message (without translations retrieved from errors.txt), if something failed ("invalid username or password")
- **chap-id** - value of chap ID ("\371")
- **chap-challenge** - value of chap challenge ("\357\015\330\013\021\234\145\245\303\253\142\246\133\175\375\316")
- **popup** - whether to pop-up checkbox ("true" or "false")
- **advert-pending** - whether an advertisement is pending to be displayed ("yes" or "no")
- **http-status** - allows the setting of the http status code and message
- **http-header** - allows the setting of the http header

RADIUS-related variables:

- **radius<id>** - show the attribute identified with <id> in text string form (in case RADIUS authentication was used; "" otherwise)
- **radius<id>u** - show the attribute identified with <id> in unsigned integer form (in case RADIUS authentication was used; "0" otherwise)
- **radius<id>-<vnd-id>** - show the attribute identified with <id> and vendor ID <vnd-id> in text string form (in case RADIUS authentication was used; "" otherwise)
- **radius<id>-<vnd-id>u** - show the attribute identified with <id> and vendor ID <vnd-id> in unsigned integer form (in case RADIUS authentication was used; "0" otherwise)

Working with variables

$(if <var_name>) statements can be used in these pages. The following content will be included, if value of <var_name> will not be an empty string. It is an equivalent to $(if <var_name> != "") It is possible to compare on equivalence as well: $(if <var_name> == <value>) These statements have effect until $(elif <var_name>), $(else) or $(endif). In general case it looks like this:

```
some content, which will always be displayed
$(if username == john)
Hey, your username is john
$(elif username == dizzy)
Hello, Dizzy! How are you? Your administrator.
$(elif ip == 10.1.2.3)
You are sitting at that old computer, which is so slow...
$(elif mac == 00:01:02:03:04:05)
This is an ethernet card, which was stolen few months ago...
$(else)
I don't know who you are, so lets live in peace.
$(endif)
other content, which will always be displayed
```

Only one of those expressions will be shown. Which one - depends on the values of those variables for each client.

Redirects and custom Headers

```
$(if http-status == 302)Hotspot login required$(endif)
$(if http-header == "Location")$(link-redirect)$(endif)
```

Note: Although the above appears to use the conditional expression 'if' it is in fact setting the 'http-status' to '302' not testing for it. Also the same for the variable 'http-header'. Once again, even though it uses an 'if' it is in fact setting the variable to 'Location' followed by the URL set from the variable 'link-redirect'.

For example, in the case where $(link-redirect) evaluates to "http://192.168.88.1/login", then the HTTP response returned to the client will be changed to:

```
HTTP/1.0 302 Hotspot login required
<regular HTTP headers>
Location: http://192.168.88.1/login
```

http-status syntax:

```
$(if http-status == XYZ)HTTP_STATUS_MESSAGE$(endif)
```

- *XYZ* - The status code you wish to return. Should be 3 decimal digits, the first one must not be 0
- *HTTP_STATUS_MESSAGE* - any text you wish to return to the client that will follow the above status code in the HTTP reply

In any HTTP response it will be on the first line and will be as follows:

```
HTTP/1.0 XYZ HTTP_STATUS_MESSAGE
```

http-header syntax:

```
$(if http-header == HTTP_HEADER_NAME)HTTP_HEADER_VALUE$(endif)
```

- *HTTP_HEADER_NAME* - name of the HTTP header to be sent in the response
- *HTTP_HEADER_VALUE* - the value of the HTTP header with the name HTTP_HEADER_NAME to be sent in the response

The HTTP response will appear as:

```
HTTP_HEADER_NAME: HTTP_HEADER_VALUE
```

All variables and conditional expressions within HTTP_HEADER_VALUE and HTTP_STATUS_MESSAGE are processed as usual.

In case multiple headers with the same name are added, then only the last one will be used (previous ones will be discarded). It allows the system to override regular HTTP headers (for example, Content-Type and Cache-Control).

Customizing Error Messages

All error messages are stored in the errors.txt file within the respective HotSpot servlet directory. You can change and translate all these messages to your native language. To do so, edit the errors.txt file. You can also use variables in the messages. All instructions are given in that file.

Multiple Versions of HotSpot Pages

Multiple HotSpot page sets for the same HotSpot server are supported. They can be chosen by the user (to select language) or automatically by JavaScript (to select PDA/regular version of HTML pages).

To utilize this feature, create subdirectories in the HotSpot HTML directory, and place those HTML files, which are different, in that subdirectory. For example, to translate everything in Latvian, the subdirectory "lv" can be created with login.html, logout.html, status.html, alogin.html, radvert.html and errors.txt files, which are translated into Latvian. If the requested HTML page can not be found in the requested subdirectory, the corresponding HTML file from the main directory will be used. Then main login.html file would contain a link to "/lv/login?dst=$(link-orig-esc)", which then displays Latvian version of login page: Latviski . And Latvian version would contain a link to English version: English

Another way of referencing directories is to specify 'target' variable:

```
<a href="$(link-login-only)?dst=$(link-orig-esc)&target=lv">Latviski</a>
<a href="$(link-login-only)?dst=$(link-orig-esc)&target=%2F">English</a>
```

After the preferred directory has been selected (for example, "lv"), all links to local HotSpot pages will contain that path (for example, $(link-status) = "http://hotspot.mt.lv/lv/status"). So, if all HotSpot pages reference links using "$(link-xxx)" variables, then no more changes are to be made - each client will stay within the selected directory all the time.

Misc

If you want to use the HTTP-CHAP authentication method, you need to include the **doLogin()** function (which references to the **md5.js** which must be already loaded) before the **Submit action** of the login form. Otherwise, CHAP login will fail.

The resulting password to be sent to the HotSpot gateway in case of HTTP-CHAP method, it is formed by MD5-hashing the concatenation of the following: chap-id, the password of the user, and chap-challenge (in the given order)

In case variables are to be used in the link directly, then they must be escaped accordingly. For example, in login page, **link** will not work as intended, if username will be "123&456=1 2". In this case instead of $(user), its escaped version must be used: $(user-esc): **link**. Now the same username will be converted to "123%26456%3D1+2", which is the valid representation of "123&456=1 2" in URL. This trick may be used with any variables, not only with $(username).

There is a boolean parameter "erase-cookie" to the logout page, which may be either "on" or "true" to delete user cookie on logout (so that the user would not be automatically logged on when he/she opens a browser next time.

Examples

With basic HTML language knowledge and the examples below it should be easy to implement the ideas described above.

- To provide predefined value as username, in login.html change:

```
<type="text" value="$(username)>
```

to this line:

```
<input type="hidden" name="username" value="hsuser">
```

(where hsuser is the username you are providing)

- To provide a predefined value as a password, in login.html change:

```
<input type="password">
```

to this line:

```
<input type="hidden" name="password" value="hspass">
```

(where hspass is the password you are providing)

- To send the client's MAC address to a registration server in the form of:

https://www.example.com/register.html?mac=XX:XX:XX:XX:XX:XX

change the Login button link in login.html to:

```
https://www.example.com/register.html?mac=$(mac)
```

(you should correct the link to point to your server)

- To show a banner after user login, in alogin.html after

$(if popup == 'true') add the following line:

```
open('http://www.example.com/your-banner-page.html', 'my-banner-name','');
```

(you should correct the link to point to the page you want to show)

- To choose a different page shown after login, in login.html change:

```
<input type="hidden" name="dst" value="$(link-orig)">
```

to this line:

```
<input type="hidden" name="dst" value="http://www.example.com">
```

(you should correct the link to point to your server)

- To erase the cookie on logoff, in the page containing a link to the logout (for example, in status.html) change:

```
open('$(link-logout)', 'hotspot_logout', ...
```

to this:

```
open('$(link-logout)?erase-cookie=on', 'hotspot_logout', ...
```

or alternatively add this line:

```
<input type="hidden" name="erase-cookie" value="on">
```

before this one:

```
<input type="submit" value="log off">
```

External authentication

Another example is making HotSpot to authenticate on a remote server (which may, for example, perform credit card charging):

- Allow direct access to the external server in walled-garden (either HTTP-based or IP-based)
- Modify the login page of the HotSpot servlet to redirect to the external authentication server. The external server should modify the RADIUS database as needed

Here is an example of such a login page to put on the HotSpot router (it is redirecting to https://auth.example.com/login.php, replace with the actual address of an external authentication server):

```
<html>
<title>...</title>
<body>
<form name="redirect" action="https://auth.example.com/login.php" method="post">
<input type="hidden" name="mac" value="$(mac)">
<input type="hidden" name="ip" value="$(ip)">
<input type="hidden" name="username" value="$(username)">
<input type="hidden" name="link-login" value="$(link-login)">
<input type="hidden" name="link-orig" value="$(link-orig)">
<input type="hidden" name="error" value="$(error)">
</form>
<script language="JavaScript">
<!--
        document.redirect.submit();
//-->
</script>
</body>
</html>
```

- The external server can log in a HotSpot client by redirecting it back to the original HotSpot servlet login page, specifying the correct username and password

Here is an example of such a page (it is redirecting to https://hotspot.example.com/login, replace with the actual address of a HotSpot router; also, it is displaying www.mikrotik.com after successful login, replace with what is needed):

```
<html>
<title>Hotspot login page</title>
<body>
<form name="login" action="https://hotspot.example.com/login" method="post">
<input type="text" name="username" value="demo">
<input type="password" name="password" value="none">
<input type="hidden" name="domain" value="">
<input type="hidden" name="dst" value="http://www.mikrotik.com/">
<input type="submit" name="login" value="log in">
</form>
</body>
</html>
```

- Hotspot will ask the RADIUS server whether to allow the login or not. If allowed, alogin.html page will be displayed (it can be modified to do anything). If not allowed, the flogin.html (or login.html) page will be displayed, which will redirect the client back to the external authentication server.

Note: as shown in these examples, HTTPS protocol and POST method can be used to secure communications.

HTTP header detection

The Hotspot login pages have access to HTTP headers by using **$(http-header-name);**

For example, there exists an ability to check the user agent (or browser), and will return any other content instead of the regular login page, if so desired. This can be used to disable automatic popups in phones, for example.

For example, to output "SUCCESS" for users of a specific Firefox mobile version, instead of the login page, you can these lines on the top of the **rlogin.html** page in your hotspot directory:

```
$(if user-agent == "Mozilla/5.0 (Android; Mobile; rv:40.0) Gecko/40.0 Firefox/40.0" )
<HTML><HEAD><TITLE>Success</TITLE></HEAD><BODY>Success</BODY></HTML>
$(else)
---- regular content of rlogin.html page  ----
$(endif)
```

This will DISABLE the login popup for Android Firefox 40 users.

One-click login

It is possible to create a modified captive portal for quick one-click login for scenarios where no user or password is required.

What you need to do is:

- Create a user for this purpose. In example, it is "notsosecretuser" with password "notsosecretpass"
- Assign this user to a user profile that allows a specific/unlimited amount of simultaneous active users.
- Copy original hotspot directory that is already generated in routers file menu on root level.
- Modify the contents of this copy directory contents.
    - Only one file requires modifications for this to work, the "login.html".

Original:

```
<table width="100" style="background-color: #ffffff">
  <tr><td align="right">login</td>
      <td><input style="width: 80px" name="username" type="text" value="$(username)"/></td>
  </tr>
  <tr><td align="right">password</td>
      <td><input style="width: 80px" name="password" type="password"/></td>
  </tr>
  <tr><td> </td>
      <td><input type="submit" value="OK" /></td>
  </tr>
</table>
```

Modified:

```
<table width="100" style="background-color: #ffffff">
  <tr style="display:none;"><td align="right">login</td>
    <td><input style="width: 80px" name="username" type="text" value="notsosecretuser"/></td>
  </tr>
  <tr style="display:none;"><td align="right">password</td>
    <td><input style="width: 80px" name="password" type="password" value="notsosecretpass"/></td>
  </tr>
  <tr><td> </td>
    <td><input type="submit" value="Proceed to Internet!" /></td>
  </tr>
</table>
```

What changed:

- User and Password "" fields are hidden.
- Both User and Password field values contain predefined values.
- Changed the "OK" button value(name) to something more fitting.

- Now upload this new hotspot folder back to the router, preferably with a different name.
- Change settings in the hotspot server profile to use this new html directory.

```
/ip hotspot profile set (profile number or name) html-directory-override=(dir path/name)
```

Firewall customizations

Summary

Apart from the obvious dynamic entries in the /ip hotspot submenu itself (like hosts and active users), some additional rules are added in the firewall tables when activating a HotSpot service.

NAT

From **/ip firewall nat print dynamic** command, you can get something like this (comments follow after each of the rules):

```
0 D chain=dstnat action=jump jump-target=hotspot hotspot=from-client
```

Putting all HotSpot-related tasks for packets from all HotSpot clients into a separate chain.

```
1 I chain=hotspot action=jump jump-target=pre-hotspot
```

Any actions that should be done before HotSpot rules apply, should be put in the pre-hotspot chain. This chain is under full administrator control and does not contain any rules set by the system, hence the invalid jump rule (as the chain does not have any rules by default).

```
2 D chain=hotspot action=redirect to-ports=64872 dst-port=53 protocol=udp
3 D chain=hotspot action=redirect to-ports=64872 dst-port=53 protocol=tcp
```

Redirect all DNS requests to the HotSpot service. The 64872 port provides DNS service for all HotSpot users. If you want the HotSpot server to listen to another port, add rules here the same way, changing dst-port property.

```
4 D chain=hotspot action=redirect to-ports=64873 hotspot=local-dst dst-port=80
    protocol=tcp
```

Redirect all HTTP login requests to the HTTP login servlet. The 64873 is HotSpot HTTP servlet port.

```
5 D chain=hotspot action=redirect to-ports=64875 hotspot=local-dst dst-port=443
    protocol=tcp
```

Redirect all HTTPS login requests to the HTTPS login servlet. The 64875 is HotSpot HTTPS servlet port.

```
6 D chain=hotspot action=jump jump-target=hs-unauth hotspot=!auth protocol=tcp
```

All other packets except DNS and login requests from unauthorized clients should pass through the hs-unauth chain.

```
7 D chain=hotspot action=jump jump-target=hs-auth hotspot=auth protocol=tcp
```

And packets from the authorized clients - through the hs-auth chain.

```
8 D ;;; www.mikrotik.com
    chain=hs-unauth action=return dst-address=66.228.113.26 dst-port=80 protocol=tcp
```

First in the **hs-unauth** chain is put everything that affects TCP protocol in the `/ip hotspot walled-garden ip` submenu (i.e., everything where either protocol is not set, or set to TCP). Here we are excluding www.mikrotik.com from being redirected to the login page.

```
9 D chain=hs-unauth action=redirect to-ports=64874 dst-port=80 protocol=tcp
```

All other HTTP requests are redirected to the Walled Garden proxy server which listens to the 64874 port. If there is an "allow" entry in the `/ip hotspot walled-garden` menu for an HTTP request, it is being forwarded to the destination. Otherwise, the request will be automatically redirected to the HotSpot login servlet (port 64873).

```
10 D chain=hs-unauth action=redirect to-ports=64874 dst-port=3128 protocol=tcp
11 D chain=hs-unauth action=redirect to-ports=64874 dst-port=8080 protocol=tcp
```

HotSpot by default assumes that only these ports may be used for HTTP proxy requests. These two entries are used to "catch" client requests to unknown proxies (you can add more rules here for other ports). I.e., to make it possible for the clients with unknown proxy settings to work with the HotSpot system. This feature is called "Universal Proxy". If it is detected that a client is using some proxy server, the system will automatically mark those packets with the HTTP hotspot mark to work around the unknown proxy problem, as we will see later on. Note that the port used (64874) is the same as for HTTP requests in rule #9 (so both HTTP and HTTP proxy requests are processed by the same code).

```
12 D chain=hs-unauth action=redirect to-ports=64875 dst-port=443 protocol=tcp
```

HTTPS proxy is listening on the 64875 port.

```
13 I chain=hs-unauth action=jump jump-target=hs-smtp dst-port=25 protocol=tcp
```

Redirect for SMTP protocol may also be defined in the HotSpot configuration. In case it is, a redirect rule will be put in the hs-smtp chain. This is done so that users with unknown SMTP configuration would be able to send their mail through the service provider's (your) SMTP server instead of going to the [possibly unavailable outside their network of origin] SMTP server users have configured on their computers. The chain is empty by default, hence the invalid jump rule.

```
14 D chain=hs-auth action=redirect to-ports=64874 hotspot=http protocol=tcp
```

Providing HTTP proxy service for authorized users. Authenticated user requests may need to be subject to transparent proxying (the "Universal Proxy" technique and advertisement feature). This HTTP mark is put automatically on the HTTP proxy requests to the servers detected by the HotSpot HTTP proxy (the one that is listening on the 64874 port) as HTTP proxy requests for unknown proxy servers. This is done so that users that have some proxy settings would use the HotSpot gateway instead of the [possibly unavailable outside their network of origin] proxy server users have configured in their computers. This mark is also applied when an advertisement is due to be shown to the user, as well as on any HTTP requests from the users whose profile is configured to transparently proxy their requests.

```
15 I chain=hs-auth action=jump jump-target=hs-smtp dst-port=25 protocol=tcp
```

Providing SMTP proxy for authorized users (the same as in rule #13).

Packet Filtering

From **/ip firewall filter print dynamic** command, you can get something like this (comments follow after each of the rules):

```
 0 D chain=forward action=jump jump-target=hs-unauth hotspot=from-client,!auth
```

Any packet that traverse the router from an unauthorized client will be sent to the **hs-unauth** chain. The hs-unauth implements the IP-based Walled Garden filter.

```
 1 D chain=forward action=jump jump-target=hs-unauth-to hotspot=to-client,!auth
```

Everything that comes to clients through the router, gets redirected to another chain, called **hs-unauth-to**. This chain should reject unauthorized requests to the clients.

```
 2 D chain=input action=jump jump-target=hs-input hotspot=from-client
```

Everything that comes from clients to the router itself, gets to yet another chain, called **hs-input**.

```
 3 I chain=hs-input action=jump jump-target=pre-hs-input
```

Before proceeding with [predefined] dynamic rules, the packet gets to the administratively controlled **pre-hs-input** chain, which is empty by default, hence the invalid state of the jump rule.

```
 4 D chain=hs-input action=accept dst-port=64872 protocol=udp
 5 D chain=hs-input action=accept dst-port=64872-64875 protocol=tcp
```

Allow client access to the local authentication and proxy services (as described earlier).

```
 6 D chain=hs-input action=jump jump-target=hs-unauth hotspot=!auth
```

All other traffic from unauthorized clients to the router itself will be treated the same way as the traffic traversing the routers.

```
 7 D chain=hs-unauth action=return protocol=icmp
 8 D ;;; www.mikrotik.com
     chain=hs-unauth action=return dst-address=66.228.113.26 dst-port=80 protocol=tcp
```

Unlike the NAT table where only TCP-protocol related Walled Garden entries were added, in the packet filter **hs-unauth** chain is added to everything you have set in the **/ip hotspot walled-garden ip** menu. That is why although you have seen only one entry in the NAT table, there are two rules here.

```
 9 D chain=hs-unauth action=reject reject-with=tcp-reset protocol=tcp
10 D chain=hs-unauth action=reject reject-with=icmp-net-prohibited
```

Everything else that has not been white-listed by the Walled Garden will be rejected. Note the usage of TCP Reset for rejecting TCP connections.

```
11 D chain=hs-unauth-to action=return protocol=icmp
12 D ;;; www.mikrotik.com
     chain=hs-unauth-to action=return src-address=66.228.113.26 src-port=80 protocol=tcp
```

The same action as in rules #7 and #8 is performed for the packets destined to the clients (chain **hs-unauth-to**) as well.

```
13 D chain=hs-unauth-to action=reject reject-with=icmp-host-prohibited
```

Reject all packets to the clients with an ICMP reject message.

PPP AAA

Summary

Sub-menu: `/ppp`

The MikroTik RouterOS provides scalable Authentication, Authorization, and Accounting (AAA) functionality.

Local authentication is performed using the User Database and the Profile Database. The actual configuration for the given user is composed using the respective user record from the User Database, the associated item from the Profile Database, and the item in the Profile database which is set as default for a given service the user is authenticating to. Default profile settings from the Profile database have the lowest priority while the user access record settings from the User Database have the highest priority with the only exception being particular IP addresses take precedence over IP pools in the local-address and remote-address settings, which are described later on.

Support for RADIUS authentication gives the ISP or network administrator the ability to manage PPP user access and accounting from one server throughout a large network. The MikroTik RouterOS has a RADIUS client that can authenticate for PPP, PPPoE, PPTP, L2TP, OPVN, and ISDN connections. The attributes received from the RADIUS server override the ones set in the default profile, but if some parameters are not received they are taken from the respective default profile.

User Profiles

Sub-menu: `/ppp profile`

PPP profiles are used to define default values for user access records stored under `/ppp secret` submenu. Settings in `/ppp secret` User Database overrides corresponding `/ppp profile` settings except that single IP addresses always take precedence over IP pools when specified as local-address or remote-address parameters.

Properties

Property	Description
address-list (*string*; Default:)	Address list name to which ppp assigned (on server) or received (on client) address will be added.
bridge (*string*; Default:)	Name of the bridge interface to which ppp interface will be added as a slave port. Both tunnel endpoints (server and client) must be in the bridge to make this work, see more details in the BCP bridging manual.
bridge-horizon (*integer 0..429496729*; Default:)	Used split-horizon value for the dynamically created bridge port. Can be used to prevent bridging loops and isolate traffic. Set the same value for a group of ports, to prevent them from sending data to ports with the same horizon value.
bridge-learning (*default* | *no* | *yes*; Default: **default**)	Changes MAC learning behavior on the dynamically created bridge port: • yes - enables MAC learning • no - disables MAC learning • default - derive this value from the interface default profile; same as yes if this is the interface default profile
bridge-path-cost (*integer 1..200000000*; Default:)	Used path cost for the dynamically created bridge port, used by STP/RSTP to determine the best path, used by MSTP to determine the best path between regions. This property has no effect when a bridge protocol-mode is set to none.
bridge-port-priority (*integer 0..240*; Default:)	Used priority for the dynamically created bridge port, used by STP/RSTP to determine the root port, used by MSTP to determine the root port between regions. This property has no effect when a bridge protocol-mode is set to none.

change-tcp-mss (*yes* | *no* | *default*; Default: **default**)	Modifies connection MSS settings (applies only for IPv4): • yes - adjust connection MSS value • no - do not adjust connection MSS value • default - derive this value from the interface default profile; same as no if this is the interface default profile
comment (*string*; Default:)	Profile comment
dhcpv6-pd-pool (*string*; Default:)	Name of the IPv6 pool which will be used by dynamically created DHCPv6 server when client connects. `Read more >>`
dns-server (*IP*; Default:)	IP address of the DNS server that is supplied to PPP clients
idle-timeout (*time*; Default:)	Specifies the amount of time after which the link will be terminated if there is no activity present. Timeout is not set by default
incoming-filter (*string*; Default:)	Firewall chain name for incoming packets. The specified chain gets control of each packet coming from the client. The ppp chain should be manually added and rules with action=jump jump-target=ppp should be added to other relevant chains for this feature to work. For more information look at the examples section
insert-queue-before (*bottom* | *first* | *queue name* ; Default:)	Inserts new queue as the last, first, or before a specified queue
interface-list (*interface list name*; Default:)	Specifies interface list to which profile interfaces will be added
local-address (*IP address* | *pool*; Default:)	Tunnel address or name of the pool from which the address is assigned to ppp interface locally
name (*string*; Default:)	PPP profile name
on-up (*script*; Default:)	Execute script on user login-event. These are available variables that are accessible for the event script: • user • local-address • remote-address • caller-id • called-id • interface
on-down (*script*; Default:)	Execute script on the user logging off. See ***on-up*** for more details
only-one (*yes* | *no* | *default*; Default: **default**)	Defines whether a user is allowed to have more than one ppp session at a time • yes - a user is not allowed to have more than one ppp session at a time • no - the user is allowed to have more than one ppp session at a time • default - derive this value from the interface default profile; same as no if this is the interface default profile
outgoing-filter (*string*; Default:)	Firewall chain name for outgoing packets. The specified chain gets control for each packet going to the client. The PPP chain should be manually added and rules with action=jump jump-target=ppp should be added to other relevant chains for this feature to work. For more information look at the Examples section.
parent-queue (*none* | *queue name*; Default:)	Specifies parent queue
queue-type (*default* | *ethernet-default* | *wireless-default* | *synchronous-default* | *hotspot-default* | *pcq-upload-default* | *pcq-download-default* | *only-hardware-queue* | *multi-queue-ethernet-default* | *default-small* | *custom queue type name* ; Default:)	Specifies queue type

rate-limit (*string*; Default:)	Rate limitation in form of **rx-rate[/tx-rate] [rx-burst-rate[/tx-burst-rate] [rx-burst-threshold[/tx-burst-threshold] [rx-burst-time[/tx-burst-time] [priority] [rx-rate-min[/tx-rate-min]]]]** from the point of view of the router (so "rx" is client upload, and "tx" is client download). All rates are measured in bits per second, unless followed by an optional 'k' suffix (kilobits per second) or 'M' suffix (megabits per second). If tx-rate is not specified, rx-rate serves as tx-rate too. The same applies to tx-burst-rate, tx-burst-threshold and tx-burst-time. If both rx-burst-threshold and tx-burst-threshold are not specified (but burst-rate is specified), rx-rate and tx-rate are used as burst thresholds. If both rx-burst-time and tx-burst-time are not specified, 1s is used as default. Priority takes values 1..8, where 1 implies the highest priority, but 8 - the lowest. If rx-rate-min and tx-rate-min are not specified rx-rate and tx-rate values are used. The rx-rate-min and tx-rate-min values can not exceed rx-rate and tx-rate values.
remote-address (*IP*; Default:)	Tunnel address or name of the pool from which address is assigned to remote ppp interface.
remote-ipv6-prefix-pool (*string* | *none*; Default: **none**)	Assign a prefix from the IPv6 pool to the client and install the corresponding IPv6 route.
session-timeout (*time*; Default:)	Maximum time the connection can stay up. By default, no time limit is set.
use-compression (*yes* | *no* | *default*; Default: **default**)	Specifies whether to use data compression or not. • yes - enable data compression • no - disable data compression • default - derive this value from the interface default profile; same as no if this is the interface default profile This setting does not affect OVPN tunnels.
use-encryption (*yes* | *no* | *default* | *require*; Default: **default**)	Specifies whether to use data encryption or not. • yes - enable data encryption • no - disable data encryption • default - derive this value from the interface default profile; same as no if this is the interface default profile • require - explicitly requires encryption This setting does not work on OVPN and SSTP tunnels.
use-ipv6 (*yes* | *no* | *default* | *require*; Default: **default**)	Specifies whether to allow IPv6. By default is enabled if IPv6 package is installed. • yes - enable IPv6 support • no - disable IPv6 support • default - derive this value from the interface default profile; same as no if this is the interface default profile • require - explicitly requires IPv6 support
use-mpls (*yes* | *no* | *default* | *require*; Default: **default**)	Specifies whether to allow MPLS over PPP. • yes - enable MPLS support • no - disable MPLS support • default - derive this value from the interface default profile; same as no if this is the interface default profile • require - explicitly requires MPLS support
use-upnp (*yes* | *no* | *default* ; Default: **default**)	Specifies whether to allow UPnP • yes - enable UPnP • no - disable UPnP • default - derive this value from the interface default profile; same as no if this is the interface default profile
wins-server (*IP address*; Default:)	IP address of the WINS server to supply to Windows clients

Notes

The two default profiles cannot be removed:

```
[admin@rb13] ppp profile> print
Flags: * - default
 0 * name="default" use-compression=no use-encryption=no only-one=no
     change-tcp-mss=yes
 1 * name="default-encryption" use-compression=default use-encryption=yes
     only-one=default change-tcp-mss=default
[admin@rb13] ppp profile>
```

incoming-filter and *outgoing-filter* arguments add dynamic jump rules to chain *ppp*, where the jump-target argument will be equal to *incoming-filter* or *outgoing-filter* argument in the profile. Therefore, chain *ppp* should be manually added before changing these arguments.

only-one parameter is ignored if RADIUS authentication is used.

User Database

Sub-menu: `/ppp secret`

PPP User Database stores PPP user access records with PPP user profile assigned to each user.

Properties

Property	Description
caller-id (*string*; Default:)	For PPTP and L2TP it is the IP address a client must connect from. For PPPoE it is the MAC address (written in CAPITAL letters) a client must connect from. For ISDN it is the caller's number (that may or may not be provided by the operator) the client may dial-in from
comment (*string*; Default:)	Short description of the user.
disabled (*yes* | *no*; Default: **no**)	Whether secret will be used.
limit-bytes-in (*integer*; Default: **0**)	The maximum amount of bytes for a session that the client can upload.
limit-bytes-out (*integer*; Default: **0**)	The maximum amount of bytes for a session that the client can download.
local-address (*IP address*; Default:)	IP address that will be set locally on ppp interface.
name (*string*; Default:)	Name used for authentication
password (*string*; Default:)	Password used for authentication
profile (*string*; Default: **default**)	Which user profile to use
remote-address (*IP*; Default:)	IP address that will be assigned to the remote ppp interface.
remote-ipv6-prefix (*IPv6 prefix*; Default:)	IPv6 prefix assigned to ppp client. Prefix is added to ND prefix list enabling stateless address auto-configuration on ppp interface.
routes (*string*; Default:)	Routes that appear on the server when the client is connected. The route format is: dst-address gateway metric (for example, 10.1.0.0/ 24 10.0.0.1 1). Other syntax is not acceptable since it can be represented incorrectly. Several routes may be specified and separated with commas. This parameter will be ignored for OpenVPN.
service (*any* | *async* | *isdn* | *l2tp* | *pppoe* | *pptp* | *ovpn* | *sstp*; Default: **any**)	Specifies the services that a particular user will be able to use.

Active Users

Sub-menu: `/ppp active`

This submenu allows monitoring active (connected) users.

`/ppp active print` command will show all currently connected users.

`/ppp active print stats` command will show received/sent bytes and packets

Properties

Property	Description
address (*IP address*)	The IP address the client got from the server
bytes (*integer*)	Amount of bytes transferred through this connection. The first figure represents the amount of transmitted traffic from the router's point of view, while the second one shows the amount of received traffic.
caller-id (*string*)	For PPTP and L2TP it is the IP address the client connected from. For PPPoE, it is the MAC address the client connected from.
encoding (*string*)	Shows encryption and encoding (separated with '/' if asymmetric) being used in this connection
limit-bytes-in (*integer*)	The maximum amount of bytes the user is allowed to send to the router.
limit-bytes-out (*integer*)	The maximum amount of bytes the user is allowed to send to the client.
name (*string*)	User name supplied at authentication stage
packets (*integer/integer*)	Amount of packets transferred through this connection. The first figure represents the amount of transmitted traffic from the router's point of view, while the second one shows the amount of received traffic
service (*async | isdn | l2tp | pppoe | pptp | ovpn | sstp*)	Type of service the user is using.
session-id (*string*)	Shows unique client identifier.
uptime (*time*)	User's uptime

Remote AAA

Sub-menu: `/ppp aaa`

Settings in this submenu allows to set RADIUS accounting and authentication. Note that the RADIUS user database is consulted only if the required username is not found in the local user database.

Properties

Property	Description
accounting (*yes | no*; Default: **yes**)	Enable RADIUS accounting
interim-update (*time*; Default: **0s**)	Interim-Update time interval
use-radius (*yes | no*; Default: **no**)	Enable user authentication via RADIUS. If an entry in the local secret database is not found, then the client will be authenticated via RADIUS.

enable-ipv6-accounting (*yes* | *no*; Default: **no**)	Enable IPv6 separate accounting. PPP service counts Layer2, IPv4 and IPv6 data all together when reporting network usage statistics to the RADIUS server by default. If it is required to differ IPv4 and IPv6 traffic, then this option can be enabled. Prerequisites for it to work are that Delegated-IPv6-Prefix must be provided for PPP user from the RADIUS and also rate-limit must be provided by the RADIUS as well. Dynamically created queue statistics will be used as counters for IPv6 data, which then will be included in accounting packets as separate IPv6 statistics attributes.

Examples

Add new profile

To add the profile ex that assigns the router itself the 10.0.0.1 address, and the addresses from the ex pool to the clients, filtering traffic coming from clients through mypppclients chain:

```
[admin@rb13] ppp profile> add name=ex local-address=10.0.0.1 remote-address=ex incoming-filter=mypppclients
[admin@rb13] ppp profile> print
Flags: * - default
 0 * name="default" use-compression=no use-vj-compression=no use-encryption=no only-one=no
     change-tcp-mss=yes
 1   name="ex" local-address=10.0.0.1 remote-address=ex use-compression=default
     use-vj-compression=default use-encryption=default only-one=default change-tcp-mss=default
     incoming-filter=mypppclients
 2 * name="default-encryption" use-compression=default use-vj-compression=default use-encryption=yes
     only-one=default change-tcp-mss=default
[admin@rb13] ppp profile>
```

Add new user

To add the user ex with password lkjrht and profile ex available for PPTP service only, enter the following command:

```
[admin@rb13] ppp secret> add name=ex password=lkjrht service=pptp profile=ex
[admin@rb13] ppp secret> print
Flags: X - disabled
 #   NAME                SERVICE CALLER-ID         PASSWORD          PROFILE            REMOTE-ADDRESS
 0   ex                  pptp                      lkjrht            ex                 0.0.0.0
[admin@rb13] ppp secret>
```

RADIUS

Summary

RADIUS, short for Remote Authentication Dial-In User Service, is a remote server that provides authentication and accounting facilities to various network appliances. RADIUS authentication and accounting allows the ISP or network administrator to manage PPP user access and accounting from one server throughout a large network. The MikroTik RouterOS has a RADIUS client that can authenticate for HotSpot, PPP, PPPoE, PPTP, L2TP, OVPN, and ISDN connections. The attributes received from the RADIUS server override the ones set in the default profile, but if some parameters are not received they are taken from the respective default profile.

The RADIUS server database is consulted only if no matching user access record is found in the router's local database.

If RADIUS accounting is enabled, accounting information is also sent to the RADIUS server default for that service.

RADIUS Client

Sub-menu: `/radius`

This sub-menu allows adding and removing RADIUS clients.

⚠ The order of added items in this list is significant.

Properties

Property	Description
accounting-backup (*yes* | *no*; Default: **no**)	Whether the configuration is for the backup RADIUS server
accounting-port (*integer [1..65535]*; Default: **1813**)	RADIUS server port used for accounting
address (*IPv4/IPv6 address*; Default: **0.0.0.0**)	IPv4 or IPv6 address of RADIUS server. The following formats are accepted: - *ipv4* - *ipv4@vrf* - *ipv6* - *ipv6@vrf*
authentication-port (*integer [1..65535]*; Default: **1812**)	RADIUS server port used for authentication.
called-id (*string*; Default:)	Value depends on Point-to-Point protocol: PPPoE - service name, PPTP - server's IP address, L2TP - server's IP address.
certificate (*string*; Default:)	Certificate file to use for communicating with RADIUS Server with RadSec enabled.
comment (*string*; Default:)	
disabled (*yes* | *no*; Default: **no**)	
domain (*string*; Default:)	Microsoft Windows domain of client passed to RADIUS servers that require domain validation.
protocol (*radsec* | *udp*; Default: **udp**)	Specifies the protocol to use when communicating with the RADIUS Server.
realm (*string*; Default:)	Explicitly stated realm (user domain), so the users do not have to provide proper ISP domain name in the user name.
secret (*string*; Default:)	The shared secret used to access the RADIUS server.

service (*ppp|login|hotspot|wireless|dhcp*; Default:)	Router services that will use this RADIUS server: • hotspot - HotSpot authentication service • login - router's local user authentication • ppp - Point-to-Point clients authentication • wireless - wireless client authentication • dhcp - DHCP protocol client authentication (client's MAC address is sent as User-Name)
src-address (*ipv4/ipv6 address*; Default: **0.0.0.0**)	Source IP/IPv6 address of the packets sent to the RADIUS server
timeout (*time*; Default: **100ms**)	Timeout after which the request should be resent, for example, "*/radius set timeout=300ms numbers=0*"

When the RADIUS server is authenticating the user with CHAP, MS-CHAPv1, MS-CHAPv2, it is not using a shared secret, the secret is used only in the authentication reply, and the router (RADIUS client) verifies it. So if you have the wrong shared secret, the RADIUS server will accept a request, but the router won't accept the reply. You can see that with "*/radius monitor*" command, the "bad-replies" number should increase whenever somebody tries to connect.

If RadSec is enabled, make sure your RADIUS Server is using "**radsec**" as the shared secret, otherwise, the RADIUS Server will not be able to decrypt data correctly (unprintable characters). With RadSec RouterOS forces the shared secret to "radsec" regardless of what has been set manually. For more details see - RFC6614.

Example

To set up a RADIUS Client for HotSpot and PPP services that will authenticate against a RADIUS Server (10.0.0.3), you need to do the following:

```
[admin@MikroTik] > /radius add service=hotspot,ppp address=10.0.0.3 secret=ex
[admin@MikroTik] > /radius print
Flags: X - disabled
# SERVICE CALLED-ID DOMAIN ADDRESS SECRET
0 ppp,hotspot
```

To set up a RADIUS Client with RadSec, you need to do the following:

```
[admin@MikroTik] > /radius add service=hotspot,ppp address=10.0.0.3 secret=radsec protocol=radsec
certificate=client.crt
[admin@MikroTik] > /radius print
Flags: X - disabled
# SERVICE CALLED-ID DOMAIN ADDRESS SECRET
0 ppp,hotspot 10.0.0.3 radsec
```

Make sure the specified certificate is trusted.

To view RADIUS Client statistics, you need to do the following:

```
[admin@MikroTik] > /radius monitor 0
pending: 0
requests: 10
accepts: 4
rejects: 1
resends: 15
timeouts: 5
bad-replies: 0
last-request-rtt: 0s
```

Make sure you enable RADIUS authentication for the desired services:

```
/ppp aaa set use-radius=yes
/ip hotspot profile set default use-radius=yes
```

Connection Terminating from RADIUS

Sub-menu: `/radius incoming`

This facility supports unsolicited messages sent from the RADIUS server. Unsolicited messages extend RADIUS protocol commands, that allow terminating a session that has already been connected from the RADIUS server. For this purpose, DM (Disconnect-Messages) is used. Disconnect messages cause a user session to be terminated immediately.

⚠ RouterOS doesn't support POD (Packet of Disconnect) the other RADIUS access request packet that performs a similar function as Disconnect Messages

Properties

Property	Description
accept (*yes* | *no*; Default: **no**)	Whether to accept unsolicited messages
port (*integer*; Default: **1700**)	The port number to listen for the requests on
vrf (*VRF name*; default value: **main**)	Set VRF on which service is listening for incoming connections

User

Summary

MikroTik RouterOS router user facility manages the users connecting the router from any of the Management tools. The users are authenticated using either a local database or a designated RADIUS server. Each user is assigned to a user group, which denotes the rights of this user. A group policy is a combination of individual policy items.

In case the user authentication is performed using RADIUS, the RADIUS client should be previously configured.

User Settings

The settings submenu allows to control the password complexity requirements of the router users.

Property	Description
minimum-password-length (*integer*; 0..4294967295; Default:)	Specifies the minimum character length of the user password
minimum-categories (*integer*; 0..4; Default:)	Specifies the complexity requirements of the password, with categories being *uppercase, lowercase, digit, symbol.*

User Groups

The router user groups provide a convenient way to assign different permissions and access rights to different user classes.

Properties

Property	Description
name (*string*; Default:)	The name of the user group

policy (*local | telnet | ssh | ftp | reboot | read | write | policy | test | winbox | password | web | sniff | sensitive | api | romon | dude | tikapp*; Default: **none**)	List of allowed policies: Login policies: • local - policy that grants rights to log in locally via console • telnet - policy that grants rights to log in remotely via telnet • ssh - policy that grants rights to log in remotely via secure shell protocol • web - policy that grants rights to log in remotely via WebFig. • winbox - policy that grants rights to log in remotely via WinBox and bandwidth test authentication • password - policy that grants rights to change the password • api - grants rights to access router via API. • rest-api - grants rights to access the router via REST API. • ftp - policy that grants full rights to log in remotely via FTP. Allows to read/write /erase files and to transfer files from/to the router. Should be used together with read/write policies. • romon - policy that grants rights to connect to the RoMon server. Config Policies: • reboot - policy that allows rebooting the router • read - policy that grants read access to the router's configuration. All console commands that do not alter the router's configuration are allowed. Doesn't affect FTP • write - policy that grants write access to the router's configuration, except for user management. This policy does not allow to read the configuration, so make sure to enable read policy as well • policy - policy that grants user management rights. Should be used together with the write policy. Allows also to see global variables created by other users (requires also 'test' policy). • test - policy that grants rights to run ping, traceroute, bandwidth-test, wireless scan, snooper, fetch, email and other test commands • sensitive - grants rights to change "hide sensitive" option, if this policy is disabled sensitive information is not displayed. • sniff - policy that grants rights to use packet sniffer tool.
skin (*name*; Default: **default**)	Used skin for WebFig

Default groups

There are three default system groups which cannot be deleted:

```
[admin@MikroTik] > /user group print
0 name="read" policy=local,telnet,ssh,reboot,read,test,winbox,password,web,sniff,sensitive,api,romon,tikapp,!
ftp,!write,!policy,!dude skin=default

1 name="write" policy=local,telnet,ssh,reboot,read,write,test,winbox,password,web,sniff,sensitive,api,romon,
tikapp,!ftp,!policy,!dude skin=default

2 name="full" policy=local,telnet,ssh,ftp,reboot,read,write,policy,test,winbox,password,web,sniff,sensitive,api,
romon,tikapp,!dude skin=default
```

Please note, that even the "*read*" group includes *sensitive*, *reboot,* and other important policies, meaning that this group should not be given to untrusted users. For truly limited groups, make a custom group, defining specific policies. All groups have access to file operations. Exclamation sign '!' just before the policy item name means NOT.

Router Users

The router user database stores information such as username, password, allowed access addresses, and group about router management personnel.

Properties

Property	Description
address (*IP/mask | IPv6 prefix*; Default:)	Host or network address from which the user is allowed to log in
group (*string*; Default:)	Name of the group the user belongs to
name (*string*; Default:)	User name. Although it must start with an alphanumeric character, it may contain "*", "_", ".", and "@" symbols.
password (*string*; Default:)	User password. If not specified, it is left blank (hit [Enter] when logging in). It conforms to standard Unix characteristics of passwords and may contain letters, digits, "*" and "_" symbols.
last-logged-in (*time and date*; Default: **""**)	Read-only field. Last time and date when a user logged in.

Notes

There is one predefined user with full access rights:

```
[admin@MikroTik] user> print
Flags: X - disabled
# NAME GROUP ADDRESS LAST-LOGGED-IN
0 ;;; system default user
admin full 0.0.0.0/0 dec/08/2010 16:19:24
```

There always should be at least one user with full access rights. If the user with full access rights is the only one, it cannot be removed.

Monitoring Active Users

```
/user active print
```

The command shows the currently active users along with respective statistics information.

Properties

All properties are read-only.

Property	Description
address (*IP/IPv6 address*)	Host IP/IPv6 address from which the user is accessing the router. 0.0.0.0 means that the user is logged in locally
group (*string*)	A group that the user belongs to.
name (*string*)	User name.
radius (*true | false*)	Whether a user is authenticated by the RADIUS server.
via (*local | telnet | ssh |winbox | api | web | tikapp | ftp | dude*)	User's access method
when (*time*)	Time and date when the user logged in.

Remote AAA

Router user remote AAA enables router user authentication and accounting via a RADIUS server. The RADIUS user database is consulted only if the required username is not found in the local user database.

Properties

Property	Description
accounting (*yes* | *no*; Default: **yes**)	
exclude-groups (*list of group names*; Default:)	Exclude-groups consist of the groups that should not be allowed to be used for users authenticated by radius. If the radius server provides a group specified in this list, the default-group will be used instead. This is to protect against privilege escalation when one user (without policy permission) can change the radius server list, set up its own radius server and log in as admin.
default-group (*string*; Default: **read**)	User group used by default for users authenticated via a RADIUS server.
interim-update (*time*; Default: **0s**)	Interim-Update time interval
use-radius (*yes* |*no*; Default: **no**)	Enable user authentication via RADIUS

If you are using RADIUS, you need to have CHAP support enabled in the RADIUS server for WinBox to work

SSH Keys

This menu allows importing of private and public keys used for SSH authentication.

By default, User is not allowed to log in via SSH by password if an SSH key for the user is added. For more details see the SSH page.

Public keys

This menu is used to import and list imported public keys. Public keys are used to approve another device's identity when logging into a router using an SSH key.

On public key import, is it possible to specify key-owner.

RSA and Ed25519 keys are supported in PEM, PKCS#8, or OpenSSH format.

Property	Description
user (*string*; Default:)	username to which the SSH key is assigned.
key-owner (*string*)	SSH key owner
public-key-file (*string*)	file name in the router's root directory containing the public key.

Private keys

This menu is used to import and list imported private keys. Private keys are used to approve the router's identity during login into another device using an SSH key.

On private key import, is it possible to specify key-owner.

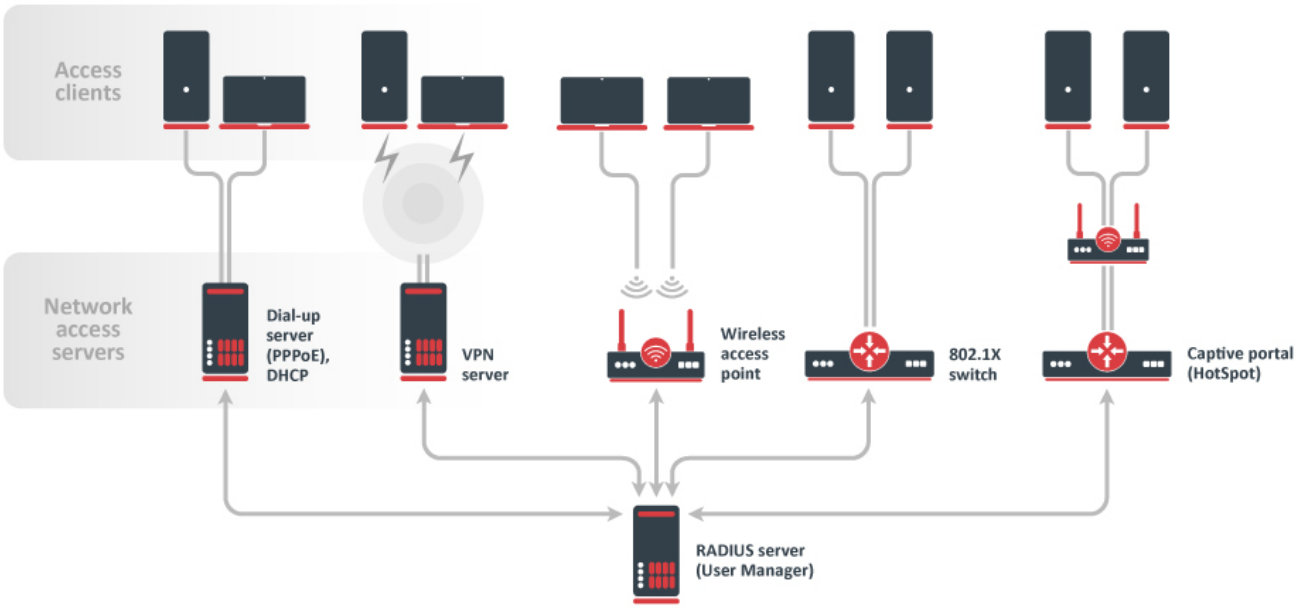
RSA keys are supported in PEM or PKCS#8 format.

Property	Description
user (*string*; Default:)	username to which the SSH key is assigned.
key-owner (*string*)	SSH key owner
private-key-file (*string*)	file name in the router's root directory containing the private key.
passphrase *(string)*	key file passphrase

User Manager

Overview

User Manager is RADIUS server implementation in RouterOS which provides centralized user authentication and authorization to a certain service. Having a central user database allows better tracking of system users and customers. As a separate package, User Manager is available on all architectures except SMIPS, however, care must be taken due to limited free space available. It supports many different authentication methods including PAP, CHAP, MS-CHAP, MS-CHAPv2, EAP-TLS, EAP-TTLS, and EAP-PEAP. In RouterOS, DHCP, Dot1x, Hotspot, IPsec, PPP, and Wireless are features that benefit from User Manager the most. Each user can see their account statistics and manage available profiles using the WEB interface. Additionally, users can buy their own data plans (profiles) using the most popular payment gateway - PayPal making it a great system for service providers. Customized reports can be generated to ease processing by the billing department. User Manager works according to RADIUS standards defined in RFC2865 and RFC3579.



Attributes

Sub-menu: `/user-manager attribute`

RADIUS attributes are defined authorization, information, and configuration parameters that are passed between the RADIUS server and the client. User Manager allows sending customized attributes defined in the "attributes" menu. RouterOS has a set of predefined attributes already present, but it is also possible to add additional attributes if necessary. Predefined attributes:

Attribute	Vendor ID	Type ID	Value type	Packet type	Description
Framed-IP-Address	0 (standard)	8	ip address	Access-Accept	RFC2865 section 5.8
Framed-IP-Netmask	0 (standard)	9	ip address	Access-Accept	RFC2865 section 5.9
Session-Timeout	0 (standard)	27	integer	Access-Accept, Access-Challenge	RFC2865 section 5.27
Idle-Timeout	0 (standard)	28	integer	Access-Accept, Access-Challenge	RFC2865 section 5.28

<table>
<tr>
<td>Tunnel-Type</td>
<td>0 (standard)</td>
<td>64</td>
<td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Point-to-Point Tunneling Protocol (PPTP)</td></tr>
<tr><td>2</td><td>Layer Two Forwarding (L2F)</td></tr>
<tr><td>3</td><td>Layer Two Tunneling Protocol (L2TP)</td></tr>
<tr><td>4</td><td>Ascend Tunnel Management Protocol (ATMP)</td></tr>
<tr><td>5</td><td>Virtual Tunneling Protocol (VTP)</td></tr>
<tr><td>6</td><td>IP Authentication Header in the Tunnel-mode (AH)</td></tr>
<tr><td>7</td><td>IP-in-IP Encapsulation (IP-IP)</td></tr>
<tr><td>8</td><td>Minimal IP-in-IP Encapsulation (MIN-IP-IP)</td></tr>
<tr><td>9</td><td>IP Encapsulating Security Payload in the Tunnel-mode (ESP)</td></tr>
<tr><td>10</td><td>Generic Route Encapsulation (GRE)</td></tr>
<tr><td>11</td><td>Bay Dial Virtual Services (DVS)</td></tr>
<tr><td>12</td><td>IP-in-IP Tunneling</td></tr>
</table>
</td>
<td>Access-Accept</td>
<td>RFC2868 section 3.1</td>
</tr>
</table>

<table>
<tr><td>Tunnel-Medium-Type</td><td>0 (standard)</td><td>65</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>IPv4 (IP version 4)</td></tr><tr><td>2</td><td>IPv6 (IP version 6)</td></tr><tr><td>3</td><td>NSAP</td></tr><tr><td>4</td><td>HDLC (8-bit multidrop)</td></tr><tr><td>5</td><td>BBN 1822</td></tr><tr><td>6</td><td>802 (includes all 802 media plus Ethernet "canonical format")</td></tr><tr><td>7</td><td>E.163 (POTS)</td></tr><tr><td>8</td><td>E.164 (SMDS, Frame Relay, ATM)</td></tr><tr><td>9</td><td>F.69 (Telex)</td></tr><tr><td>10</td><td>X.121 (X.25, Frame Relay)</td></tr><tr><td>11</td><td>IPX</td></tr><tr><td>12</td><td>Appletalk</td></tr><tr><td>13</td><td>Decnet IV</td></tr><tr><td>14</td><td>Banyan Vines</td></tr><tr><td>15</td><td>E.164 with NSAP format subaddress</td></tr></table></td><td>Access-Accept</td><td>RFC2868 section 3.2</td></tr>
<tr><td>Tunnel-Private-Group-ID</td><td>0 (standard)</td><td>81</td><td>string</td><td>Access-Accept</td><td>RFC2868 section 3.6</td></tr>
<tr><td>Framed-Pool</td><td>0 (standard)</td><td>88</td><td>string</td><td>Access-Accept</td><td>RFC2869 section 5.18</td></tr>
<tr><td>Framed-IPv6-Prefix</td><td>0 (standard)</td><td>97</td><td>ipv6 prefix</td><td>Access-Accept</td><td>RFC3162 section 2.3</td></tr>
<tr><td>Framed-IPv6-Pool</td><td>0 (standard)</td><td>100</td><td>string</td><td>Access-Accept</td><td>RFC3162 section 2.6</td></tr>
<tr><td>Delegated-IPv6-Prefix</td><td>0 (standard)</td><td>123</td><td>ipv6 prefix</td><td>Access-Accept</td><td>RFC4818</td></tr>
<tr><td>Framed-IPv6-Address</td><td>0 (standard)</td><td>168</td><td>ip address</td><td>Access-Accept</td><td>RFC6911 section 3.1</td></tr>
<tr><td>Mikrotik-Recv-Limit</td><td>14988 (Mikrotik)</td><td>1</td><td>integer</td><td>Access-Accept</td><td>Total receive limit in bytes for the client.</td></tr>
<tr><td>Mikrotik-Xmit-Limit</td><td>14988 (Mikrotik)</td><td>2</td><td>integer</td><td>Access-Accept</td><td>Total transmit limit in bytes for the client.</td></tr>
<tr><td>Mikrotik-Group</td><td>14988 (Mikrotik)</td><td>3</td><td>string</td><td>Access-Accept</td><td>User's group for local users.

HotSpot profile for HotSpot users.

PPP profile for PPP users.</td></tr>
<tr><td>Mikrotik-Wireless-Forward</td><td>14988 (Mikrotik)</td><td>4</td><td>integer</td><td>Access-Accept</td><td>Not forward the client's frames back to the wireless infrastructure if this attribute is set to "0" (wireless only).</td></tr>
<tr><td>Mikrotik-Wireless-Skip-Dot1x</td><td>14988 (Mikrotik)</td><td>5</td><td>integer</td><td>Access-Accept</td><td>Disable 802.1x authentication for the particular wireless client if set to a non-zero value (wireless only).</td></tr>
</table>

Mikrotik-Wireless-Enc-Algo	14988 (Mikrotik)	6	<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>No-encryption</td></tr><tr><td>1</td><td>40-bit-WEP</td></tr><tr><td>2</td><td>104-bit-WEP</td></tr><tr><td>3</td><td>AES-CCM</td></tr><tr><td>4</td><td>TKIP</td></tr></table>	Access-Accept	WEP encryption algorithm(wireless only).
Mikrotik-Wireless-Enc-Key	14988 (Mikrotik)	7	string	Access-Accept	WEP encryption key for the client (wireless only).
Mikrotik-Rate-Limit	14988 (Mikrotik)	8	string	Access-Accept	Datarate limitation for clients. The format is: rx-rate[/tx-rate] [rx-burst-rate[/tx-burst-rate] [rx-burst-threshold[/tx-burst-threshold] [rx-burst-time[/tx-burst-time] [priority] [rx-rate-min[/tx-rate-min]]]] from the point of view of the router (so "rx" is client upload, and "tx" is client download). All rates should be numbers with optional 'k' (1,000s) or 'M' (1,000,000s). If the tx-rate is not specified, the rx-rate is as tx-rate too. The same goes for tx-burst-rate and tx-burst-threshold and tx-burst-time. If both rx-burst-threshold and tx-burst-threshold are not specified (but burst-rate is specified), rx-rate and tx-rate are used as burst thresholds. If both rx-burst-time and tx-burst-time are not specified, 1s is used as default. Priority takes values 1..8, where 1 implies the highest priority, but 8 - the lowest. If rx-rate-min and tx-rate-min are not specified rx-rate and tx-rate values are used. The rx-rate-min and tx-rate-min values can not exceed rx-rate and tx-rate values.
Mikrotik-Realm	14988 (Mikrotik)	9	string	Access-Request	If it is set in /radius menu, it is included in every RADIUS request as Mikrotik-Realm attribute. If it is not set, the same value is sent as in the MS-CHAP-Domain attribute (if MS-CHAP-Domain is missing, Realm is not included either).
Mikrotik-Host-IP	14988 (Mikrotik)	10	ip address	Access-Request	The IP address of HotSpot client before Universal Client translation (the original IP address of the client).
Mikrotik-Mark-Id	14988 (Mikrotik)	11	string	Access-Accept	Firewall mangle chain name (HotSpot only). The MikroTik RADIUS client upon receiving this attribute creates a dynamic firewall mangle rule with action=jump chain=hotspot and jump-target equal to the attribute value. Mangle chain name can have suffixes .in or .out, which will install rule only for incoming or outgoing traffic. Multiple Mark-id attributes can be provided, but only the last ones for incoming and outgoing are used.
Mikrotik-Advertise-URL	14988 (Mikrotik)	12	string	Access-Accept	URL of the page with advertisements that should be displayed to clients. If this attribute is specified, advertisements are enabled automatically, including transparent proxy, even if they were explicitly disabled in the corresponding user profile. Multiple attribute instances may be sent by the RADIUS server to specify additional URLs which are chosen in a round-robin fashion.
Mikrotik-Advertise-Interval	14988 (Mikrotik)	13	integer	Access-Accept	The time interval between two adjacent advertisements. Multiple attribute instances may be sent by the RADIUS server to specify additional intervals. All interval values are treated as a list and are taken one by one for each successful advertisement. If the end of the list is reached, the last value is continued to be used.
Mikrotik-Recv-Limit-Gigawords	14988 (Mikrotik)	14	integer	Access-Accept	4G (2^32) bytes of total receive limit (bits 32..63, when bits 0..31 are delivered in Mikrotik-Recv-Limit).
Mikrotik-Xmit-Limit-Gigawords	14988 (Mikrotik)	15	integer	Access-Accept	4G (2^32) bytes of total transmit limit (bits 32..63, when bits 0..31 are delivered in Mikrotik-Recv-Limit).
Mikrotik-Wireless-PSK	14988 (Mikrotik)	16	string	Access-Accept	
Mikrotik-Total-Limit	14988 (Mikrotik)	17	integer	Access-Accept	
Mikrotik-Total-Limit-Gigawords	14988 (Mikrotik)	18	integer	Access-Accept	
Mikrotik-Address-List	14988 (Mikrotik)	19	string	Access-Accept	
Mikrotik-Wireless-MPKey	14988 (Mikrotik)	20	string	Access-Accept	
Mikrotik-Wireless-Comment	14988 (Mikrotik)	21	string	Access-Accept	
Mikrotik-Delegated-IPv6-Pool	14988 (Mikrotik)	22	string	Access-Accept	IPv6 pool used for Prefix Delegation.

Mikrotik-DHCP-Option-Set	14988 (Mikrotik)	23	string	Access-Accept	
Mikrotik-DHCP-Option-Param-STR1	14988 (Mikrotik)	24	string	Access-Accept	
Mikrotik-DHCP-Option-Param-STR2	14988 (Mikrotik)	25	string	Access-Accept	
Mikrotik-Wireless-VLANID	14988 (Mikrotik)	26	integer	Access-Accept	VLAN ID for the client (Wireless only).
Mikrotik-Wireless-VLANIDtype	14988 (Mikrotik)	27	Value: 0 — 802.1q; 1 — 802.1ad	Access-Accept	VLAN ID type for the client (Wireless only).
Mikrotik-Wireless-Minsignal	14988 (Mikrotik)	28	string	Access-Accept	
Mikrotik-Wireless-Maxsignal	14988 (Mikrotik)	29	string	Access-Accept	
Mikrotik-Switching-Filter	14988 (Mikrotik)	30	string	Access-Accept	Allows to create dynamic switch rules when authenticating clients with dot1x server.

Properties

Property	Description
name (*string*; Default:)	Name of the attribute.
packet-types (*string*; Default: **access-accept**)	• access-accept - use this attribute in RADIUS Access-Accept messages • access-challenge - use this attribute in RADIUS Access-Challenge messages
type-id (*integer:1..255*; Default:)	Attribute identification number from the specific vendor's attribute database.
value-type (*string*; Default:)	• hex • ip-address - IPv4 or IPv6 IP address • ip6-prefix - IPv6 prefix • macro • string • uint32
vendor-id (*integer*; Default: **0**)	IANA allocated a specific enterprise identification number.

Database

Sub-menu: `/user-manager database`

All RADIUS-related information is stored in a separate User Manager's database configurable under the "database" sub-menu. "Enabled" and "db-path" are the only parameters that are not stored in the User Manager's database and instead are stored in the main RouterOS configuration table meaning that these parameters will be affected by the RouterOS configuration reset. The rest of the configuration, session, and payment data is stored in a separate SQLite database on the FLASH storage of the device. When performing any actions with databases, it is advised to make a backup before and after any activity.

Properties

Property	Description
db-path (*string*; Default:)	Path to the location where database files will be stored.

Read-only properties

Property	Description
db-size	The current size of the database.
free-disk-space	Free space left on the disk where the database is stored.

Commands

Property	Description
load (*name*)	Restore previously created backup file in .umb format.
migrate-legacy-db (*database-path; overwrite*)	Convert the old User Manager (from RouterOS v6 or before) to the new standard. It is possible to overwrite the current database.
optimize-db ()	
save (name; overwrite)	Save the current state of the User Manager database.

Limitations

Sub-menu: `/user-manager limitation`

Limitations are used by Profiles and are linked together by Profile-Limitations. RADIUS accounting and Interim updates must be enabled to seamlessly switch between multiple limitations or disconnect active sessions when *download-limit*, *upload-limit* or *uptime-limit* is reached.

To disconnect already active sessions from User Manager, *accept* must be set to *yes* on the RADIUS client side. If simultaneous session limits are not unlimited (shared-users) and it has reached the maximum allowed number, then the router will try to disconnect the older user session first.

User-Manager attempts to disconnect an active session before a new user will be accepted (when the appropriate limit is set), that's why in such setups it is suggested to use 1s for /radius client timeout.

IPsec service in RouterOS does not support rate limitations.

Properties

Property	Description
comment (*string*; Default:)	Short description of the limitation.
download-limit (*integer*; Default: **0**)	The total amount of traffic a user can download in Bytes.
name (*string*; Default:)	Unique name of the limitation.
rate-limit-burst-rx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-burst-threshold-rx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-burst-threshold-tx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-burst-time-rx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-burst-time-tx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-burst-tx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.

rate-limit-min-rx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-min-tx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-priority ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-rx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
rate-limit-tx ()	Part of *MT-Rate-Limit* RADIUS attribute. Refer to Queues#SimpleQueue.
reset-counters-interval (*hourly* | *daily* | *weekly* | *monthly* | *disabled*); Default: **disabled**)	The interval from *reset-counters-start-time* when all associated user statistics are cleared.
reset-counters-start-time (*datetime*; Default:)	Static date and time value from which *reset-counters-interval* is calculated.
transfer-limit (*integer*; Default: **0**)	The total amount of aggregated (download+upload) traffic in Bytes.
upload-limit (*integer*; Default: **0**)	The total amount of traffic a user can upload in Bytes.
uptime-limit (*time*; Default: **00:00:00**)	The total amount of uptime a user can stay active.

Payments

Sub-menu: `/user-manager payment`

Information about all received payments is available in this section.

Read-only properties

Property	Description
currency (*string*)	The currency used in the transaction.
method (*string*)	Service used for the transaction (currently PayPal only).
price (*decimal*)	Amount paid by the user.
profile (*profile*)	Name of the profile the user purchased.
trans-end (*datetime*)	Date and time when the transaction started.
trans-start (*datetime*)	Date and time when the transaction ended.
trans-status (*string*)	Status of the transaction. Possible statuses - *started*, *pending*, *approved*, *declined*, *error*, *timeout*, *aborted*, *user approved*. Only *approved* should be considered as a complete transaction.
user (*string*; Default:)	Name of the user who performed the transaction.
user-message (*string*; Default:)	

Profiles

Sub-menu: `/user-manager profile`

Properties

Property	Description
comment (*string*; Default:)	Short description of the entry.
name (*string*; Default:)	Unique name of the profile.
name-for-users (*string*; Default:)	Name of the profile that will be shown for users on the Web page.

override-shared-users (*decimal | off | unlimited*; Default: **off**)	Whether to allow multiple sessions with the same user name. This overrides the *shared-users* setting.
price (*decimal*; Default: **0.00**)	
starts-when (*assigned | first-auth*; Default: **assigned**)	The time when does the profile become active. *Assigned* - immediately when a User Profile entry is created. *First-auth* - upon first authentication request from the user.
validity (*time | unlimited*; Default: **unlimited**)	The total amount of time a user can use this profile.

Profile Limitations

Sub-menu: `/user-manager profile-limitation`

Profile-Limitations table links Limitations and Profiles together and defines their validity period. When multiple Limitations are assigned to the same Profile, a user must comply with all Limitations for the session to be established. This allows more complicated setups to be created, for example, separate monthly and daily bandwidth limits.

Properties

Property	Description
comment (*string*; Default:)	Short description of the entry.
from-time (*time*; Default: **00:00:00**)	Time of day when the limitation should start.
limitation (*limitation*; Default:)	Name of already created **Limitation**.
profile (*profile*; Default:)	Name of already created **Profile**.
till-time (*time*; Default: **23:59:59**)	Time of day when the limitation should end.
weekdays (*day of week*; Default: **Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday**)	Day of the week when the limitation should be active.

Routers

Sub-menu: `/user-manager router`

Here you can define NAS devices that can use User Manager as a RADIUS server.

Properties

Property	Description
coa-port (*integer:1..65535*; Default: **3799**)	Port number of CoA (Change of Authorization) communication.
address (*IP/IPv6*; Default:)	IP address of the RADIUS client.
comment (*string*; Default:)	Short description of the NAS.
disabled (*yes | no*; Default: **no**)	Controls whether the entry is currently active or not.
name (*string*; Default:)	Unique name of the RADIUS client.
shared-secret (*string*; Default:)	Used to secure communication between a RADIUS server and a RADIUS client.

Commands

Property	Description
reset-counters ()	Clear all statistics for specific RADIUS client.

Sessions

Sub-menu: `/user-manager session`

Sessions are logged only if accounting is enabled on NAS.

Read-only properties

Property	Description
acct-session-id (*string*)	Unique identification of the accounting session.
active (*yes* | *no*)	Whether the session is currently used.
calling-station-id (*string*)	User's identifier, usually IP address or MAC address.
download (*Bytes*)	Amount of traffic downloaded.
ended (*datetime*)	Date and time when the session was closed. Empty for active sessions.
last-accounting-packet (*datetime*)	Date and time when the last accounting update was received.
nas-ip-address (*IP address*)	The IP address of the NAS.
nas-port-id (*string*)	Identifier of the NAS port that is authenticating the user.
nas-port-type (*string*)	The port type (*physical* or *virtual*) that is authenticating the user.
started (*datetime*)	Date and time when the session was established.
status (*list of statuses*)	Possible available statuses of a session: *start* - accounting message *Start* has been received, *stop* - accounting message *Stop* has been received, *interim* - *Interim update* has been received, *close-acked* - session is successfully closed, *expired*.
terminate-cause (*string*)	The reason why the session was closed.
upload (*Bytes*)	Amount of traffic uploaded.
uptime (*time*)	Total logged uptime on the session.
user (*string*)	Name of the user.
user-address (*IP address*)	IP address provided to the user.

Settings

Sub-menu: `/user-manager`

Properties

Property	Description
accounting-port (*integer*; Default: **1813**)	Port to listen for RADIUS accounting requests.
authentication-port (*integer*; Default: **1812**)	Port to listen for RADIUS authentication requests.
certificate (*certificate*; Default:)	Certificate for use in EAP TLS-type authentication methods.

enabled (*yes* | *no*; Default: **no**)	Whether the User Manager functionality is enabled.
use-profiles (*yes* | *no*; Default: **no**)	Whether to use **Profiles** and **Limitations**. When set to *no,* only **User** configuration is required to run User Manager.

Advanced

Sub-menu: `/user-manager advanced`

Properties

Property	Description
paypal-allow (*yes* | *no*; Default: **no**)	Whether to enable PayPal functionality for User Manager.
paypal-currency (*string*; Default: **USD**)	The currency related to *price* setting in which users will be billed.
paypal-password (*string*; Default:)	The password of your PayPal API account.
paypal-signature (*string*; Default:)	Signature of your PayPal API account.
paypal-use-sandbox (*yes* | *no*; Default: **no**)	Whether to use PayPal's sandbox environment for testing purposes.
paypal-user (*string*; Default:)	Username of your PayPal API account.
web-private-password (*string*; Default:)	Password for accessing */um/PRIVATE/* section over HTTP.
web-private-username (*string*; Default:)	Username for accessing */um/PRIVATE/* section over HTTP.

Users

Sub-menu: `/user-manager user`

Properties

Property	Description
attributes (*array of attributes*; Default:)	Custom set of **Attributes** with their values that will additionally be added to Access-Accept messages.
caller-id (*string*; Default:)	Allow user's authentication with a specific *Calling-Station-Id* value.
comment (*string*; Default:)	Short description of the user.
disabled (*yes* | *no*; Default: **no**)	Controls whether the user can be used or not.
group (*group*; Default: **default**)	Name of the **Group** the user is associated to.
name (*string*; Default:)	Username for session authentication.
otp-secret (*string*; Default:)	A one-time password token that is attached to the password.
password (*string*; Default:)	The password of the user for session authentication.
shared-users (*integer* | *unlimited*; Default: **1**)	The total amount of sessions the user can simultaneously establish.

Commands

Property	Description
add-batch-users ()	The command can generate multiple user accounts based on various parameters.
generate-voucher ()	Generates a file based on *voucher-template* that can be presented to the end user.

monitor ()	Shows total statistics for a user. Stats include *total-uptime*, *total-download*, *total-upload*, *active-sessions*, *actual-profile*, *attributes-details*.

User Groups

Sub-menu: `/user-manager user group`

User groups define common characteristics of multiple users such as allowed authentication methods and RADIUS attributes. There are two groups already present in User Manager called *default* and *default-anonymous*.

Properties

Property	Description
attributes (*array of attributes*; Default:)	Custom set of **Attributes** with their values that will additionally be added to Access-Accept messages for users in this group.
comment (*string*; Default:)	Short description of the group.
inner-auths (*list of auths*; Default:)	List of allowed authentication methods for tunneled (outer) authentication methods. Supported inner authentication methods - *ttls-pap*, *ttls-chap*, *ttls-mschap1*, *ttls-mschap2*, *peap-mschap2*.
name (*string*; Default:)	Unique name of the group.
outer-auths (*list of auths*; Default:)	List of allowed authentication methods. Supported outer authentication methods - *pap*, *chap*, *mschap1*, *mschap2*, *eap-tls*, *eap-ttls*, *eap-peap*, *eap-mschap2*.

User Profiles

Sub-menu: `/user-manager user-profile`

This menu assigns users a profile and tracks the status of the profile. A single user can have multiple profiles assigned, however, only one can be used at the same time. A user will seamlessly be switched to the next profile when the currently active profile expires without dropping the user's session.

Properties

Property	Description
profile (*profile*; Default:)	Name of the profile to assign for user.
user (*user*; Default:)	Name of the user to use particular profile.

Read-only properties

Property	Description
end-time (*datetime*)	Date and time the **User Profile** will expire.
state (*running active* | running | *used*)	The current state of the **User Profile**. *Running active* - currently used profile by the user. *Running* - a profile is ready to be used. *Used* - an expired profile that can no longer be activated.

Commands

Property	Description
activate-user-profile ()	Make a **User Profile** entry active immediately.

WEB Interface

Each user has access to his personal profile using a WEB interface. The WEB interface can be accessed by adding "/um/" directory to the router's IP or domain, for example, http://example.com/um/. Note that the WEB interface is affected by IP Services "www" and "www-ssl". The WEB interface can be customized using CSS, JavaScript, and HTML.

Customizable file reference

File	Description
css/login.css	Cascading style sheet file used in login prompt page.
css/user.css	Cascading style sheet file used in user's profile page.
img/PayPal_mark_37x23.gif	PayPal logo image.
img/ajax-loader.gif	Loading gif while processing page switching.
img/mikrotik_logo.png	MikroTik logo that is displayed on all pages.
js/generic.js	Javascript file used on all pages.
js/login.js	Javascript file used in login prompt page.
js/user.js	Javascript file used in user's profile page.
user/login_dynamic.html	Layout of the login prompt page.
user/user_dynamic.html	Layout of the user's profile page.

Application Guides

Batch user creation

It is possible to create multiple new users with randomly generated usernames and passwords. For example, the following command will generate 3 new users with 6 lowercase symbols as the username and 6 lowercase, uppercase, and numbers as the password.

```
/user-manager user
add-batch-users number-of-users=3 password-characters=lowercase,numbers,uppercase password-length=6 username-
characters=lowercase username-length=6
```

The command generated users can be seen by printing the user's table:

```
/user-manager user print
Flags: X - disabled
 0   name="olsgkl" password="86a6zH" otp-secret="" group=default shared-users=1 attributes=""

 1   name="lkbwss" password="jaKY5V" otp-secret="" group=default shared-users=1 attributes=""

 2   name="cwxbwu" password="a62yZd" otp-secret="" group=default shared-users=1 attributes=""
```

Providing NAS with custom RADIUS attributes

It is possible to send additional RADIUS attributes during the authentication process to provide NAS with custom information about the session, such as what IP address should be assigned to the supplicant or what address pool to use for address assigning.

Static IP address for a user

To assign the end user a static IP address, *Framed-IP-Address* attribute can be used. When using static IP address allocation, *shared-sessions* must be set to 1 to prevent cases when a user has multiple simultaneous sessions, but there is only one IP address. For example:

```
/user-manager user
set [find name=username] shared-users=1 attributes=Framed-IP-Address:192.168.1.4
```

Specifying address pool for a group of users

We can group up multiple similar users and assign RADIUS attributes to all of them at once. First of all, create a new group:

```
/user-manager user group
add name=location1 outer-auths=chap,eap-mschap2,eap-peap,eap-tls,eap-ttls,mschap1,mschap2,pap \
inner-auths=peap-mschap2,ttls-chap,ttls-mschap1,ttls-mschap2,ttls-pap attributes=Framed-Pool:pool1
```

The next step is to assign a user to the group:

```
/user-manager user
 set [find name=username] group=location1
```

In this case, an IP address from *pool1* will be assigned to the user upon authentication - make sure *pool1* is created on the NAS device.

Using TOTP (time-based one-time password) for user authentication

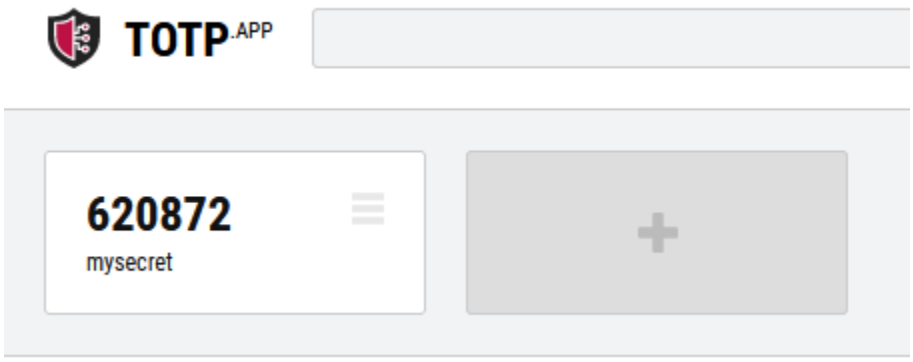
User Manager supports time-based authentication token addition to the user's password field that is regenerated every 30 seconds.

> OTP depends on the clock, so make sure time settings are configured correctly.

TOTP works by having a shared secret on the supplicant (client) and the authentication server (User Manager). To configure TOTP on RouterOS, simply set the *otp-secret* for the user. For example:

```
/user-manager user
set [find name=username] password=mypass otp-secret=mysecret
```

To calculate the TOTP token on the supplicant side, many widely available applications can be used, for example, Google Authenticator or https://totp.app/. Adding mysecret to the TOTP token generator will provide a new unique 6-digit code that must be added to the user password.



The following example will accept the user's authentication with a calculated TOTP token added to the common password until a new TOTP token is generated, for example,

```
User-Name=username
User-Password=mypass620872
```

Exporting user credentials

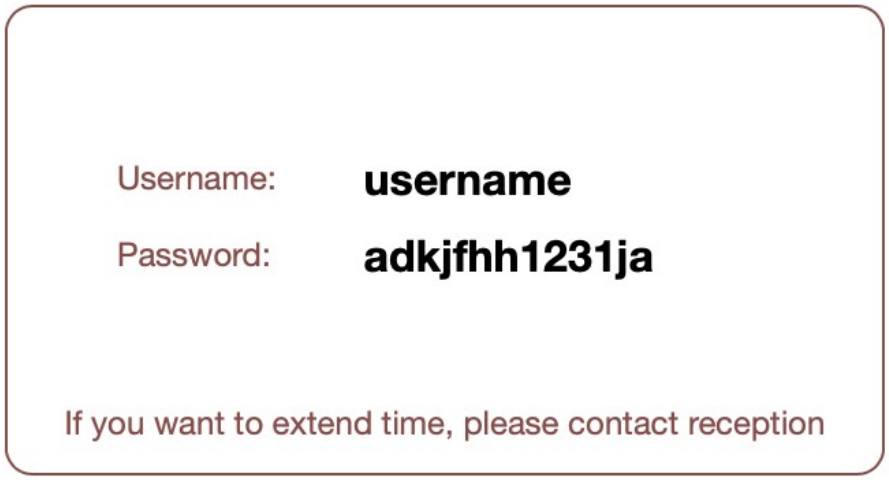
Printable login credentials for a single user

To generate a single user's printable voucher card, simply use the *generate-voucher* command. Specify the RouterOS ID number of the user or use the *find* command to specify a username. A template is already included in User Manager's installation available in the Files section of your device. You can customize the template for your needs.

```
/user-manager user
generate-voucher voucher-template=printable_vouchers.html [find where name=username]
```

The generated voucher card is available by accessing the router using a WEB browser and navigating to */um/PRIVATE/GENERATED/vouchers /gen_printable_vouchers.html*

By default, the printable card looks like this:



⚠ To access the PRIVATE path of the /um/ directory by the WEB browser, *private-username* and *private-password* must be configured. See **Settin gs** section.

It is possible to use different variables when generating vouchers. Currently, supported variables are:

$(username) - Represents User Manager username
$(password) - Password of the username
$(userprofname) - Profile that is active for the particular user
$(userprofendtime) - Profile validity end time if specified

Multiple user credential export

It is possible to generate a CSV or XML file with multiple or all user credentials at once by using the *export.xml* or *export.csv* as *voucher-template*.

```
/user-manager user
generate-voucher voucher-template=export.xml [find]
```

The command generates an XML file *um5files/PRIVATE/GENERATED/vouchers/gen_export.xml* which can either be accessible by the WEB browser or any other file access tools.

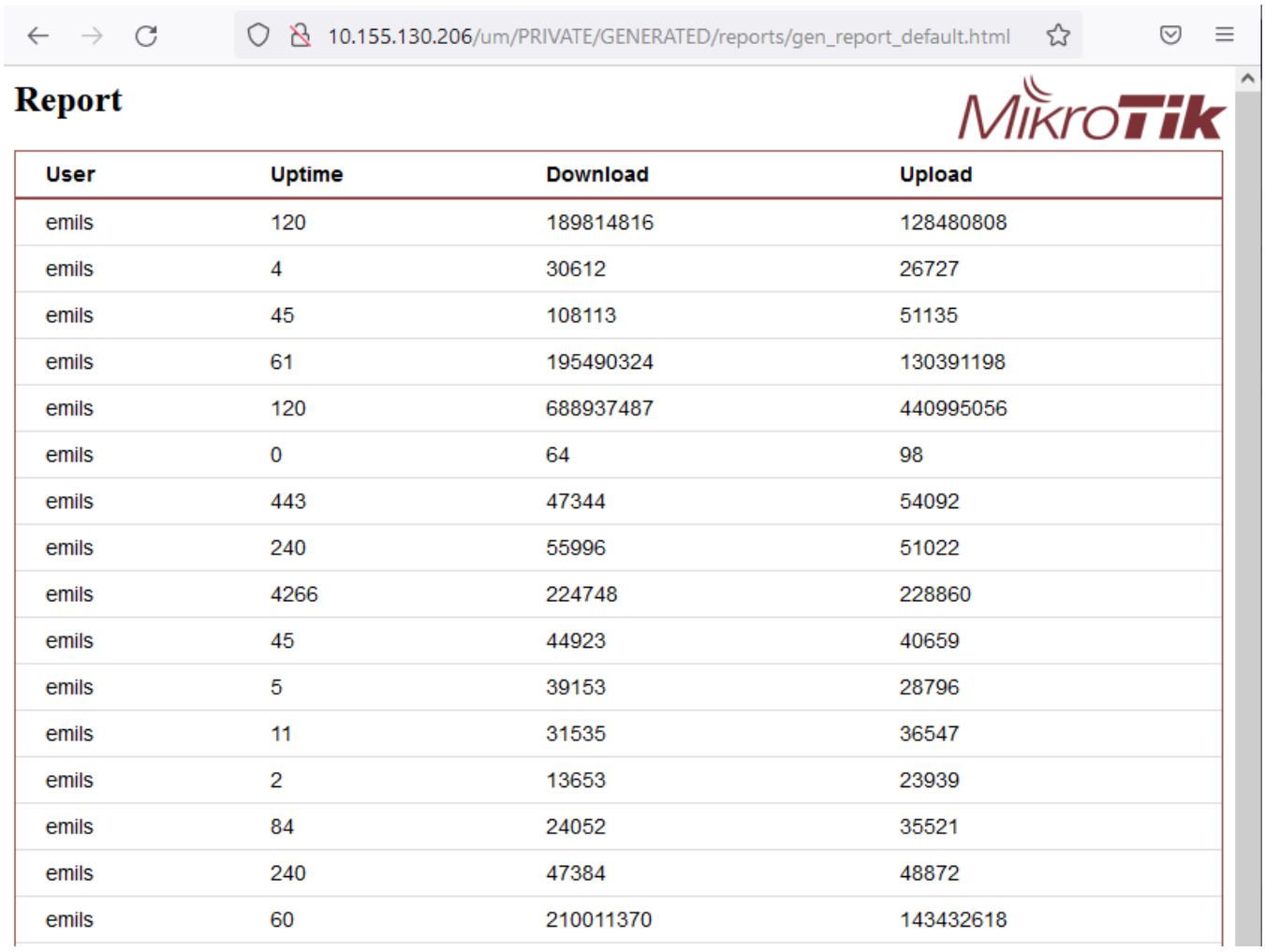
```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user>
        <username>olsgkl</username>
        <password>86a6zH</password>
    </user>
    <user>
        <username>lkbwss</username>
        <password>jaKY5V</password>
    </user>
    <user>
        <username>cwxbwu</username>
        <password>a62yZd</password>
    </user>
    <user>
        <username>username</username>
        <password>secretpassword</password>
    </user>

</users>
```

Generating usage report

In cases where presentable network usage information is required by companies billing or legal team an automated session export can be created using the *generate-report* command. The command requires an input of the report template - an example of the template is available in *um5files/PRIVATE /TEMPLATES/reports/report_default.html*. Example of the report generation:

```
/user-manager
generate-report report-template=report_default.html columns=username,uptime,download,upload
```

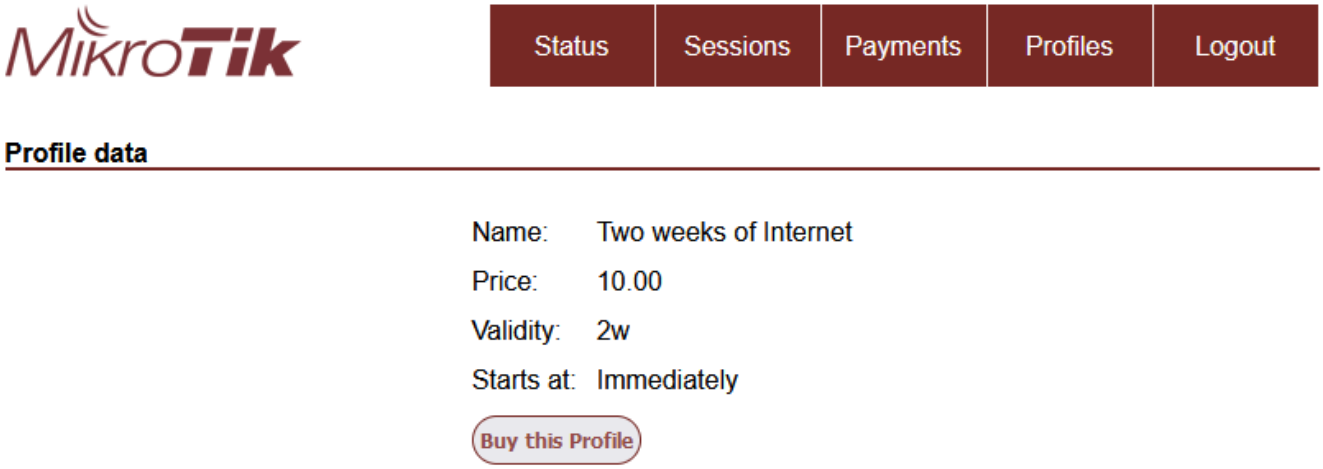
The generated report is available by accessing the router using a WEB browser and navigating to */um/PRIVATE/GENERATED/reports/gen_report_default. html*



User	Uptime	Download	Upload
emils	120	189814816	128480808
emils	4	30612	26727
emils	45	108113	51135
emils	61	195490324	130391198
emils	120	688937487	440995056
emils	0	64	98
emils	443	47344	54092
emils	240	55996	51022
emils	4266	224748	228860
emils	45	44923	40659
emils	5	39153	28796
emils	11	31535	36547
emils	2	13653	23939
emils	84	24052	35521
emils	240	47384	48872
emils	60	210011370	143432618

Purchasing a profile

After logging into the user's private profile by accessing the router's */um/* directory using a WEB browser, for example, http://example.com/um/, he will be able to see all available **Profiles** in the respective menu. Profiles that have specified *price* values will have a *Buy this Profile* button available.



After pressing the *Buy this Profile* button, the user will be asked to choose from available transaction service providers (currently only PayPal is available) and later redirected to PayPal's payment processing page.

Test Store

PayPal

USD 10,00

Hi, John! **Log out**

Pay with

PayPal Balance

EUR 9,10

Make this my preferred way to pay

PayPal's conversion rate: 1 EUR = 1,09885 USD

Visa

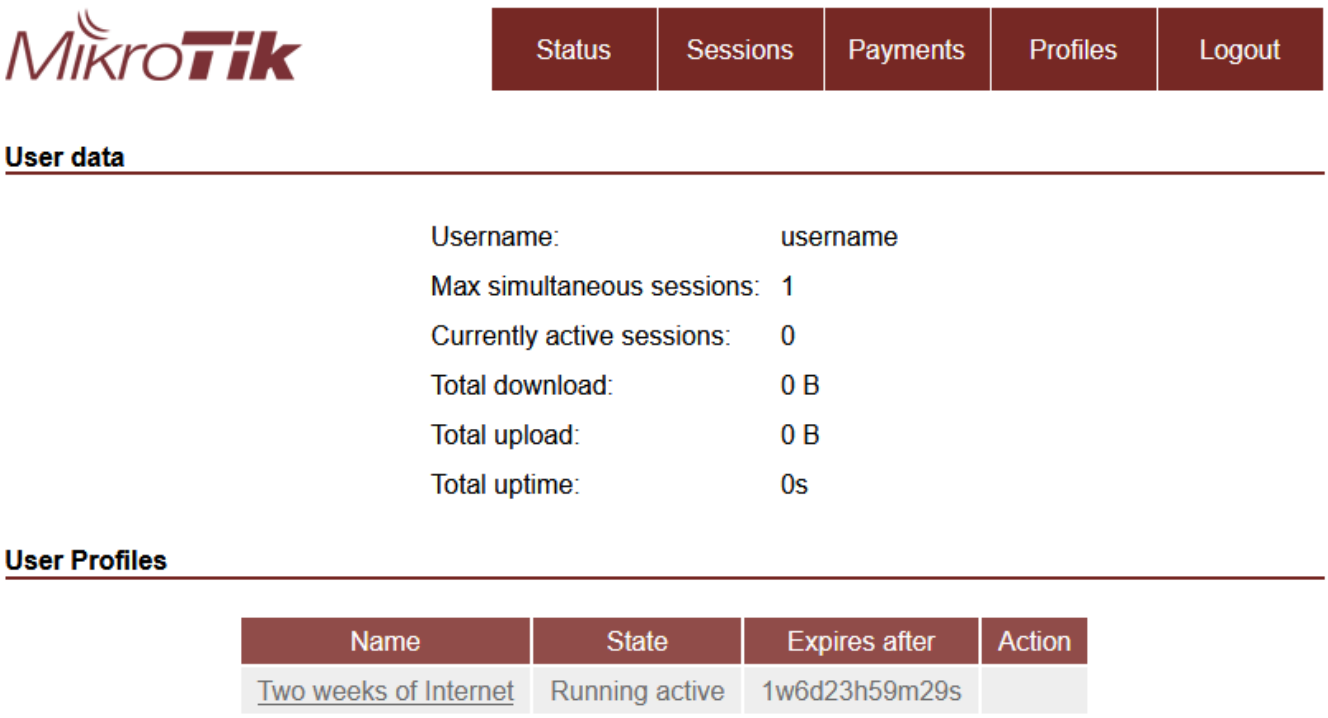
Credit ••••5733

+ Add debit or credit card

Pay Now

Cancel and return to Test Store

When the payment is completed, the User Manager will ask PayPal to approve the transaction. After approval, the profile is assigned to the user and is ready to use.



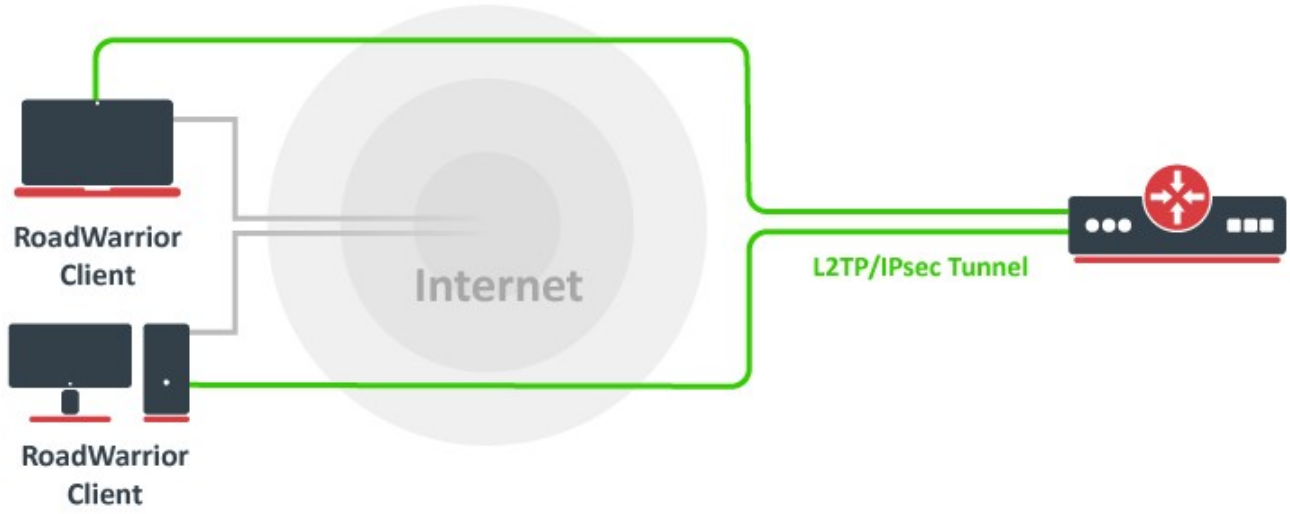
Migrating from RouterOS v6

When you upgrade your User Manager router from RouterOS v6 to the v7 the new User Manager will work with new database files and configuration. To continue using the old user, router, profile, etc. configuration you must manually execute the migrate command. To do so you must have files from the old User Manager server folder "user-manager" present. The folder can be renamed, but all the contents from the old installation must be transferred to the new v7 installation (you can move the old configuration from one router to another router with v7, you must copy "user-manager" folder). After that, all you need to do is execute this command - "/user-manager/database/migrate-legacy-db database-path=<path_to_old_user_manager_folder>".

The import process will try to convert such configuration - users, profiles, user-profiles, limitations, profile-limitations, user-counters, routers, and sessions.

Application Examples

Basic L2TP/IPsec server with User Manager authentication



User Manager configuration

Start off by enabling User Manager functionality.

```
/user-manager
set enabled=yes
```

Allow receiving RADIUS requests from the localhost (the router itself).

```
/user-manager router
add address=127.0.0.1 comment=localhost name=local shared-secret=test
```

Next, add users and their credentials that clients will use to authenticate to the server.

```
/user-manager user
add name=user1 password=password
```

Configuring RADIUS client

For the router to use the RADIUS server for user authentication, it is required to add a new RADIUS client that has the same shared secret that we already configured on User Manager.

```
/radius
add address=127.0.0.1 secret=test service=ipsec
```

L2TP/IPsec server configuration

Configure the IP pool from which IP addresses will be assigned to the users and assign it to the PPP Profile.

```
/ip pool
add name=vpn-pool range=192.168.99.2-192.168.99.100

/ppp profile
set default-encryption local-address=192.168.99.1 remote-address=vpn-pool
```

Enable the use of RADIUS for PPP authentication.

```
/ppp aaa
set use-radius=yes
```

Enable the L2TP server with IPsec encryption.

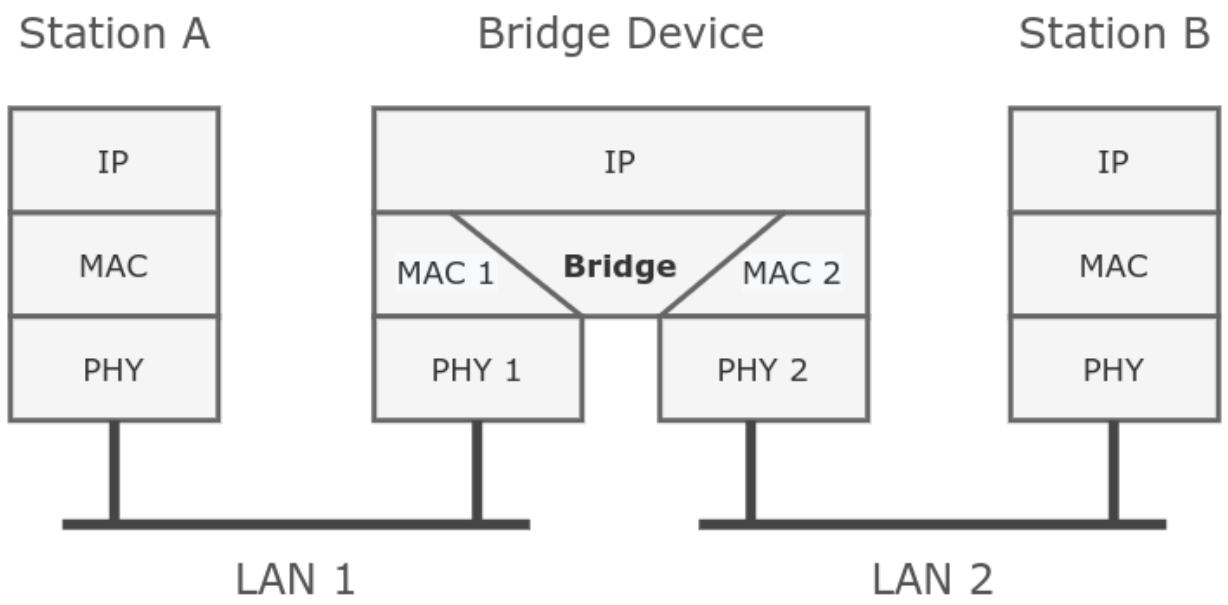
```
/interface l2tp-server server
set enabled=yes use-ipsec=required ipsec-secret=mySecret
```

That is it. Your router is now ready to accept L2TP/IPsec connections and authenticate them to the internal User Manager.

Bridging and Switching

Other resources:

Summary



Ethernet-like networks (Ethernet, Ethernet over IP, IEEE 802.11 in ap-bridge or bridge mode, WDS, VLAN) can be connected together using MAC bridges. The bridge feature allows the interconnection of hosts connected to separate LANs (using EoIP, geographically distributed networks can be bridged as well if any kind of IP network interconnection exists between them) as if they were attached to a single LAN. As bridges are transparent, they do not appear in the traceroute list, and no utility can make a distinction between a host working in one LAN and a host working in another LAN if these LANs are bridged. However, depending on the way the LANs are interconnected, latency and data rate between hosts may vary.

Network loops may emerge (intentionally or not) in complex topologies. Without any special treatment, loops would prevent the network from functioning normally, as they would lead to avalanche-like packet multiplication. Each bridge runs an algorithm that calculates how the loop can be prevented. (R/M)STP allows bridges to communicate with each other, so they can negotiate a loop-free topology. All other alternative connections that would otherwise form loops are put on standby, so that should the main connection fail, another connection could take its place. This algorithm exchanges configuration messages (BPDU - Bridge Protocol Data Unit) periodically, so that all bridges are updated with the newest information about changes in a network topology. (R/M)STP selects a root bridge which is responsible for network reconfiguration, such as blocking and opening ports on other bridges. The root bridge is the bridge with the lowest bridge ID.

Bridge Interface Setup

To combine a number of networks into one bridge, a bridge interface should be created. Later, all the desired interfaces should be set up as its ports. One MAC address from slave (secondary) ports will be assigned to the bridge interface. The MAC address will be chosen automatically, depending on the "port-number", and it can change after a reboot. To avoid unwanted MAC address changes, it is recommended to disable "`auto-mac`" and manually specifying the MAC address by using "`admin-mac`".

Sub-menu: `/interface bridge`

Property	Description
add-dhcp-option82 (*yes* | *no*; Default: **no**)	Whether to add DHCP Option-82 information (Agent Remote ID and Agent Circuit ID) to DHCP packets. Can be used together with Option-82 capable DHCP server to assign IP addresses and implement policies. This property only has an effect when `dhcp-snooping` is set to `yes`.
admin-mac (*MAC address*; Default: **none**)	Static MAC address of the bridge. This property only has an effect when `auto-mac` is set to `no`.
ageing-time (*time*; Default: **00:05:00**)	How long a host's information will be kept in the bridge database.

arp (*disabled* | *enabled* | *local-proxy-arp* | *proxy-arp* | *reply-only*; Default: **enabled**)	Address Resolution Protocol setting • `disabled` - the interface will not use ARP • `enabled` - the interface will use ARP • `local-proxy-arp` - the router performs proxy ARP on the interface and sends replies to the same interface • `proxy-arp` - the router performs proxy ARP on the interface and sends replies to other interfaces • `reply-only` - the interface will only respond to requests originating from matching IP address/MAC address combinations that are entered as static entries in the IP/ARP table. No dynamic entries will be automatically stored in the IP/ARP table. Therefore, for communications to be successful, a valid static entry must already exist.
arp-timeout (*auto* | *integer*; Default: **auto**)	How long the ARP record is kept in the ARP table after no packets are received from IP address. Value `auto` equals to the value of `arp-timeout` in `ip/settings`, default is 30s.
auto-mac (*yes* | *no*; Default: **yes**)	Automatically select one MAC address of bridge ports as a bridge MAC address, bridge MAC will be chosen from the first added bridge port. After a device reboots, the bridge MAC can change depending on the port-number.
comment (*string*; Default:)	Short description of the interface.
dhcp-snooping (*yes* | *no*; Default: **no**)	Enables or disables DHCP Snooping on the bridge.
disabled (*yes* | *no*; Default: **no**)	Changes whether the bridge is disabled.
ether-type (*0x9100* | *0x8100* | *0x88a8*; Default: **0x8100**)	Changes the EtherType, which will be used to determine if a packet has a VLAN tag. Packets that have a matching EtherType are considered as tagged packets. This property only has an effect when `vlan-filtering` is set to `yes`.
fast-forward (*yes* | *no*; Default: **yes**)	Special and faster case of Fast Path which works only on bridges with 2 interfaces (enabled by default only for new bridges). More details can be found in the Fast Forward section.
forward-delay (*time*; Default: **00:00:15**)	The time which is spent during the initialization phase of the bridge interface (i.e., after router startup or enabling the interface) in the listening/learning state before the bridge will start functioning normally.
frame-types (*admit-all* | *admit-only-untagged-and-priority-tagged* | *admit-only-vlan-tagged*; Default: **admit-all**)	Specifies allowed frame types on a bridge port. This property only has an effect when `vlan-filtering` is set to `yes`.
igmp-snooping (*yes* | *no*; Default: **no**)	Enables multicast group and port learning to prevent multicast traffic from flooding all interfaces in a bridge.
igmp-version (*2* | *3*; Default: **2**)	Selects the IGMP version in which IGMP membership queries will be generated when the bridge interface is acting as an IGMP querier. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
ingress-filtering (*yes* | *no*; Default: **yes**)	Enables or disables VLAN ingress filtering, which checks if the ingress port is a member of the received VLAN ID in the bridge VLAN table. By default, VLANs that don't exist in the bridge VLAN table are dropped before they are sent out (egress), but this property allows you to drop the packets when they are received (ingress). Should be used with `frame-types` to specify if the ingress traffic should be tagged or untagged. This property only has an effect when `vlan-filtering` is set to `yes`. The setting is enabled by default since RouterOS v7.
l2mtu (*read-only*; Default:)	L2MTU indicates the maximum size of the frame without a MAC header that can be sent by this interface. The L2MTU value will be automatically set by the bridge and it will use the lowest L2MTU value of any associated bridge port. This value cannot be manually changed.
last-member-interval (*time*; Default: **1s**)	When the last client on the bridge port unsubscribes to a multicast group and the bridge is acting as an active querier, the bridge will send group-specific IGMP/MLD query, to make sure that no other client is still subscribed. The setting changes the response time for these queries. In case no membership reports are received in a certain time period (`last-member-interval` * `last-member-query-count`), the multicast group is removed from the multicast database (MDB). If the bridge port is configured with fast-leave, the multicast group is removed right away without sending any queries. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.

last-member-query-count (*integer: 0..4294967295*; Default: **2**)	How many times should `last-member-interval` pass until the IGMP/MLD snooping bridge stops forwarding a certain multicast stream. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
max-hops (*integer: 6..40*; Default: **20**)	Bridge count which BPDU can pass in an MSTP enabled network in the same region before BPDU is being ignored. This property only has an effect when `protocol-mode` is set to `mstp`.
max-message-age (*time: 6s..40s*; Default: **0 0:00:20**)	Changes the Max Age value in BPDU packets, which is transmitted by the root bridge. A root bridge sends a BPDUs with Max Age set to `max-message-age` value and a Message Age of 0. Every sequential bridge will increment the Message Age before sending their BPDUs. Once a bridge receives a BPDU where Message Age is equal or greater than Max Age, the BPDU is ignored. This property only has an effect when `protocol-mode` is set to `stp` or `rstp`.
membership-interval (*time*; Default: **4m20s**)	The amount of time after an entry in the Multicast Database (MDB) is removed if no IGMP/MLD membership reports are received on a bridge port. This property only has an effect when `igmp-snooping` is set to `yes`.
mld-version (*1 | 2*; Default: **1**)	Selects the MLD version in which MLD membership queries will be generated, when the bridge interface is acting as an MLD querier. This property only has an effect when the bridge has an active IPv6 address, `igmp-snooping` and `multicast-querier` is set to `yes`.
mtu (*integer*; Default: **auto**)	Maximum transmission unit, by default, the bridge will set MTU automatically and it will use the lowest MTU value of any associated bridge port. The default bridge MTU value without any bridge ports added is 1500. The MTU value can be set manually, but it cannot exceed the bridge L2MTU or the lowest bridge port L2MTU. If a new bridge port is added with L2MTU which is smaller than the `actual-mtu` of the bridge (set by the `mtu` property), then manually set value will be ignored and the bridge will act as if `mtu=auto` is set. When adding VLAN interfaces on the bridge, and when VLAN is using higher MTU than default 1500, it is recommended to set manually the MTU of the bridge.
multicast-querier (*yes | no*; Default: **no**)	Multicast querier generates periodic IGMP/MLD general membership queries to which all IGMP/MLD capable devices respond with an IGMP/MLD membership report, usually a PIM (multicast) router or IGMP proxy generates these queries. By using this property you can make an IGMP/MLD snooping enabled bridge to generate IGMP/MLD general membership queries. This property should be used whenever there is no active querier (PIM router or IGMP proxy) in a Layer2 network. Without a multicast querier in a Layer2 network, the Multicast Database (MDB) is not being updated, the learned entries will timeout and IGMP/MLD snooping will not function properly. Only untagged IGMP/MLD general membership queries are generated, IGMP queries are sent with IPv4 0.0.0.0 source address, MLD queries are sent with IPv6 link-local address of the bridge interface. The bridge will not send queries if an external IGMP/MLD querier is detected (see the monitoring values `igmp-querier` and `mld-querier`). This property only has an effect when `igmp-snooping` is set to `yes`.
multicast-router (*disabled | permanent | temporary-query*; Default: **temporary-query**)	A multicast router port is a port where a multicast router or querier is connected. On this port, unregistered multicast streams and IGMP/MLD membership reports will be sent. This setting changes the state of the multicast router for a bridge interface itself. This property can be used to send IGMP/MLD membership reports to the bridge interface for further multicast routing or proxying. This property only has an effect when `igmp-snooping` is set to `yes`. • `disabled` - disabled multicast router state on the bridge interface. Unregistered multicast and IGMP/MLD membership reports are not sent to the bridge interface regardless of what is configured on the bridge interface. • `permanent` - enabled multicast router state on the bridge interface. Unregistered multicast and IGMP/MLD membership reports are sent to the bridge interface itself regardless of what is configured on the bridge interface. • `temporary-query` - automatically detect multicast router state on the bridge interface using IGMP/MLD queries.
name (*text*; Default: **bridgeN**)	Name of the bridge interface.

<table>
<tr><td>port-cost-mode (long | short; Default: long)</td><td>Changes the port path-cost and internal-path-cost mode for bridged ports, utilizing automatic values based on interface speed. This setting does not impact bridged ports with manually configured <code>path-cost</code> or <code>internal-path-cost</code> properties. Below are examples illustrating the path-costs corresponding to specific data rates (with proportionate calculations for intermediate rates):

<table>
<tr><th>Data rate</th><th>Long</th><th>Short</th></tr>
<tr><td>10 Mbps</td><td>2,000,000</td><td>100</td></tr>
<tr><td>100 Mbps</td><td>200,000</td><td>19</td></tr>
<tr><td>1 Gbps</td><td>20,000</td><td>4</td></tr>
<tr><td>10 Gbps</td><td>2,000</td><td>2</td></tr>
<tr><td>25 Gbps</td><td>800</td><td>1</td></tr>
<tr><td>40 Gbps</td><td>500</td><td>1</td></tr>
<tr><td>50 Gbps</td><td>400</td><td>1</td></tr>
<tr><td>100 Gbps</td><td>200</td><td>1</td></tr>
</table>

For bonded interfaces, the highest path-cost among all bonded member ports is applied, this value remains unaffected by the total link speed of the bonding.

For virtual interfaces (such as VLAN, EoIP, VXLAN), as well as wifi, wireless, and 60GHz interfaces, a path-cost of 20,000 is assigned for long mode, and 10 for short mode.

For dynamically bridged interfaces (e.g. wifi, wireless, PPP, VPLS), the path-cost defaults to 20,000 for long mode and 10 for short mode. However, this can be manually overridden by the service that dynamically adds interfaces to bridge, for instance, by using the CAPsMAN <code>datapath.bridge-cost</code> setting.

Use port monitor to observe the applied path-cost.

This property has an effect when <code>protocol-mode</code> is set to <code>stp</code>, <code>rstp</code>, or <code>mstp</code>.</td></tr>
<tr><td>priority (integer: 0..65535 decimal format or 0x0000-0xffff hex format; Default: 32768 / 0x8000)</td><td>Bridge priority, used by R/STP to determine root bridge, used by MSTP to determine CIST and IST regional root bridge. This property has no effect when <code>protocol-mode</code> is set to <code>none</code>.</td></tr>
<tr><td>protocol-mode (none | rstp | stp | mstp; Default: rstp)</td><td>Select Spanning tree protocol (STP) or Rapid spanning tree protocol (RSTP) to ensure a loop-free topology for any bridged LAN. RSTP provides a faster spanning tree convergence after a topology change. Select MSTP to ensure loop-free topology across multiple VLANs. Since RouterOS v6.43 it is possible to forward Reserved MAC addresses that are in the 01:80:C2:00:00:0X range, this can be done by setting the <code>protocol-mode</code> to <code>none</code>.</td></tr>
<tr><td>pvid (integer: 1..4094; Default: 1)</td><td>Port VLAN ID (pvid) specifies which VLAN the untagged ingress traffic is assigned to. It applies e.g. to frames sent from bridge IP and destined to a bridge port. This property only has an effect when <code>vlan-filtering</code> is set to <code>yes</code>.</td></tr>
<tr><td>querier-interval (time; Default: 4m15s)</td><td>Changes the timeout period for detected querier and multicast-router ports. This property only has an effect when <code>igmp-snooping</code> is set to <code>yes</code>.</td></tr>
<tr><td>query-interval (time; Default: 2m5s)</td><td>Changes the interval on how often IGMP/MLD general membership queries are sent out when the bridge interface is acting as an IGMP/MLD querier. The interval takes place when the last startup query is sent. This property only has an effect when <code>igmp-snooping</code> and <code>multicast-querier</code> is set to <code>yes</code>.</td></tr>
<tr><td>query-response-interval (time; Default: 10s)</td><td>The setting changes the response time for general IGMP/MLD queries when the bridge is active as an IGMP/MLD querier. This property only has an effect when <code>igmp-snooping</code> and <code>multicast-querier</code> is set to <code>yes</code>.</td></tr>
<tr><td>region-name (text; Default:)</td><td>MSTP region name. This property only has an effect when <code>protocol-mode</code> is set to <code>mstp</code>.</td></tr>
<tr><td>region-revision (integer: 0..65535; Default: 0)</td><td>MSTP configuration revision number. This property only has an effect when <code>protocol-mode</code> is set to <code>mstp</code>.</td></tr>
</table>

startup-query-count (*integer: 0..4294967295*; Default: **2**)	Specifies how many times general IGMP/MLD queries must be sent when bridge interface is enabled or active querier timeouts. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
startup-query-interval (*time*; Default: **31s250 ms**)	Specifies the interval between startup general IGMP/MLD queries. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
transmit-hold-count (*integer: 1..10*; Default: **6**)	The Transmit Hold Count used by the Port Transmit state machine to limit the transmission rate.
vlan-filtering (*yes* | *no*; Default: **no**)	Globally enables or disables VLAN functionality for the bridge.

Changing certain properties can cause the bridge to temporarily disable all ports. This must be taken into account whenever changing such properties on production environments since it can cause all packets to be temporarily dropped. Such properties include `vlan-filtering`, `protocol-mode`, `igmp-snooping`, `fast-forward` and others.

Example

To add and enable a bridge interface that will forward L2 packets:

```
[admin@MikroTik] > interface bridge add
[admin@MikroTik] > interface bridge print
Flags: X - disabled, R - running
0 R name="bridge1" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled arp-timeout=auto mac-address=5E:D2:42:95:
56:7F protocol-mode=rstp fast-forward=yes
igmp-snooping=no auto-mac=yes ageing-time=5m priority=0x8000 max-message-age=20s forward-delay=15s transmit-
hold-count=6 vlan-filtering=no
dhcp-snooping=no
```

Bridge Monitoring

To monitor the current status of a bridge interface, use the `monitor` command.

Sub-menu: `/interface bridge monitor`

Property	Description
current-mac-address (*MAC address*)	Current MAC address of the bridge
designated-port-count (*integer*)	Number of designated bridge ports
igmp-querier (*none* | *interface & IPv4 address*)	Shows a bridge port and source IP address from the detected IGMP querier. Only shows detected external IGMP querier, local bridge IGMP querier (including IGMP proxy and PIM) will not be displayed. Monitoring value appears only when `igmp-snooping` is enabled.
mld-querier (*none* | *interface & IPv6 address*)	Shows a bridge port and source IPv6 address from the detected MLD querier. Only shows detected external MLD querier, local bridge MLD querier will not be displayed. Monitoring value appears only when `igmp-snooping` is enabled and the bridge has an active IPv6 address.
multicast-router (*yes* | *no*)	Shows if a multicast router is detected on the port. Monitoring value appears only when `igmp-snooping` is enabled.
port-count (*integer*)	Number of the bridge ports
root-bridge (*yes* | *no*)	Shows whether the bridge is the root bridge of the spanning tree

root-bridge-id (*text*)	The root bridge ID, which is in form of bridge-priority.bridge-MAC-address
root-path-cost (*integer*)	The total cost of the path to the root-bridge
root-port (*name*)	Port to which the root bridge is connected to
state (*enabled / disabled*)	State of the bridge

```
[admin@MikroTik] /interface bridge monitor bridge1
                  state: enabled
    current-mac-address: CC:2D:E0:E4:B3:38
            root-bridge: yes
         root-bridge-id: 0x8000.CC:2D:E0:E4:B3:38
         root-path-cost: 0
              root-port: none
             port-count: 2
  designated-port-count: 2
           fast-forward: no
```

Spanning Tree Protocol

RouterOS bridge interfaces are capable of running Spanning Tree Protocol to ensure a loop-free and redundant topology. For small networks with just 2 bridges STP does not bring many benefits, but for larger networks properly configured STP is very crucial, leaving STP-related values to default may result in a completely unreachable network in case of an even single bridge failure. To achieve a proper loop-free and redundant topology, it is necessary to properly set bridge priorities, port path costs, and port priorities.

In RouterOS it is possible to set any value for bridge priority between 0 and 65535, the IEEE 802.1W standard states that the bridge priority must be in steps of 4096. This can cause incompatibility issues between devices that do not support such values. To avoid compatibility issues, it is recommended to use only these priorities: 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672, 32768, 36864, 40960, 45056, 49152, 53248, 57344, 61440

STP has multiple variants, currently, RouterOS supports STP, RSTP, and MSTP. Depending on needs, either one of them can be used, some devices are able to run some of these protocols using hardware offloading, detailed information about which device support it can be found in the Hardware Offloading section. STP is considered to be outdated and slow, it has been almost entirely replaced in all network topologies by RSTP, which is backward compatible with STP. For network topologies that depend on VLANs, it is recommended to use MSTP since it is a VLAN aware protocol and gives the ability to do load balancing per VLAN groups. There are a lot of considerations that should be made when designing an STP enabled network, more detailed case studies can be found in the Spanning Tree Protocol article. In RouterOS, the `protocol-mode` property controls the used STP variant.

RouterOS bridge does not work with PVST and its variants. The PVST BPDUs (with a MAC destination 01:00:0C:CC:CC:CD) are treated by RouterOS bridges as typical multicast packets. In simpler terms, they undergo RouterOS bridge/switch forwarding logic and may get tagged or untagged.

By the IEEE 802.1ad standard, the BPDUs from bridges that comply with IEEE 802.1Q are not compatible with IEEE 802.1ad bridges, this means that the same bridge VLAN protocol should be used across all bridges in a single Layer2 domain, otherwise (R/M)STP will not function properly.

Per-port STP

There might be certain situations where you want to limit STP functionality on single or multiple ports. Below you can find some examples for different use cases.

Be careful when changing the default (R/M)STP functionality, make sure you understand the working principles of STP and BPDUs. Misconfigured (R/M)STP can cause unexpected behavior.

Create edge ports

Setting a bridge port as an edge port will restrict it from sending BPDUs and will ignore any received BPDUs:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 edge=yes
add bridge=bridge1 interface=ether2
```

Drop received BPDUs

The bridge filter or NAT rules cannot drop BPDUs when the bridge has STP/RSTP/MSTP enabled due to the special processing of BPDUs. However, dropping received BPDUs on a certain port can be done on some switch chips using ACL rules:

On CRS3xx:

```
/interface ethernet switch rule
add dst-mac-address=01:80:C2:00:00:00/FF:FF:FF:FF:FF:FF new-dst-ports="" ports=ether1 switch=switch1
```

On CRS1xx/CRS2xx with Access Control List (ACL) support:

```
/interface ethernet switch acl
add action=drop mac-dst-address=01:80:C2:00:00:00 src-ports=ether1
```

In this example all received BPDUs on **ether1** are dropped.

If you intend to drop received BPDUs on a port, then make sure to prevent BPDUs from being sent out from the interface that this port is connected to. A root bridge always sends out BPDUs and under normal conditions is waiting for a more superior BPDU (from a bridge with a lower bridge ID), but the bridge must temporarily disable the new root-port when transitioning from a root bridge to a designated bridge. If you have blocked BPDUs only on one side, then a port will flap continuously.

Enable BPDU guard

In this example, if **ether1** receives a BPDU, it will block the port and will require you to manually re-enable it.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 bpdu-guard=yes
add bridge=bridge1 interface=ether2
```

Enable Root guard

In this example, **ether1** is configured with `restricted-role=yes.` It prevented the port from becoming the root port for the CIST or any MSTI, regardless of its best spanning tree priority vector. Such a port will be selected as an Alternate Port (discarding state) and remains so as long as it continues to receive superior BPDUs. It will automatically transition to the forwarding state when it no longer detects a superior root path. Network administrators may enable this setting to safeguard against external bridges influencing the active spanning tree.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 restricted-role=yes
add bridge=bridge1 interface=ether2

[admin@MikroTik] /interface/bridge/port monitor [find]
                  interface: ether1                   ether2
                     status: in-bridge                in-bridge
                port-number: 1                        2
                       role: alternate-port           designated-port
                  edge-port: no                       yes
        edge-port-discovery: yes                      yes
        point-to-point-port: yes                      yes
               external-fdb: no                       no
               sending-rstp: yes                      yes
                   learning: no                       yes
                 forwarding: no                       yes
           actual-path-cost: 20000                    20000
             root-path-cost: 20000
          designated-bridge: 0x7000.64:D1:54:C7:3A:6E
            designated-cost: 0
    designated-port-number: 1
          hw-offload-group: switch1                  switch1
```

Bridge Settings

Under the bridge settings menu, it is possible to control certain features for all bridge interfaces and monitor global bridge counters.

Sub-menu: `/interface bridge settings`

Property	Description
use-ip-firewall (*yes* | *no*; Default: **no**)	Force bridged traffic to also be processed by prerouting, forward, and postrouting sections of IP routing (see more details on Packet Flow article). This does not apply to routed traffic. This property is required in case you want to assign Simple Queues or global Queue Tree to traffic in a bridge. Property `use-ip-firewall-for-vlan` is required in case bridge `vlan-filtering` is used.
use-ip-firewall-for-pppoe (*yes* | *no*; Default: **no**)	Send bridged un-encrypted PPPoE traffic to also be processed by IP/Firewall. This property only has an effect when `use-ip-firewall` is set to `yes`. This property is required in case you want to assign Simple Queues or global Queue Tree to PPPoE traffic in a bridge.
use-ip-firewall-for-vlan (*yes* | *no*; Default: **no**)	Send bridged VLAN traffic to also be processed by IP/Firewall. This property only has an effect when `use-ip-firewall` is set to `yes`. This property is required in case you want to assign Simple Queues or global Queue Tree to VLAN traffic in a bridge.
allow-fast-path (*yes* | *no*; Default: **yes**)	Whether to enable a bridge Fast Path globally.
bridge-fast-path-active (*yes* | *no*; Default: *)*	Shows whether a bridge FastPath is active globally, Fast Path status per bridge interface is not displayed.
bridge-fast-path-packets (*integer*; Default: *)*	Shows packet count forwarded by bridge Fast Path.

bridge-fast-path-bytes (*integer*; Default: *)*	Shows byte count forwarded by bridge Fast Path.
bridge-fast-forward-packets (*integer*; Default: *)*	Shows packet count forwarded by bridge Fast Forward.
bridge-fast-forward-bytes (*integer*; Default: *)*	Shows byte count forwarded by bridge Fast Forward.

⚠ In case you want to assign Simple Queues or global Queue Trees to traffic that is being forwarded by a bridge, then you need to enable the `use-ip-firewall` property. Without using this property the bridge traffic will never reach the postrouting chain, Simple Queues and global Queue Trees are working in the postrouting chain. To assign Simple Queues or global Queue Trees for VLAN or PPPoE traffic in a bridge you should enable appropriate properties as well.

Port Settings

Port submenu is used to add interfaces in a particular bridge.

Sub-menu: `/interface bridge port`

Property	Description
auto-isolate (*yes* | *no*; Default: **no**)	When enabled, prevents a port moving from discarding into forwarding state if no BPDUs are received from the neighboring bridge. The port will change into a forwarding state only when a BPDU is received. This property only has an effect when `protocol-mode` is set to `rstp` or `mstp` and `edge` is set to `no`.
bpdu-guard (*yes* | *no*; Default: **no**)	Enables or disables BPDU Guard feature on a port. This feature puts the port in a disabled role if it receives a BPDU and requires the port to be manually disabled and enabled if a BPDU was received. Should be used to prevent a bridge from BPDU related attacks. This property has no effect when `protocol-mode` is set to `none`.
bridge (*name*; Default: **none**)	The bridge interface where the respective interface is grouped in.
broadcast-flood (*yes* | *no*; Default: **yes**)	When enabled, bridge floods broadcast traffic to all bridge egress ports. When disabled, drops broadcast traffic on egress ports. Can be used to filter all broadcast traffic on an egress port. Broadcast traffic is considered as traffic that uses **FF:FF:FF:FF:FF:FF** as destination MAC address, such traffic is crucial for many protocols such as DHCP, ARP, NDP, BOOTP (Netinstall), and others. This option does not limit traffic flood to the CPU.
edge (*auto* | *no* | *no-discover* | *yes* | *yes-discover*; Default: **auto**)	Set port as edge port or non-edge port, or enable edge discovery. Edge ports are connected to a LAN that has no other bridges attached. An edge port will skip the learning and the listening states in STP and will transition directly to the forwarding state, this reduces the STP initialization time. If the port is configured to discover edge port then as soon as the bridge detects a BPDU coming to an edge port, the port becomes a non-edge port. This property has no effect when `protocol-mode` is set to `none`. • `no` - non-edge port, will participate in learning and listening states in STP. • `no-discover` - non-edge port with enabled discovery, will participate in learning and listening states in STP, a port can become an edge port if no BPDU is received. • `yes` - edge port without discovery, will transit directly to forwarding state. • `yes-discover` - edge port with enabled discovery, will transit directly to forwarding state. • `auto` - same as `no-discover`, but will additionally detect if a bridge port is a Wireless interface with disabled bridge-mode, such interface will be automatically set as an edge port without discovery.
fast-leave (*yes* | *no*; Default: **no**)	Enables IGMP/MLD fast leave feature on the bridge port. The bridge will stop forwarding multicast traffic to a bridge port when an IGMP/MLD leave message is received. This property only has an effect when `igmp-snooping` is set to `yes`.

frame-types (*admit-all | admit-only-untagged-and-priority-tagged | admit-only-vlan-tagged*; Default: **admit-all**)	Specifies allowed ingress frame types on a bridge port. This property only has an effect when `vlan-filtering` is set to `yes`.
ingress-filtering (*yes | no*; Default: **yes**)	Enables or disables VLAN ingress filtering, which checks if the ingress port is a member of the received VLAN ID in the bridge VLAN table. Should be used with `frame-types` to specify if the ingress traffic should be tagged or untagged. This property only has effect when `vlan-filtering` is set to `yes`. The setting is enabled by default since RouterOS v7.
learn (*auto | no | yes*; Default: **auto**)	Changes MAC learning behavior on a bridge port • `yes` - enables MAC learning • `no` - disables MAC learning • `auto` - detects if bridge port is a Wireless interface and uses a Wireless registration table instead of MAC learning, will use Wireless registration table if the Wireless interface is set to one of `ap-bridge`, `bridge`, `wds-slave` mode and bridge mode for the Wireless interface is disabled.
multicast-router (*disabled | permanent | temporary-query*; Default: **temporary-query**)	A multicast router port is a port where a multicast router or querier is connected. On this port, unregistered multicast streams and IGMP/MLD membership reports will be sent. This setting changes the state of the multicast router for bridge ports. This property can be used to send IGMP/MLD membership reports to certain bridge ports for further multicast routing or proxying. This property only has an effect when `igmp-snooping` is set to `yes`. • `disabled` - disabled multicast router state on the bridge port. Unregistered multicast and IGMP/MLD membership reports are not sent to the bridge port regardless of what is connected to it. • `permanent` - enabled multicast router state on the bridge port. Unregistered multicast and IGMP/MLD membership reports are sent to the bridge port regardless of what is connected to it. • `temporary-query` - automatically detect multicast router state on the bridge port using IGMP/MLD queries.
horizon (*integer 0..429496729*; Default: **none**)	Use split horizon bridging to prevent bridging loops. Set the same value for a group of ports, to prevent them from sending data to ports with the same horizon value. Split horizon is a software feature that disables hardware offloading. Read more about Bridge split horizon.
hw (*yes | no*; Default: **yes**)	Allows to enable or disable hardware offloading on interfaces capable of HW offloading. For software interfaces like EoIP or VLAN this setting is ignored and has no effect. Certain bridge or port functions can automatically disable HW offloading, use the `print` command to see whether the "H" flag is active.
internal-path-cost (*integer: 1..200000000*; Default:)	Path cost to the interface for MSTI0 inside a region. If not manually configured, the bridge automatically determines the internal-path-cost based on the interface speed and the `port-cost-mode` setting. To revert to the automatic determination and remove any manually applied value, simply use an exclamation mark before the `internal-path-cost` property. This property only has effect when `protocol-mode` is set to `mstp`. `/interface bridge port set [find interface=sfp28-1] !internal-path-cost` Use port monitor to observe the applied internal-path-cost.
interface (*name*; Default: **none**)	Name of the interface.
path-cost (*integer: 1..200000000*; Default:)	Path cost to the interface, used by STP and RSTP to determine the best path, and used by MSTP to determine the best path between regions. If not manually configured, the bridge automatically determines the path-cost based on the interface speed and the `port-cost-mode` setting. To revert to the automatic determination and remove any manually applied value, simply use an exclamation mark before the `path-cost` property. This property has no effect when `protocol-mode` is set to `none`. `/interface bridge port set [find interface=sfp28-1] !path-cost` Use port monitor to observe the applied path-cost.

point-to-point (*auto* | *yes* | *no*; Default: **auto**)	Specifies if a bridge port is connected to a bridge using a point-to-point link for faster convergence in case of failure. By setting this property to `yes`, you are forcing the link to be a point-to-point link, which will skip the checking mechanism, which detects and waits for BPDUs from other devices from this single link. By setting this property to `no`, you are expecting that a link can receive BPDUs from multiple devices. By setting the property to `yes`, you are significantly improving (R/M)STP convergence time. In general, you should only set this property to `no` if it is possible that another device can be connected between a link, this is mostly relevant to Wireless mediums and Ethernet hubs. If the Ethernet link is full-duplex, `auto` enables point-to-point functionality. This property has no effect when `protocol-mode` is set to `none`.
priority (*integer: 0..240*; Default: **128**)	The priority of the interface, used by STP to determine the root port, used by MSTP to determine root port between regions.
pvid (*integer 1..4094*; Default: **1**)	Port VLAN ID (pvid) specifies which VLAN the untagged ingress traffic is assigned to. This property only has an effect when `vlan-filtering` is set to `yes`.
restricted-role (*yes* | *no*; Default: **no**)	Enables or disables the restricted role on a port. When enabled, it prevents the port from becoming the root port for the CIST or any MSTI, regardless of its best spanning tree priority vector. Such a port will be selected as an Alternate Port (discarding state) and remains so as long as it continues to receive superior BPDUs. It will automatically transition to the forwarding state when it no longer detects a superior root path. Network administrators may enable this setting to safeguard against external bridges influencing the active spanning tree, a feature also known as root-guard or root-protection. This property has an effect when `protocol-mode` is set to `stp`, `rstp`, or `mstp` (support for STP and RSTP is available since RouterOS v7.14).
restricted-tcn (*yes* | *no*; Default: **no**)	Enables or disables topology change notification (TCN) handling on a port. When enabled, it causes the port not to propagate received topology change notifications to other ports, and any changes caused by the port itself does not result in topology change notification to other ports. This parameter is disabled by default. It can be set by a network administrator to prevent external bridges causing MAC address flushing in local network. This property has an effect when `protocol-mode` is set to `stp`, `rstp`, or `mstp` (support for STP and RSTP is available since RouterOS v7.14).
tag-stacking (*yes* | *no*; Default: **no**)	Forces all packets to be treated as untagged packets. Packets on ingress port will be tagged with another VLAN tag regardless if a VLAN tag already exists, packets will be tagged with a VLAN ID that matches the `pvid` value and will use EtherType that is specified in `ether-type`. This property only has effect when `vlan-filtering` is set to `yes`.
trusted (*yes* | *no*; Default: **no**)	When enabled, it allows forwarding DHCP packets towards the DHCP server through this port. Mainly used to limit unauthorized servers to provide malicious information for users. This property only has an effect when `dhcp-snooping` is set to `yes`.
unknown-multicast-flood (*yes* | *no*; Default: **yes**)	Changes the multicast flood option on bridge port, only controls the egress traffic. When enabled, the bridge allows flooding multicast packets to the specified bridge port, but when disabled, the bridge restricts multicast traffic from being flooded to the specified bridge port. The setting affects all multicast traffic, this includes non-IP, IPv4, IPv6 and the link-local multicast ranges (e. g. 224.0.0.0/24 and ff02::1). Note that when `igmp-snooping` is enabled and IGMP/MLD querier is detected, the bridge will automatically restrict unknown IP multicast from being flooded, so the setting is not mandatory for IGMP/MLD snooping setups. When using this setting together with `igmp-snooping`, the only multicast traffic that is allowed on the bridge port is the known multicast from the MDB table.
unknown-unicast-flood (*yes* | *no*; Default: **yes**)	Changes the unknown unicast flood option on bridge port, only controls the egress traffic. When enabled, the bridge allows flooding unknown unicast packets to the specified bridge port, but when disabled, the bridge restricts unknown unicast traffic from being flooded to the specified bridge port. If a MAC address is not learned in the host table, then the traffic is considered as unknown unicast traffic and will be flooded to all ports. MAC address is learned as soon as a packet on a bridge port is received and the source MAC address is added to the bridge host table. Since it is required for the bridge to receive at least one packet on the bridge port to learn the MAC address, it is recommended to use static bridge host entries to avoid packets being dropped until the MAC address has been learned.

RouterOS can handle a maximum of 1024 bridged interfaces, this limit is fixed and cannot be modified. If you try to add more interfaces as bridge ports, it may lead to unpredictable behavior.

Example

To group ether1 and ether2 in the already created bridge1 interface.

```
[admin@MikroTik] /interface bridge port add bridge=bridge1 interface=ether1
[admin@MikroTik] /interface bridge port add bridge=bridge1 interface=ether2
[admin@MikroTik] /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE       BRIDGE       HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0     ether1          bridge1      yes  100     0x80         10                 10       none
 1     ether2          bridge1      yes  200     0x80         10                 10       none
```

Interface lists

Starting with RouterOS v6.41 it possible to add interface lists as a bridge port and sort them. Interface lists are useful for creating simpler firewall rules. Below is an example how to add an interface list to a bridge:

```
/interface list
add name=LAN1
add name=LAN2
/interface list member
add interface=ether1 list=LAN1
add interface=ether2 list=LAN1
add interface=ether3 list=LAN2
add interface=ether4 list=LAN2
/interface bridge port
add bridge=bridge1 interface=LAN1
add bridge=bridge1 interface=LAN2
```

Ports from an interface list added to a bridge will show up as dynamic ports:

```
[admin@MikroTik] /interface bridge port> pr
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE       BRIDGE       HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0     LAN1            bridge1      yes    1     0x80         10                 10       none
 1  D  ether1          bridge1      yes    1     0x80         10                 10       none
 2  D  ether2          bridge1      yes    1     0x80         10                 10       none
 3     LAN2            bridge1      yes    1     0x80         10                 10       none
 4  D  ether3          bridge1      yes    1     0x80         10                 10       none
 5  D  ether4          bridge1      yes    1     0x80         10                 10       none
```

It is also possible to sort the order of lists in which they appear. This can be done using the `move` command. Below is an example of how to sort interface lists:

```
[admin@MikroTik] > /interface bridge port move 3 0
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE       BRIDGE       HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0     LAN2            bridge1      yes    1     0x80         10                 10       none
 1  D  ether3          bridge1      yes    1     0x80         10                 10       none
 2  D  ether4          bridge1      yes    1     0x80         10                 10       none
 3     LAN1            bridge1      yes    1     0x80         10                 10       none
 4  D  ether1          bridge1      yes    1     0x80         10                 10       none
 5  D  ether2          bridge1      yes    1     0x80         10                 10       none
```


The second parameter when moving interface lists is considered as "before id", the second parameter specifies before which interface list should be the selected interface list moved. When moving the first interface list in place of the second interface list, then the command will have no effect since the first list will be moved before the second list, which is the current state either way.

Bridge Port Monitoring

To monitor the current status of bridge ports, use the `monitor` command.

Sub-menu: `/interface bridge port monitor`

Property	Description
designated-bridge (*bridge identifier*)	Shows the received bridge identifier.
designated-cost (*integer*)	Shows the received root-path-cost.
designated-port-number (*integer*)	Shows the received port number.
edge-port (*yes* | *no*)	Whether the port is an edge port or not.
edge-port-discovery (*yes* | *no*)	Whether the port is set to automatically detect edge ports.
external-fdb (*yes* | *no*)	Whether the registration table is used instead of a forwarding database.
forwarding (*yes* | *no*)	Shows if the port is not blocked by (R/M)STP.
hw-offload-group (*switchX*)	Switch chip used by the port.
interface (*name*)	Interface name.
learning (*yes* | *no*)	Shows whether the port is capable of learning MAC addresses.
multicast-router (*yes* | *no*)	Shows if a multicast router is detected on the port. Monitoring value appears only when `igmp-snooping` is enabled.
path-cost (*integer: 1.. 200000000*)	Shows the actual port path-cost. Either manually applied or automatically determined based on the interface speed and the `port-cost-mode` setting.
port-number (*integer 1..4095*)	A port-number will be assigned in the order that ports got added to the bridge, but this is only true until reboot. After reboot, the internal port numbering will be used.
point-to-point-port (*yes* | *no*)	Whether the port is connected to a bridge port using full-duplex (yes) or half-duplex (no).
role (*designated* | *root port* | *alternate* | *backup* | *disabled*)	(R/M)STP algorithm assigned the role of the port: • `disabled-port` - not strictly part of STP, a network administrator can manually disable a port • `root-port` - a forwarding port that is the best port facing towards the root bridge • `alternative-port` - an alternate path to the root bridge • `designated-port` - a forwarding port for every LAN segment • `backup-port` - a backup/redundant path to a segment where another bridge port already connects.
root-path-cost (*integer*)	The total cost of the path to the root-bridge
sending-rstp (*yes* | *no*)	Whether the port is using RSTP or MSTP BPDU types. A port will transit to STP type when RSTP/MSTP enabled port receives an STP BPDU. This settings **does not** indicate whether the BDPUs are actually sent.
status (*in-bridge* | *inactive*)	Port status: • `in-bridge` - port is enabled • `inactive` - port is disabled.

```
[admin@MikroTik] /interface bridge port monitor [find interface=sfp-sfpplus2]
              interface: sfp-sfpplus2
                 status: in-bridge
            port-number: 1
                   role: root-port
              edge-port: no
    edge-port-discovery: yes
    point-to-point-port: yes
           external-fdb: no
           sending-rstp: yes
               learning: yes
             forwarding: yes
              path-cost: 2000
         root-path-cost: 4000
      designated-bridge: 0x8000.DC:2C:6E:9E:11:1C
        designated-cost: 2000
  designated-port-number: 2
```

Hosts Table

MAC addresses that have been learned on a bridge interface can be viewed in the host menu. Below is a table of parameters and flags that can be viewed.

Sub-menu: `/interface bridge host`

Property	Description
bridge (*read-only: name*)	The bridge the entry belongs to
disabled (*read-only: flag*)	Whether the static host entry is disabled
dynamic (*read-only: flag*)	Whether the host has been dynamically created
external (*read-only: flag*)	Whether the host has been learned using an external table, for example, from a switch chip or Wireless registration table. Adding a static host entry on a hardware-offloaded bridge port will also display an active external flag
invalid (*read-only: flag*)	Whether the host entry is invalid, can appear for statically configured hosts on already removed interface
local (*read-only: flag*)	Whether the host entry is created from the bridge itself (that way all local interfaces are shown)
mac-address (*read-only: MAC address*)	Host's MAC address
on-interface (*read-only: name*)	Which of the bridged interfaces the host is connected to

Monitoring

To get the active hosts table:

```
[admin@MikroTik] /interface bridge host print
Flags: X - disabled, I - invalid, D - dynamic, L - local, E - external
 #       MAC-ADDRESS        VID ON-INTERFACE           BRIDGE
 0   D   B8:69:F4:C9:EE:D7      ether1                 bridge1
 1   D   B8:69:F4:C9:EE:D8      ether2                 bridge1
 2   DL  CC:2D:E0:E4:B3:38      bridge1                bridge1
 3   DL  CC:2D:E0:E4:B3:39      ether2                 bridge1
```

Static entries

It is possible to add a static MAC address entry into the host table. This can be used to forward a certain type of traffic through a specific port. Another use case for static host entries is to protect the device resources by disabling dynamic learning and relying only on configured static host entries. Below is a table of possible parameters that can be set when adding a static MAC address entry into the host table.

Sub-menu: `/interface bridge host`

Property	Description
bridge (*name*; Default: **none**)	The bridge interface to which the MAC address is going to be assigned.
disabled (*yes* | *no*; Default: **no**)	Disables/enables static MAC address entry.
interface (*name*; Default: **none**)	Name of the interface.
mac-address (*MAC address*; Default:)	MAC address that will be added to the host table statically.
vid (*integer: 1..4094*; Default:)	VLAN ID for the statically added MAC address entry.

For example, if it was required that all traffic destined to **4C:5E:0C:4D:12:43** is forwarded only through **ether2**, then the following commands can be used:

```
/interface bridge host
add bridge=bridge interface=ether2 mac-address=4C:5E:0C:4D:12:43
```

Multicast Table

When IGMP/MLD snooping is enabled, the bridge will start to listen to IGMP/MLD communication, create multicast database (MDB) entries, and make forwarding decisions based on the received information. Packets with link-local multicast destination addresses 224.0.0.0/24 and ff02::1 are not restricted and are always flooded on all ports and VLANs. To see learned multicast database entries, use the `print` command.

Sub-menu: `/interface bridge mdb`

Property	Description
bridge (*read-only: name*)	Shows the bridge interface the entry belongs to.
group (*read-only: ipv4 | ipv6 address*)	Shows a multicast group address.
on-ports (*read-only: name*)	Shows the bridge ports which are subscribed to the certain multicast group.
vid (*read-only: integer*)	Shows the VLAN ID for the multicast group, only applies when `vlan-filtering` is enabled.

```
[admin@MikroTik] /interface bridge mdb print
Flags: D - DYNAMIC
Columns: GROUP, VID, ON-PORTS, BRIDGE
 #   GROUP              VID  ON-PORTS  BRIDGE
 0 D ff02::2              1  bridge1   bridge1
 1 D ff02::6a             1  bridge1   bridge1
 2 D ff02::1:ff00:0       1  bridge1   bridge1
 3 D ff02::1:ff01:6a43    1  bridge1   bridge1
 4 D 229.1.1.1           10  ether2    bridge1
 5 D 229.2.2.2           10  ether3    bridge1
                             ether2
 6 D ff02::2             10  ether5    bridge1
                             ether3
                             ether2
                             ether4
```

Static entries

Since RouterOS version 7.7, it is possible to create static MDB entries for IPv4 and IPv6 multicast groups.

Sub-menu: `/interface bridge mdb`

Property	Description
bridge (*name*; Default:)	The bridge interface to which the MDB entry is going to be assigned.
disabled (*yes* | *no* ; Default: **no**)	Disables or enables static MDB entry.
group (*ipv4* | *ipv6 address*; Default:)	The IPv4 or IPv6 multicast address. Static entries for link-local multicast groups 224.0.0.0/24 and ff02::1 cannot be created, as these packets are always flooded on all ports and VLANs.
ports (*name*; Default:)	The list of bridge ports to which the multicast group will be forwarded.
vid (*integer: 1.. 4094*; Default:)	The VLAN ID on which the MDB entry will be created, only applies when `vlan-filtering` is enabled. When VLAN ID is not specified, the entry will work in shared-VLAN mode and dynamically apply on all defined VLAN IDs for particular ports.

For example, to create a static MDB entry for multicast group 229.10.10.10 on ports ether2 and ether3 on VLAN 10, use the command below:

```
/interface bridge mdb
add bridge=bridge1 group=229.10.10.10 ports=ether2,ether3 vid=10
```

Verify the results with the `print` command:

```
[admin@MikroTik] > /interface bridge mdb print where group=229.10.10.10
Columns: GROUP, VID, ON-PORTS, BRIDGE
 # GROUP         VID  ON-PORTS  BRIDGE
12 229.10.10.10   10  ether2    bridge1
                      ether3
```

In case a certain IPv6 multicast group does not need to be snooped and it is desired to be flooded on all ports and VLANs, it is possible to create a static MDB entry on all VLANs and ports, including the bridge interface itself. Use the command below to create a static MDB entry for multicast group ff02::2 on all VLANs and ports (modify the `ports` setting for your particular setup):

```
/interface bridge mdb
add bridge=bridge1 group=ff02::2 ports=bridge1,ether2,ether3,ether4,ether5

[admin@MikroTik] > /interface bridge mdb print where group=ff02::2
Flags: D - DYNAMIC
Columns: GROUP, VID, ON-PORTS, BRIDGE
 #   GROUP    VID  ON-PORTS  BRIDGE
 0   ff02::2                bridge1
15 D ff02::2    1  bridge1   bridge1
16 D ff02::2   10  bridge1   bridge1
                   ether2
                   ether3
                   ether4
                   ether5
17 D ff02::2   20  bridge1   bridge1
                   ether2
                   ether3
18 D ff02::2   30  bridge1   bridge1
                   ether2
                   ether3
```

Bridge Hardware Offloading

It is possible to switch multiple ports together if a device has a built-in switch chip. While a bridge is a software feature that will consume CPU's resources, the bridge hardware offloading feature will allow you to use the built-in switch chip to forward packets. This allows you to achieve higher throughput if configured correctly.

In previous versions (prior to RouterOS v6.41) you had to use the master-port property to switch multiple ports together, but in RouterOS v6.41 this property is replaced with the bridge hardware offloading feature, which allows your to switch ports and use some of the bridge features, for example, Spanning Tree Protocol.

> ⚠ When upgrading from previous versions (before RouterOS v6.41), the old master-port configuration is automatically converted to the new **Bridge Hardware Offloading** configuration. When downgrading from newer versions (RouterOS v6.41 and newer) to older versions (before RouterOS v6.41) the configuration is not converted back, a bridge without hardware offloading will exist instead, in such a case you need to reconfigure your device to use the old master-port configuration.

Below is a list of devices and feature that supports hardware offloading (+) or disables hardware offloading (-):

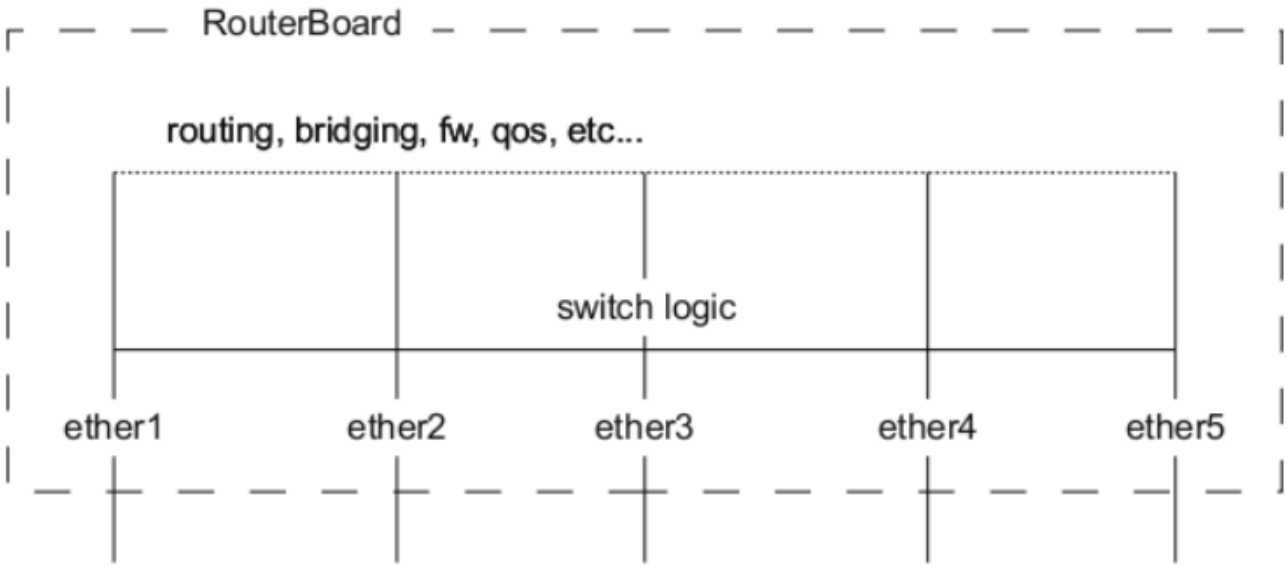
RouterBoard/[Switch Chip] Model	Features in Switch menu	Bridge STP /RSTP	Bridge MSTP	Bridge IGMP Snooping	Bridge DHCP Snooping	Bridge VLAN Filtering	Bonding [4, 5]	Horizon [4]
CRS3xx, CRS5xx series	+	+	+	+	+	+	+	-
CCR2116, CCR2216	+	+	+	+	+	+	+	-
CRS1xx/CRS2xx series	+	+	-	+[2]	+[1]	-	-	-
[QCA8337]	+	+	-	-	+[2]	-	-	-
[Atheros8327]	+	+	-	-	+[2]	-	-	-
[Atheros8316]	+	+	-	-	+[2]	-	-	-
[Atheros8227]	+	+	-	-	-	-	-	-
[Atheros7240]	+	+	-	-	-	-	-	-
[IPQ-PPE]	+[6]	-	-	-	-	-	-	-
[ICPlus175D]	+	-	-	-	-	-	-	-
[MT7621, MT7531]	+	+[3]	+[3]	-	-	+[3]	-	-
[RTL8367]	+	+[3]	+[3]	-	-	+[3]	-	-
[88E6393X, 88E6191X, 88E6190]	+	+	+	+	+	+[3]	+[7]	-

Footnotes:

1. The feature will not work properly in VLAN switching setups. It is possible to correctly snoop DHCP packets only for a single VLAN, but this requires that these DHCP messages get tagged with the correct VLAN tag using an ACL rule, for example, `/interface ethernet switch acl add dst-l3-port=67-68 ip-protocol=udp mac-protocol=ip new-customer-vid=10 src-ports=switch1-cpu`. DHCP Option 82 will not contain any information regarding VLAN-ID.
2. The feature will not work properly in VLAN switching setups.
3. The HW vlan-filtering and R/M/STP was added in the RouterOS 7.1rc1 (for RTL8367) and 7.1rc5 (for MT7621) versions. The switch does not support other ether-type 0x88a8 or 0x9100 (only 0x8100 is supported) and no tag-stacking. Using these features will disable HW offload.
4. The HW offloading will be disabled only for the specific bridge port, not the entire bridge.
5. Only `802.3ad` and `balance-xor` modes can be HW offloaded. Other bonding modes do not support HW offloading.
6. Currently, HW offloaded bridge support for the IPQ-PPE switch chip is still a work in progress. We recommend using, the default, non-HW offloaded bridge (enabled RSTP).
7. The `802.3ad` mode is compatible only with R/M/STP enabled bridge.

> ⚠ When upgrading from older versions (before RouterOS v6.41), only the master-port configuration is converted. For each master-port a bridge will be created. VLAN configuration is not converted and should not be changed, check the Basic VLAN switching guide to be sure how VLAN switching should be configured for your device.

Bridge Hardware Offloading should be considered as port switching, but with more possible features. By enabling hardware offloading you are allowing a built-in switch chip to process packets using its switching logic. The diagram below illustrates that switching occurs before any software related action.



A packet that is received by one of the ports always passes through the switch logic first. Switch logic decides which ports the packet should be going to (most commonly this decision is made based on the destination MAC address of a packet, but there might be other criteria that might be involved based on the packet and the configuration). In most cases the packet will not be visible to RouterOS (only statistics will show that a packet has passed through), this is because the packet was already processed by the switch chip and never reached the CPU.

Though it is possible in certain situations to allow a packet to be processed by the CPU, this is usually called a packet forwarding to the switch CPU port (or the bridge interface in bridge VLAN filtering scenario). This allows the CPU to process the packet and lets the CPU to forward the packet. Passing the packet to the CPU port will give you the opportunity to route packets to different networks, perform traffic control and other software related packet processing actions. To allow a packet to be processed by the CPU, you need to make certain configuration changes depending on your needs and on the device you are using (most commonly passing packets to the CPU are required for VLAN filtering setups). Check the manual page for your specific device:

- CRS1xx/2xx series switches
- CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers
- non-CRS series switches

Certain bridge and Ethernet port properties are directly related to switch chip settings. Changing such properties can trigger a **switch chip reset**, temporarily disabling all Ethernet ports that are on the switch chip for the settings to take effect. This must be taken into account whenever changing properties in production environments. Such properties include DHCP Snooping, IGMP Snooping, VLAN filtering, L2MTU, Flow Control, and others. The exact settings that can trigger a switch chip reset depend on the device's model.

The CRS1xx/2xx series switches support multiple hardware offloaded bridges per switch chip. All other devices support only one hardware offloaded bridge per switch chip. Use the hw=yes/no parameter to select which bridge will use hardware offloading.

Example

Port switching with bridge configuration and enabled hardware offloading:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

Make sure that hardware offloading is enabled and active by checking the "H" flag:

```
[admin@MikroTik] /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE       BRIDGE       HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0   H ether2          bridge1      yes    1     0x80         10                 10       none
 1   H ether3          bridge1      yes    1     0x80         10                 10       none
 2   H ether4          bridge1      yes    1     0x80         10                 10       none
 3   H ether5          bridge1      yes    1     0x80         10                 10       none
```

Port switching in RouterOS v6.41 and newer is done using the bridge configuration. Prior to RouterOS v6.41 port switching was done using the master-port property.

Bridge VLAN Filtering

Bridge VLAN Filtering provides VLAN-aware Layer 2 forwarding and VLAN tag modifications within the bridge. This set of features makes bridge operation more similar to a traditional Ethernet switch and allows overcoming Spanning Tree compatibility issues compared to the configuration when VLAN interfaces are bridged. Bridge VLAN Filtering configuration is highly recommended to comply with STP (IEEE 802.1D), RSTP (IEEE 802.1W) standards and is mandatory to enable MSTP (IEEE 802.1s) support in RouterOS.

The main VLAN setting is `vlan-filtering` which globally controls VLAN-awareness and VLAN tag processing in the bridge. If `vlan-filtering=no` is configured, the bridge ignores VLAN tags, works in a shared-VLAN-learning (SVL) mode, and cannot modify VLAN tags of packets. Turning on `vlan-filtering` enables all bridge VLAN related functionality and independent-VLAN-learning (IVL) mode. Besides joining the ports for Layer2 forwarding, the bridge itself is also an interface therefore it has Port VLAN ID (pvid).

Currently, CRS3xx, CRS5xx series switches, CCR2116, CCR2216 routers and RTL8367, 88E6393X, 88E6191X, 88E6190, MT7621 and MT7531 switch chips (since RouterOS v7) are capable of using bridge VLAN filtering and hardware offloading at the same time, other devices will not be able to use the benefits of a built-in switch chip when bridge VLAN filtering is enabled. Other devices should be configured according to the method described in the Basic VLAN switching guide. If an improper configuration method is used, your device can cause throughput issues in your network.

Bridge VLAN table

Bridge VLAN table represents per-VLAN port mapping with an egress VLAN tag action. The `tagged` ports send out frames with a corresponding VLAN ID tag. The `untagged` ports remove a VLAN tag before sending out frames. Bridge ports with `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged` will be automatically added as untagged ports for the `pvid` VLAN.

Sub-menu: `/interface bridge vlan`

Property	Description
bridge (*name*; Default: **none**)	The bridge interface which the respective VLAN entry is intended for.
disabled (*yes* | *no*; Default: **no**)	Enables or disables Bridge VLAN entry.
tagged (*interfaces*; Default: **none**)	Interface list with a VLAN tag adding action in egress. This setting accepts comma-separated values. e.g. `tagged=ether1,ether2`.
untagged (*interfaces*; Default: **none**)	Interface list with a VLAN tag removing action in egress. This setting accepts comma-separated values. e.g. `untagged=ether3,ether4`
vlan-ids (*integer 1..4094*; Default: **1**)	The list of VLAN IDs for certain port configuration. This setting accepts the VLAN ID range as well as comma-separated values. e.g. `vlan-ids=100-115,120,122,128-130`.

The `vlan-ids` parameter can be used to specify a set or range of VLANs, but specifying multiple VLANs in a single bridge VLAN table entry should only be used for ports that are tagged ports. In case multiple VLANs are specified for access ports, then tagged packets might get sent out as untagged packets through the wrong access port, regardless of the PVID value.

Make sure you have added all needed interfaces to the bridge VLAN table when using bridge VLAN filtering. For routing functions to work properly on the same device through ports that use bridge VLAN filtering, you will need to allow access to the bridge interface (this automatically include a switch-cpu port when HW offloaded vlan-filtering is used, e.g. on CRS3xx series switches), this can be done by adding the bridge interface itself to the VLAN table, for tagged traffic you will need to add the bridge interface as a tagged port and create a VLAN interface on the bridge interface. Examples can be found in the inter-VLAN routing and Management port sections.

When allowing access to the CPU, you are allowing access from a certain port to the actual router/switch, this is not always desirable. Make sure you implement proper firewall filter rules to secure your device when access to the CPU is allowed from a certain VLAN ID and port, use firewall filter rules to allow access to only certain services.

Improperly configured bridge VLAN filtering can cause security issues, make sure you fully understand how Bridge VLAN table works before deploying your device into production environments.

Bridge port settings

Each bridge port have multiple VLAN related settings, that can change untagged VLAN membership, VLAN tagging/untagging behavior and packet filtering based on VLAN tag presence.

Sub-menu: `/interface bridge port`

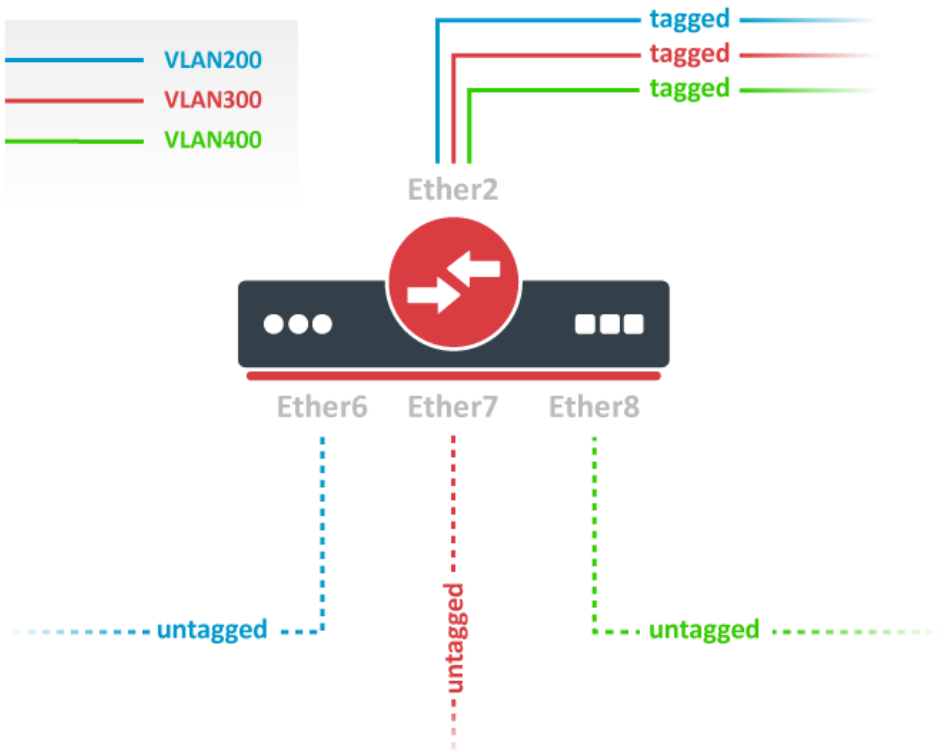
Property	Description
frame-types (*admit-all* | *admit-only-untagged-and-priority-tagged* | *admit-only-vlan-tagged*; Default: **admit-all**)	Specifies allowed ingress frame types on a bridge port. This property only has an effect when `vlan-filtering` is set to `yes`.
ingress-filtering (*yes* | *no*; Default: **yes**)	Enables or disables VLAN ingress filtering, which checks if the ingress port is a member of the received VLAN ID in the bridge VLAN table. Should be used with `frame-types` to specify if the ingress traffic should be tagged or untagged. This property only has effect when `vlan-filtering` is set to `yes`. The setting is enabled by default since RouterOS v7.
pvid (*integer 1..4094*; Default: **1**)	Port VLAN ID (pvid) specifies which VLAN the untagged ingress traffic is assigned to. This property only has an effect when `vlan-filtering` is set to `yes`.
tag-stacking (*yes* | *no*; Default: **no**)	Forces all packets to be treated as untagged packets. Packets on ingress port will be tagged with another VLAN tag regardless if a VLAN tag already exists, packets will be tagged with a VLAN ID that matches the `pvid` value and will use EtherType that is specified in `ether-type`. This property only has effect when `vlan-filtering` is set to `yes`.

Bridge host table

Bridge host table allows monitoring learned MAC addresses. When `vlan-filtering` is enabled, it shows learned VLAN ID as well (enabled independent-VLAN-learning or IVL).

```
[admin@MikroTik] > /interface bridge host print where !local
Flags: X - disabled, I - invalid, D - dynamic, L - local, E - external
 #       MAC-ADDRESS        VID ON-INTERFACE       BRIDGE
 0   D   CC:2D:E0:E4:B3:AA  300 ether3            bridge1
 1   D   CC:2D:E0:E4:B3:AB  400 ether4            bridge1
```

VLAN Example - Trunk and Access Ports



Create a bridge with disabled `vlan-filtering` to avoid losing access to the device before VLANs are completely configured. If you need a management access to the bridge, see the Management access configuration section.

```
/interface bridge
add name=bridge1 vlan-filtering=no
```

Add bridge ports and specify `pvid` for access ports to assign their untagged traffic to the intended VLAN. Use `frame-types` setting to accept only tagged or untagged packets.

```
/interface bridge port
add bridge=bridge1 interface=ether2 frame-types=admit-only-vlan-tagged
add bridge=bridge1 interface=ether6 pvid=200 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether7 pvid=300 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether8 pvid=400 frame-types=admit-only-untagged-and-priority-tagged
```

Add Bridge VLAN entries and specify tagged ports in them. Bridge ports with `frame-types` set to `admit-only-untagged-and-priority-tagged` will be automatically added as untagged ports for the `pvid` VLAN.

```
/interface bridge vlan
add bridge=bridge1 tagged=ether2 vlan-ids=200
add bridge=bridge1 tagged=ether2 vlan-ids=300
add bridge=bridge1 tagged=ether2 vlan-ids=400
```

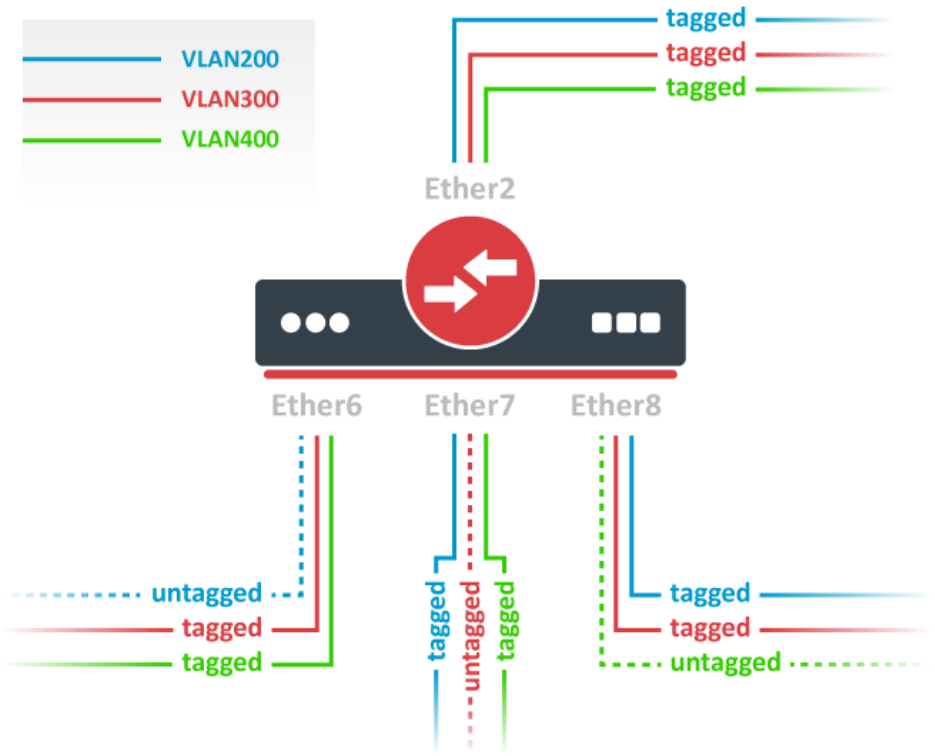
In the end, when VLAN configuration is complete, enable Bridge VLAN Filtering.

```
/interface bridge set bridge1 vlan-filtering=yes
```

Optional step is to set `frame-types=admit-only-vlan-tagged` on the bridge interface in order to disable the default untagged VLAN 1 (`pvid=1`).

```
/interface bridge set bridge1 frame-types=admit-only-vlan-tagged
```

VLAN Example - Trunk and Hybrid Ports



Create a bridge with disabled `vlan-filtering` to avoid losing access to the router before VLANs are completely configured. If you need a management access to the bridge, see the Management access configuration section.

```
/interface bridge
add name=bridge1 vlan-filtering=no
```

Add bridge ports and specify `pvid` on hybrid VLAN ports to assign untagged traffic to the intended VLAN. Use `frame-types` setting to accept only tagged packets on ether2.

```
/interface bridge port
add bridge=bridge1 interface=ether2 frame-types=admit-only-vlan-tagged
add bridge=bridge1 interface=ether6 pvid=200
add bridge=bridge1 interface=ether7 pvid=300
add bridge=bridge1 interface=ether8 pvid=400
```

Add Bridge VLAN entries and specify tagged ports in them. In this example egress VLAN tagging is done on ether6,ether7,ether8 ports too, making them into hybrid ports. Bridge ports with `frame-types` set to `admit-all` will be automatically added as untagged ports for the `pvid` VLAN.

```
/interface bridge vlan
add bridge=bridge1 tagged=ether2,ether7,ether8 vlan-ids=200
add bridge=bridge1 tagged=ether2,ether6,ether8 vlan-ids=300
add bridge=bridge1 tagged=ether2,ether6,ether7 vlan-ids=400
```

In the end, when VLAN configuration is complete, enable Bridge VLAN Filtering.

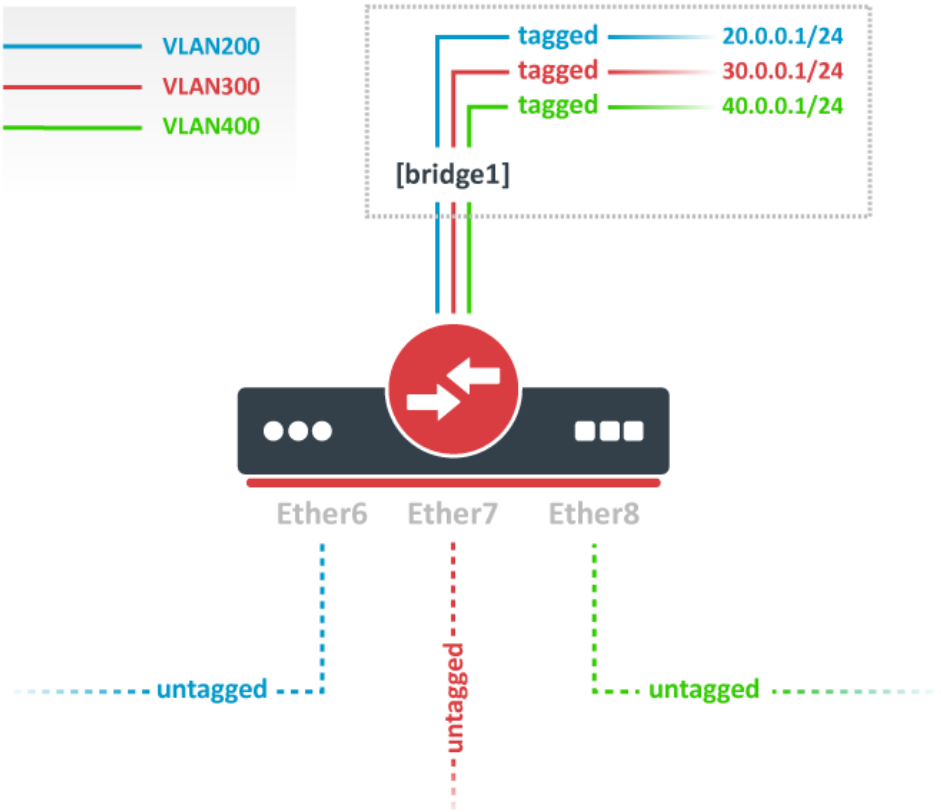
```
/interface bridge set bridge1 vlan-filtering=yes
```

Optional step is to set `frame-types=admit-only-vlan-tagged` on the bridge interface in order to disable the default untagged VLAN 1 (`pvid=1`).

```
/interface bridge set bridge1 frame-types=admit-only-vlan-tagged
```

> You don't have to add access ports as untagged ports, because they will be added dynamically as an untagged port with the VLAN ID that is specified in `pvid`, you can specify just the trunk port as a tagged port. All ports that have the same `pvid` set will be added as untagged ports in a single entry. You must take into account that the bridge itself is a port and it also has a `pvid` value, this means that the bridge port also will be added as an untagged port for the ports that have the same `pvid`. You can circumvent this behavior by either setting different `pvid` on all ports (even the trunk port and bridge itself), or to use `frame-type` set to `accept-only-vlan-tagged`.

VLAN Example - InterVLAN Routing by Bridge



Create a bridge with disabled `vlan-filtering` to avoid losing access to the router before VLANs are completely configured. If you need a management access to the bridge, see the Management access configuration section.

```
/interface bridge
add name=bridge1 vlan-filtering=no
```

Add bridge ports and specify `pvid` for VLAN access ports to assign their untagged traffic to the intended VLAN. Use `frame-types` setting to accept only untagged packets.

```
/interface bridge port
add bridge=bridge1 interface=ether6 pvid=200 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether7 pvid=300 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether8 pvid=400 frame-types=admit-only-untagged-and-priority-tagged
```

Add Bridge VLAN entries and specify tagged ports in them. In this example **bridge1** interface is the VLAN trunk that will send traffic further to do InterVLAN routing. Bridge ports with `frame-types` set to `admit-only-untagged-and-priority-tagged` will be automatically added as untagged ports for the `pvid` VLAN.

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1 vlan-ids=200
add bridge=bridge1 tagged=bridge1 vlan-ids=300
add bridge=bridge1 tagged=bridge1 vlan-ids=400
```

Configure VLAN interfaces on the **bridge1** to allow handling of tagged VLAN traffic at routing level and set IP addresses to ensure routing between VLANs as planned.

```
/interface vlan
add interface=bridge1 name=VLAN200 vlan-id=200
add interface=bridge1 name=VLAN300 vlan-id=300
add interface=bridge1 name=VLAN400 vlan-id=400

/ip address
add address=20.0.0.1/24 interface=VLAN200
add address=30.0.0.1/24 interface=VLAN300
add address=40.0.0.1/24 interface=VLAN400
```

In the end, when VLAN configuration is complete, enable Bridge VLAN Filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```

Optional step is to set `frame-types=admit-only-vlan-tagged` on the bridge interface in order to disable the default untagged VLAN 1 (`pvid=1`).

```
/interface bridge set bridge1 frame-types=admit-only-vlan-tagged
```

Since RouterOS v7, it is possible to route traffic using the L3 HW offloading on certain devices. See more details on L3 Hardware Offloading.

Management access configuration

There are multiple ways to set up management access on a device that uses bridge VLAN filtering. Below are some of the most popular approaches to properly enable access to a router/switch. Start by creating a bridge without VLAN filtering enabled:

```
/interface bridge
add name=bridge1 vlan-filtering=no
```

Untagged access without VLAN filtering

In case VLAN filtering will not be used and access with untagged traffic is desired, the only requirement is to create an IP address on the bridge interface.

```
/ip address
add address=192.168.99.1/24 interface=bridge1
```

Tagged access without VLAN filtering

In case VLAN filtering will not be used and access with tagged traffic is desired, create a routable VLAN interface on the bridge and add an IP address on the VLAN interface.

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT
```

Tagged access with VLAN filtering

In case VLAN filtering is used and access with tagged traffic is desired, additional steps are required. In this example, VLAN 99 will be used to access the device. A VLAN interface on the bridge must be created and an IP address must be assigned to it.

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT
```

For example, if you want to allow access to the device from ports **ether3**, **ether4, sfp-sfpplus1** using tagged VLAN 99 traffic, then you must add this entry to the VLAN table. Note that the **bridge1** interface is also included in the tagged port list:

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,ether3,ether4,sfp-sfpplus1 vlan-ids=99
```

After that you can enable VLAN filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```

Untagged access with VLAN filtering

In case VLAN filtering is used and access with untagged traffic is desired, the VLAN interface must use the same VLAN ID as the untagged port VLAN ID (`pvid`). Just like in the previous example, start by creating a VLAN interface on the bridge and add an IP address for the VLAN.

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT
```

For example, untagged ports **ether2** and **ether3** should be able to communicate with the VLAN 99 interface using untagged traffic. In order to achieve this, these ports should be configured with the `pvid` that matches the VLAN ID on management VLAN. Note that the **bridge1** interface is a tagged port member, you can configure additional tagged ports if necessary (see the previous example).

```
/interface bridge port
set [find interface=ether2] pvid=99
set [find interface=ether3] pvid=99
/interface bridge vlan
add bridge=bridge1 tagged=bridge1 untagged=ether2,ether3 vlan-ids=99
```

After that you can enable VLAN filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```

Changing untagged VLAN for the bridge interface

In case VLAN filtering is used, it is possible to change the untagged VLAN ID for the bridge interface using the `pvid` setting. Note that creating routable VLAN interfaces and allowing tagged traffic on the bridge is a more flexible and generally recommended option.

First, create an IP address on the bridge interface.

```
/ip address
add address=192.168.99.1/24 interface=bridge1
```

For example, untagged **bridge1** traffic should be able to communicate with untagged **ether2** and **ether3** ports and tagged **sfp-sfpplus1** port in VLAN 99. In order to achieve this, **bridge1**, **ether2**, **ether3** should be configured with the same `pvid` and sfp-sfpplus1 added as a tagged member.

```
/interface bridge
set [find name=bridge1] pvid=99
/interface bridge port
set [find interface=ether2] pvid=99
set [find interface=ether3] pvid=99
/interface bridge vlan
add bridge=bridge1 tagged=sfp-sfpplus1 untagged=bridge1,ether2,ether3 vlan-ids=99
```

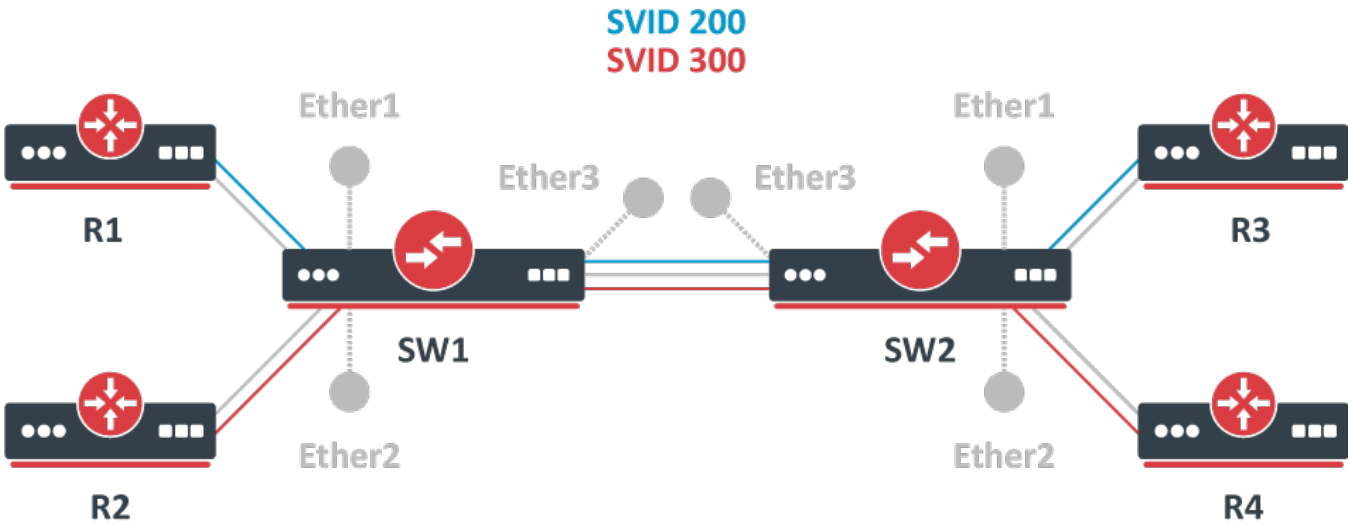
After that you can enable VLAN filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```

> ⚠ If the connection to the router/switch through an IP address is not required, then steps adding an IP address can be skipped since a connection to the router/switch through Layer2 protocols (e.g. MAC-telnet) will be working either way.

VLAN Tunneling (QinQ)

Since RouterOS v6.43 the RouterOS bridge is IEEE 802.1ad compliant and it is possible to filter VLAN IDs based on Service VLAN ID (0x88a8) rather than Customer VLAN ID (0x8100). The same principles can be applied as with IEEE 802.1Q VLAN filtering (the same setup examples can be used). Below is a topology for a common **Provider bridge**:



In this example, **R1**, **R2**, **R3,** and **R4** might be sending any VLAN tagged traffic by 802.1Q (CVID), but **SW1** and **SW2** needs isolate traffic between routers in a way that **R1** is able to communicate only with **R3**, and **R2** is only able to communicate with **R4**. To do so, you can tag all ingress traffic with an SVID and only allow these VLANs on certain ports. Start by enabling the service tag 0x88a8, introduced by `802.1ad`, on the bridge. Use these commands on **SW1** and **SW2**:

```
/interface bridge
add name=bridge1 vlan-filtering=no ether-type=0x88a8
```

In this setup, **ether1** and **ether2** are going to be access ports (untagged), use the `pvid` parameter to tag all ingress traffic on each port, use these commands on **SW1** and **SW2**:

```
/interface bridge port
add interface=ether1 bridge=bridge1 pvid=200
add interface=ether2 bridge=bridge1 pvid=300
add interface=ether3 bridge=bridge1
```

Specify tagged and untagged ports in the bridge VLAN table, use these commands on **SW1** and **SW2**:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether3 untagged=ether1 vlan-ids=200
add bridge=bridge1 tagged=ether3 untagged=ether2 vlan-ids=300
```

When the bridge VLAN table is configured, you can enable bridge VLAN filtering, use these commands on **SW1** and **SW2:**

```
/interface bridge set bridge1 vlan-filtering=yes
```

By enabling vlan-filtering you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

Note, that if you are using the new EtherType/TPID 0x88a8 (service tag) and you also need a VLAN interface for your Service VLAN, you will also have to apply the `use-service-tag` parameter on the VLAN interface.

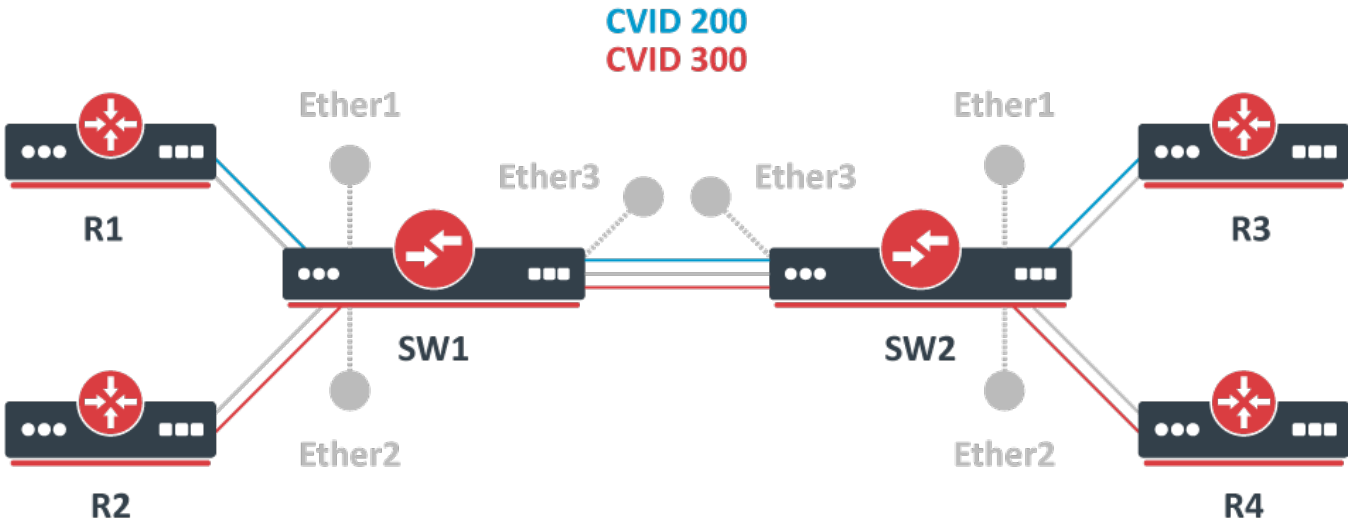
When `ether-type=0x8100` is configured, the bridge checks the outer VLAN tag and sees if it is using EtherType `0x8100`. If the bridge receives a packet with an outer tag that has a different EtherType, it will mark the packet as `untagged`. Since RouterOS only checks the outer tag of a packet, it is not possible to filter 802.1Q packets when the 802.1ad protocol is used.

Currently, only CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers are capable of hardware offloaded VLAN filtering using the Service tag, EtherType/TPID `0x88a8`.

Devices with switch chip Marvell-98DX3257 (e.g. CRS354 series) do not support VLAN filtering on 1Gbps Ethernet interfaces for other VLAN types (`0x88a8` and `0x9100`).

Tag stacking

Since RouterOS v6.43 it is possible to forcefully add a new VLAN tag over any existing VLAN tags, this feature can be used to achieve a CVID stacking setup, where a CVID (0x8100) tag is added before an existing CVID tag. This type of setup is very similar to the Provider bridge setup, to achieve the same setup but with multiple CVID tags (CVID stacking) we can use the same topology:



In this example **R1**, **R2**, **R3,** and **R4** might be sending any VLAN tagged traffic, it can be 802.1ad, 802.1Q or any other type of traffic, but **SW1** and **SW2** needs isolate traffic between routers in a way that **R1** is able to communicate only with **R3**, and **R2** is only able to communicate with **R4**. To do so, you can tag all ingress traffic with a new CVID tag and only allow these VLANs on certain ports. Start by selecting the proper EtherType, use these commands on **SW1** and **SW2**:

```
/interface bridge
add name=bridge1 vlan-filtering=no ether-type=0x8100
```

In this setup, **ether1** and **ether2** will ignore any VLAN tags that are present and add a new VLAN tag, use the `pvid` parameter to tag all ingress traffic on each port and allow `tag-stacking` on these ports, use these commands on **SW1** and **SW2**:

```
/interface bridge port
add interface=ether1 bridge=bridge1 pvid=200 tag-stacking=yes
add interface=ether2 bridge=bridge1 pvid=300 tag-stacking=yes
add interface=ether3 bridge=bridge1
```

Specify tagged and untagged ports in the bridge VLAN table, you only need to specify the VLAN ID of the outer tag, use these commands on **SW1** and **SW2**:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether3 untagged=ether1 vlan-ids=200
add bridge=bridge1 tagged=ether3 untagged=ether2 vlan-ids=300
```

When the bridge VLAN table is configured, you can enable bridge VLAN filtering, which is required in order for the `pvid` parameter to have any effect, use these commands on **SW1** and **SW2:**

```
/interface bridge set bridge1 vlan-filtering=yes
```

> By enabling vlan-filtering you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

MVRP

Multiple VLAN Registration protocol (MVRP) is a protocol based on Multiple Registration Protocol (MRP) which allows to register attributes (VLAN IDs in case of MVRP) with other members of Bridged LAN.

An MRP application can make or withdraw declarations of attributes which result in registration or leaving of those attributes with other MRP participants.

Here's how it works.

MRP consists of two parts:

- **Applicant** - responsible for sending declarations (or leaves). Its behavior can be configured on a per-port basis using the setting called `mvrp-applicant-state`, and per-VLAN using the `mvrp-forbidden` setting.
- **Registrar** - responsible for registering incoming declarations. Its configuration can be set per-port using the `mvrp-registrar-state` setting, and per-VLAN using the `mvrp-forbidden` setting.

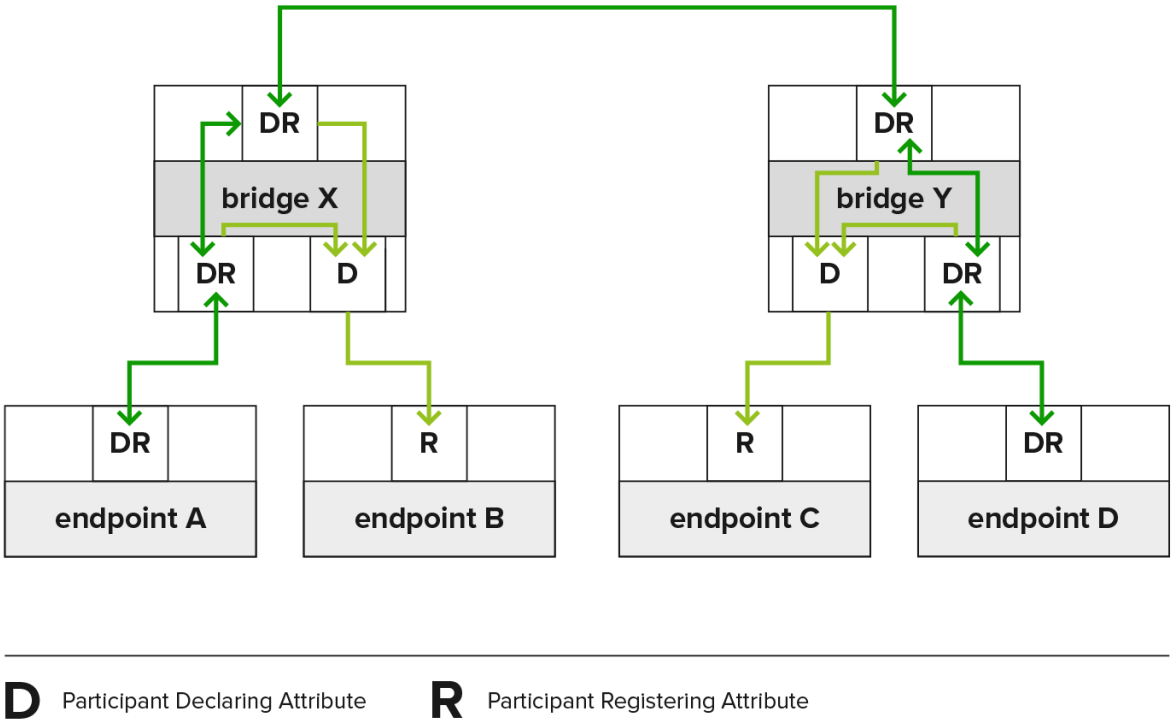
Registration Propagation: Incoming registration on a bridge port dynamically makes that specific port a tagged VLAN member. Additionally, the attributes associated with this registration are spread to all active (forwarding) bridge ports as a declaration.

Declaration Operation: In case of MVRP, the configured VLAN's get declared on each port, but they will only get configured as members of those VLAN's when a declaration is received from the LAN (Registrar will register the VLAN). From the perspective of an end-station, a single declaration will be registered on each upstream port across the entire LAN. When another end-station declares the same attribute, a path of registrations will be made between the two (or more) end stations, see the picture below.

MVRP helps to dynamically propagate VLAN information throughout the bridged network and configure VLANs only on the needed ports. This makes the network efficient by avoiding unnecessary traffic flooding.

As noted before, MVRP is only active on ports that are forwarding. In case of MSTP declarations and registrations are made only if the port is forwarding in the MSTI in which VLAN is mapped.

The point-to-point ports speed up the process of registration (or leaving). Manually configuring `point-to-point=yes` can be advantageous for non-Ethernet interfaces.



Property Reference

Sub-menu: `/interface bridge`

Property	Description
mvrp (*yes* | *no*; Default: **no**)	Enables MVRP for bridge. It ensures that the MAC address 01:80:C2:00:00:21 is trapped and not forwarded, the `vlan-filtering` must be enabled.

Sub-menu: `/interface bridge port`

The port menu enables control over the applicant and registrar settings on a per-port basis.

Property	Description
mvrp-applicant-state (*non-participant* | *normal-participant;* Default: **normal-participant**)	MVRP applicant options: • **non-participant** - port does not send any MRP messages; • **normal-participant** - port participates normally in MRP exchanges.

mvrp-registrar-state (*fixed* | *normal*; Default: **normal**)	MVRP registrar options: • **fixed** - port ignores all MRP messages, and remains Registered (IN) in all configured vlans. • **normal** - port receives MRP messages and handles them according to the standard.

To monitor the currently declared and registered VLAN IDs, use the `monitor` command.

```
[admin@MikroTik] > interface/bridge/port monitor [find interface=sfp-sfpplus1]
            interface: sfp-sfpplus1
               status: in-bridge
          port-number: 1
                 role: designated-port
            edge-port: no
  edge-port-discovery: yes
  point-to-point-port: yes
         external-fdb: no
         sending-rstp: yes
             learning: yes
           forwarding: yes
     actual-path-cost: 2000
     hw-offload-group: switch1
    declared-vlan-ids: 1,10,20-21
  registered-vlan-ids: 1,10,20,30-33
```

Sub-menu: `/interface bridge vlan`

All ports that are members of static VLANs or dynamic untagged VLANs created by the port `pvid` setting are treated as "fixed." Meaning the registrar disregards all MRP messages and remains registered (IN) for those VLANs.

When VLAN is neither manually configured nor created by the port `pvid` setting, incoming registrations on a bridge port can dynamically designate that specific port as a tagged VLAN member. The `mvrp-forbidden` feature allows creating a list of ports that are restricted from registering into a specific VLAN ID.

VLANs that are static or dynamic will be declared by the applicants unless this functionality is disabled by the port's `mvrp-applicant-state`, or by VLAN's `mvrp-forbidden` setting.

Property	Description
mvrp-forbidden (*interfaces*; Default:)	Ports that ignore all MRP messages and remains Not Registered (MT), as well as disables applicant from declaring specific VLAN ID.

Sub-menu: `/interface bridge vlan mvrp`

The MVRP attributes menu can be used to see internal MVRP attribute states, as specified in the IEEE 802.1Q-2011.

Property	Description

applicant-state	The Applicant state machine that declares attributes. Its state can be VO, VP, VN, AN, AA, QA, LA, AO, QO, AP, QP, or LO. Each state consists of two letters. The first letter indicates the state: • V—Very anxious; • A—Anxious; • Q—Quiet; • L—Leaving. The second letter indicates the membership state: • A - Active member; • P - Passive member; • O - Observer; • N - New. For example, VP indicates "Very anxious, Passive member."
registrar-state	The Registrar state machine that records the registration state of attributes declared by other participants. Its state can be IN, LV, or MT: • IN—Registered; • LV—Previously registered, but now being timed out; • MT—Not registered.

```
[admin@Mikrotik] /interface/bridge/vlan/mvrp print where vlan-id=10
Columns: BRIDGE, PORT, VLAN-ID, REGISTRAR-STATE, APPLICANT-STATE, LAST-EVENT
 #  BRIDGE    PORT           VLAN-ID  REGISTRAR-STATE  APPLICANT-STATE  LAST-EVENT
 1  bridge67  sfp-sfpplus1        10  IN               Quiet Active     JoinIn
 9  bridge67  sfp-sfpplus5        10  MT               Quiet Active     JoinEmpty
17  bridge67  sfp-sfpplus9        10  MT               Quiet Active     JoinEmpty
25  bridge67  sfp-sfpplus13       10  IN               Quiet Active     JoinIn
```

Fast Forward

Fast Forward allows forwarding packets faster under special conditions. When Fast Forward is enabled, then the bridge can process packets even faster since it can skip multiple bridge-related checks, including MAC learning. Below you can find a list of conditions that **MUST** be met in order for Fast Forward to be active:

- Bridge has `fast-forward` set to `yes`
- Bridge has only 2 running ports
- Both bridge ports support Fast Path, Fast Path is active on ports and globally on the bridge
- Bridge Hardware Offloading is disabled
- Bridge VLAN Filtering is disabled
- Bridge DHCP snooping is disabled
- `unknown-multicast-flood` is set to `yes`
- `unknown-unicast-flood` is set to `yes`
- `broadcast-flood` is set to `yes`
- MAC address for the bridge matches with a MAC address from one of the bridge slave ports
- `horizon` for both ports is set to `none`

⚠ Fast Forward disables MAC learning, this is by design to achieve faster packet forwarding. MAC learning prevents traffic from flooding multiple interfaces, but MAC learning is not needed when a packet can only be sent out through just one interface.

Fast Forward is disabled when hardware offloading is enabled. Hardware offloading can achieve full write-speed performance when it is active since it will use the built-in switch chip (if such exists on your device), fast forward uses the CPU to forward packets. When comparing throughput results, you would get such results: Hardware offloading > Fast Forward > Fast Path > Slow Path.

It is possible to check how many packets where processed by Fast Forward:

```
[admin@MikroTik] /interface bridge settings> pr
              use-ip-firewall: no
     use-ip-firewall-for-vlan: no
    use-ip-firewall-for-pppoe: no
              allow-fast-path: yes
      bridge-fast-path-active: yes
     bridge-fast-path-packets: 0
       bridge-fast-path-bytes: 0
  bridge-fast-forward-packets: 16423
    bridge-fast-forward-bytes: 24864422
```

If packets are processed by Fast Path, then Fast Forward is not active. Packet count can be used as an indicator of whether Fast Forward is active or not.

Since RouterOS 6.44 it is possible to monitor Fast Forward status, for example:

```
[admin@MikroTik] /interface bridge monitor bridge1
                  state: enabled
    current-mac-address: B8:69:F4:C9:EE:D7
            root-bridge: yes
         root-bridge-id: 0x8000.B8:69:F4:C9:EE:D7
         root-path-cost: 0
              root-port: none
             port-count: 2
  designated-port-count: 2
           fast-forward: yes
```

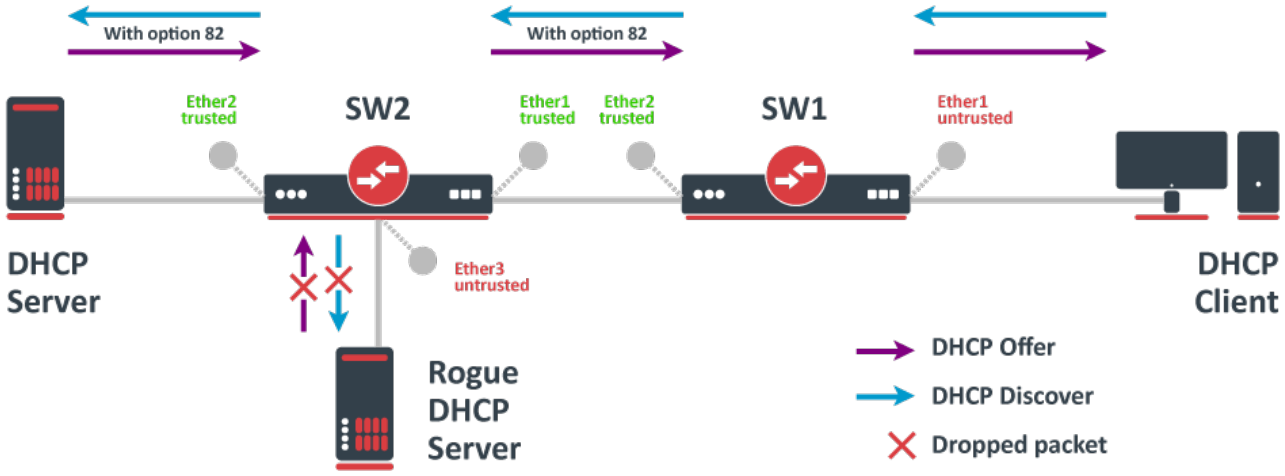
Disabling or enabling fast-forward will temporarily disable all bridge ports for settings to take effect. This must be taken into account whenever changing this property on production environments since it can cause all packets to be temporarily dropped.

IGMP/MLD Snooping

The bridge supports IGMP/MLD snooping. It controls multicast streams and prevents multicast flooding on unnecessary ports. Its settings are placed in the bridge menu and it works independently in every bridge interface. Software-driven implementation works on all devices with RouterOS, but CRS3xx, CRS5xx series switches, CCR2116, CR2216 routers, and 88E6393X, 88E6191X, 88E6190 switch chips also support IGMP/MLD snooping with hardware offloading. See more details on IGMP/MLD snooping manual.

DHCP Snooping and DHCP Option 82

DHCP Snooping and DHCP Option 82 is supported by bridge. The DHCP Snooping is a Layer2 security feature, that limits unauthorized DHCP servers from providing malicious information to users. In RouterOS, you can specify which bridge ports are trusted (where known DHCP server resides and DHCP messages should be forwarded) and which are untrusted (usually used for access ports, received DHCP server messages will be dropped). The DHCP Option 82 is additional information (Agent Circuit ID and Agent Remote ID) provided by DHCP Snooping enabled devices that allow identifying the device itself and DHCP clients.



In this example, SW1 and SW2 are DHCP Snooping, and Option 82 enabled devices. First, we need to create a bridge, assign interfaces and mark trusted ports. Use these commands on **SW1**:

```
/interface bridge
add name=bridge
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2 trusted=yes
```

For SW2, the configuration will be similar, but we also need to mark ether1 as trusted, because this interface is going to receive DHCP messages with Option 82 already added. You need to mark all ports as trusted if they are going to receive DHCP messages with added Option 82, otherwise these messages will be dropped. Also, we add ether3 to the same bridge and leave this port untrusted, imagine there is an unauthorized (rogue) DHCP server. Use these commands on **SW2**:

```
/interface bridge
add name=bridge
/interface bridge port
add bridge=bridge interface=ether1 trusted=yes
add bridge=bridge interface=ether2 trusted=yes
add bridge=bridge interface=ether3
```

Then we need to enable DHCP Snooping and Option 82. In case your DHCP server does not support DHCP Option 82 or you do not implement any Option 82 related policies, this option can be disabled. Use these commands on **SW1** and **SW2**:

```
/interface bridge
set [find where name="bridge"] dhcp-snooping=yes add-dhcp-option82=yes
```

Now both devices will analyze what DHCP messages are received on bridge ports. The **SW1** is responsible for adding and removing the DHCP Option 82. The **SW2** will limit rogue DHCP server from receiving any discovery messages and drop malicious DHCP server messages from ether3.

⚠ Currently, CRS3xx, CRS5xx series switches, CCR2116, CR2216 routers, and 88E6393X, 88E6191X, 88E6190 switch chips fully support hardware offloaded DHCP Snooping and Option 82. For CRS1xx and CRS2xx series switches it is possible to use DHCP Snooping along with VLAN switching, but then you need to make sure that DHCP packets are sent out with the correct VLAN tag using egress ACL rules. Other devices are capable of using DHCP Snooping and Option 82 features along with hardware offloading, but you must make sure that there is no VLAN-related configuration applied on the device, otherwise, DHCP Snooping and Option 82 might not work properly. See the Br idge Hardware Offloading section with supported features.

For CRS3xx, CRS5xx series switches and CCR2116, CR2216 routers DHCP snooping will not work when hardware offloading bonding interfaces are created.

Controller Bridge and Port Extender

Controller Bridge (CB) and Port Extender (PE) is an IEEE 802.1BR standard implementation in RouterOS for CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers. It allows virtually extending the CB ports with a PE device and managing these extended interfaces from a single controlling device. Such configuration provides a simplified network topology, flexibility, increased port density, and ease of manageability. See more details on Controller Bridge and Port Extender manual.

Bridge Firewall

The bridge firewall implements packet filtering and thereby provides security functions that are used to manage data flow to, from, and through the bridge.

Packet flow diagram shows how packets are processed through the router. It is possible to force bridge traffic to go through `/ip firewall filter` rules (see the bridge settings).

There are two bridge firewall tables:

- **filter** - bridge firewall with three predefined chains:
  - **input** - filters packets, where the destination is the bridge (including those packets that will be routed, as they are destined to the bridge MAC address anyway)
  - **output** - filters packets, which come from the bridge (including those packets that has been routed normally)
  - **forward** - filters packets, which are to be bridged (note: this chain is not applied to the packets that should be routed through the router, just to those that are traversing between the ports of the same bridge)
- **nat** - bridge network address translation provides ways for changing source/destination MAC addresses of the packets traversing a bridge. Has two built-in chains:
  - **srcnat** - used for "hiding" a host or a network behind a different MAC address. This chain is applied to the packets leaving the router through a bridged interface
  - **dstnat** - used for redirecting some packets to other destinations

You can put packet marks in bridge firewall (filter and NAT), which are the same as the packet marks in IP firewall configured by '`/ip firewall mangle`'. In this way, packet marks put by bridge firewall can be used in 'IP firewall', and vice versa.

General bridge firewall properties are described in this section. Some parameters that differ between nat and filter rules are described in further sections.

Sub-menu: `/interface bridge filter, /interface bridge nat`

Property	Description
802.3-sap (*integer*; Default:)	DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) are 2 one-byte fields, which identify the network protocol entities which use the link-layer service. These bytes are always equal. Two hexadecimal digits may be specified here to match an SAP byte.
802.3-type (*integer*; Default:)	Ethernet protocol type, placed after the IEEE 802.2 frame header. Works only if 802.3-sap is 0xAA (SNAP - Sub-Network Attachment Point header). For example, AppleTalk can be indicated by the SAP code of 0xAA followed by a SNAP type code of 0x809B.

action (*accept | drop | jump | log | mark-packet | passthrough | return | set-priority*; Default:)	Action to take if the packet is matched by the rule: • accept - accept the packet. The packet is not passed to the next firewall rule • drop - silently drop the packet • jump - jump to the user-defined chain specified by the value of `jump-target` parameter • log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After the packet is matched it is passed to the next rule in the list, similar as `passthrough` • mark-packet - place a mark specified by the new-packet-mark parameter on a packet that matches the rule • passthrough - if the packet is matched by the rule, increase counter and go to next rule (useful for statistics) • return - passes control back to the chain from where the jump took place • set-priority - set priority specified by the new-priority parameter on the packets sent out through a link that is capable of transporting priority (VLAN or WMM-enabled wireless interface). Read more
arp-dst-address (*IP address*; Default:)	ARP destination IP address.
arp-dst-mac-address (*MAC address*; Default:)	ARP destination MAC address.
arp-gratuitous (*yes | no*; Default:)	Matches ARP gratuitous packets.
arp-hardware-type (*integer*; Default: **1**)	ARP hardware type. This is normally Ethernet (Type 1).
arp-opcode (*arp-nak | drarp-error | drarp-reply | drarp-request | inarp-reply | inarp-request | reply | reply-reverse | request | request-reverse*; Default:)	ARP opcode (packet type) • arp-nak - negative ARP reply (rarely used, mostly in ATM networks) • drarp-error - Dynamic RARP error code, saying that an IP address for the given MAC address can not be allocated • drarp-reply - Dynamic RARP reply, with a temporary IP address assignment for a host • drarp-request - Dynamic RARP request to assign a temporary IP address for the given MAC address • inarp-reply - InverseARP Reply • inarp-request - InverseARP Request • reply - standard ARP reply with a MAC address • reply-reverse - reverse ARP (RARP) reply with an IP address assigned • request - standard ARP request to a known IP address to find out unknown MAC address • request-reverse - reverse ARP (RARP) request to a known MAC address to find out the unknown IP address (intended to be used by hosts to find out their own IP address, similarly to DHCP service)
arp-packet-type (*integer 0..65535 | hex 0x0000-0xffff*; Default:)	ARP Packet Type.
arp-src-address (*IP address*; Default:)	ARP source IP address.
arp-src-mac-address (*MAC addres*; Default:)	ARP source MAC address.
chain (*text*; Default:)	Bridge firewall chain, which the filter is functioning in (either a built-in one, or a user-defined one).
dst-address (*IP address*; Default:)	Destination IP address (only if MAC protocol is set to IP).
dst-address6 (*IPv6 address*; Default:)	Destination IPv6 address (only if MAC protocol is set to IPv6).
dst-mac-address (*MAC address*; Default:)	Destination MAC address.

dst-port (*integer 0..65535*; Default:)	Destination port number or range (only for TCP or UDP protocols).
in-bridge (*name*; Default:)	Bridge interface through which the packet is coming in.
in-bridge-list (*name*; Default:)	Set of bridge interfaces defined in interface list. Works the same as `in-bridge`.
in-interface (*name*; Default:)	Physical interface (i.e., bridge port) through which the packet is coming in.
in-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as `in-interface`.
ingress-priority (*integer 0..63*; Default:)	Matches the priority of an ingress packet. Priority may be derived from VLAN, WMM, DSCP or MPLS EXP bit. read more
ip-protocol (*dccp | ddp | egp | encap | etherip | ggp | gre | hmp | icmp | icmpv6 | idpr-cmtp | igmp | ipencap | ipip | ipsec-ah | ipsec-esp | ipv6 | ipv6-frag | ipv6-nonxt | ipv6-opts | ipv6-route | iso-tp4 | l2tp | ospf | pim | pup | rdp | rspf | rsvp | sctp | st | tcp | udp | udp-lite | vmtp | vrrp | xns-idp | xtp*; Default:)	IP protocol (only if MAC protocol is set to IPv4) • dccp - Datagram Congestion Control Protocol • ddp - Datagram Delivery Protocol • egp - Exterior Gateway Protocol • encap - Encapsulation Header • etherip - Ethernet-within-IP Encapsulation • ggp - Gateway-to-Gateway Protocol • gre - Generic Routing Encapsulation • hmp - Host Monitoring Protocol • icmp - IPv4 Internet Control Message Protocol • icmpv6 - IPv6 Internet Control Message Protocol • idpr-cmtp - Inter-Domain Policy Routing Control Message Transport Protocol • igmp - Internet Group Management Protocol • ipencap - IP in IP (encapsulation) • ipip - IP-within-IP Encapsulation Protocol • ipsec-ah - IPsec Authentication Header • ipsec-esp - IPsec Encapsulating Security Payload • ipv6 - Internet Protocol version 6 • ipv6-frag - Fragment Header for IPv6 • ipv6-nonxt - No Next Header for IPv6 • ipv6-opts - Destination Options for IPv6 • ipv6-route - Routing Header for IPv6 • iso-tp4 - ISO Transport Protocol Class 4 • l2tp - Layer Two Tunneling Protocol • ospf - Open Shortest Path First • pim - Protocol Independent Multicast • pup - PARC Universal Packet • rdp - Reliable Data Protocol • rspf - Radio Shortest Path First • rsvp - Reservation Protocol • sctp - Stream Control Transmission Protocol • st - Internet Stream Protocol • tcp - Transmission Control Protocol • udp - User Datagram Protocol • udp-lite - Lightweight User Datagram Protocol • vmtp - Versatile Message Transaction Protocol • vrrp - Virtual Router Redundancy Protocol • xns-idp - Xerox Network Systems Internet Datagram Protocol • xtp - Xpress Transport Protocol
jump-target (*name*; Default:)	If `action=jump` specified, then specifies the user-defined firewall chain to process the packet.

limit (*integer/time,integer*; Default:)	Restricts packet match rate to a given limit. • count - maximum average packet rate, measured in packets per second (pps), unless followed by Time option • time - specifies the time interval over which the packet rate is measured • burst - number of packets to match in a burst
log (*yes* | *no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, dst-mac, eth-protocol, ip-protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*text*; Default:)	Defines the prefix to be printed before the logging information.
mac-protocol (*802.2* | *arp* | *homeplug-av* | *ip* | *ipv6* | *ipx* | *length* | *lldp* | *loop-protect* | *mpls-multicast* | *mpls-unicast* | *packing-compr* | *packing-simple* | *pppoe* | *pppoe-discovery* | *rarp* | *service-vlan* | *vlan* | *integer 0..65535* | *hex 0x0000-0xffff*; Default:)	Ethernet payload type (MAC-level protocol). To match protocol type for VLAN encapsulated frames (0x8100 or 0x88a8), a vlan-encap prop erty should be used. • 802.2 - 802.2 Frames (0x0004) • arp - Address Resolution Protocol (0x0806) • homeplug-av - HomePlug AV MME (0x88E1) • ip - Internet Protocol version 4 (0x0800) • ipv6 - Internet Protocol Version 6 (0x86DD) • ipx - Internetwork Packet Exchange (0x8137) • length - Packets with length field (0x0000-0x05DC) • lldp - Link Layer Discovery Protocol (0x88CC) • loop-protect - Loop Protect Protocol (0x9003) • mpls-multicast - MPLS multicast (0x8848) • mpls-unicast - MPLS unicast (0x8847) • packing-compr - Encapsulated packets with compressed IP packing (0x9001) • packing-simple - Encapsulated packets with simple IP packing (0 x9000) • pppoe - PPPoE Session Stage (0x8864) • pppoe-discovery - PPPoE Discovery Stage (0x8863) • rarp - Reverse Address Resolution Protocol (0x8035) • service-vlan - Provider Bridging (IEEE 802.1ad) & Shortest Path Bridging IEEE 802.1aq (0x88A8) • vlan - VLAN-tagged frame (IEEE 802.1Q) and Shortest Path Bridging IEEE 802.1aq with NNI compatibility (0x8100)
new-packet-mark (*string*; Default:)	Sets a new packet-mark value.
new-priority (*integer* | *from-ingress*; Default:)	Sets a new priority for a packet. This can be the VLAN, WMM or MPLS EXP priority Read more. This property can also be used to set an internal priori
out-bridge (*name*; Default:)	Outgoing bridge interface.
out-bridge-list (*name*; Default:)	Set of bridge interfaces defined in interface list. Works the same as `out-bridge`.
out-interface (*name*; Default:)	Interface that the packet is leaving the bridge through.
out-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as `out-interface`.
packet-mark (*name*; Default:)	Match packets with a certain packet mark.

packet-type (*broadcast* / *host* / *multicast* / *other-host*; Default:)	MAC frame type: • broadcast - broadcast MAC packet • host - packet is destined to the bridge itself • multicast - multicast MAC packet • other-host - packet is destined to some other unicast address, not to the bridge itself
src-address (*IP address*; Default:)	Source IP address (only if MAC protocol is set to IPv4).
src-address6 (*IPv6 address*; Default:)	Source IPv6 address (only if MAC protocol is set to IPv6).
src-mac-address (*MAC address*; Default:)	Source MAC address.
src-port (*integer 0..65535*; Default:)	Source port number or range (only for TCP or UDP protocols).
stp-flags (*topology-change* / *topology-change-ack*; Default:)	The BPDU (Bridge Protocol Data Unit) flags. Bridge exchange configuration messages named BPDU periodically for preventing loops • topology-change - topology change flag is set when a bridge detects port state change, to force all other bridges to drop their host tables and recalculate network topology • topology-change-ack - topology change acknowledgment flag is sent in replies to the notification packets
stp-forward-delay (*integer 0..65535*; Default:)	Forward delay timer.
stp-hello-time (*integer 0..65535*; Default:)	STP hello packets time.
stp-max-age (*integer 0..65535*; Default:)	Maximal STP message age.
stp-msg-age (*integer 0..65535*; Default:)	STP message age.
stp-port (*integer 0..65535*; Default:)	STP port identifier.
stp-root-address (*MAC address*; Default:)	Root bridge MAC address.
stp-root-cost (*integer 0..65535*; Default:)	Root bridge cost.
stp-root-priority (*integer 0..65535*; Default:)	Root bridge priority.
stp-sender-address (*MAC address*; Default:)	STP message sender MAC address.
stp-sender-priority (*integer 0..65535*; Default:)	STP sender priority.
stp-type (*config* / *tcn*; Default:)	The BPDU type: • config - configuration BPDU • tcn - topology change notification
tls-host (*string*; Default:)	Allows matching https traffic based on TLS SNI hostname. Accepts GLOB syntax for wildcard matching. Note that matcher will not be able to match hostname if the TLS handshake frame is fragmented into multiple TCP segments (packets).
vlan-encap (*802.2* / *arp* / *ip* / *ipv6* / *ipx* / *length* / *mpls-multicast* / *mpls-unicast* / *pppoe* / *pppoe-discovery* / *rarp* / *vlan* / *integer 0..65535* / *hex 0x0000-0xffff*; Default:)	Matches the MAC protocol type encapsulated in the VLAN frame.
vlan-id (*integer 0..4095*; Default:)	Matches the VLAN identifier field.
vlan-priority (*integer 0..7*; Default:)	Matches the VLAN priority (priority code point)

Footnotes:

- STP matchers are only valid if the destination MAC address is `01:80:C2:00:00:00/FF:FF:FF:FF:FF:FF` (Bridge Group address), also STP should be enabled.

- ARP matchers are only valid if mac-protocol is `arp` or `rarp`
- VLAN matchers are only valid for `0x8100` or `0x88a8` ethernet protocols
- IP or IPv6 related matchers are only valid if mac-protocol is either set to `ip` or `ipv6`
- 802.3 matchers are only consulted if the actual frame is compliant with IEEE 802.2 and IEEE 802.3 standards. These matchers are ignored for other packets.

Bridge Packet Filter

This section describes specific bridge filter options.

Sub-menu: `/interface bridge filter`

Property	Description
action (*accept* | *drop* | *jump* | *log* | *mark-packet* | *passthrough* | *return* | *set-priority*; Default: **accept**)	Action to take if the packet is matched by the rule: - accept - accept the packet. No action, i.e., the packet is passed through without undertaking any action, and no more rules are processed in the relevant list/chain - drop - silently drop the packet (without sending the ICMP reject message) - jump - jump to the chain specified by the value of the jump-target argument - log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, dst-mac, eth-proto, protocol, src-ip:port->dst-ip:port and length of the packet. After packet is matched it is passed to the next rule in the list, similar as passthrough - mark - mark the packet to use the mark later - passthrough - ignore this rule and go on to the next one. Acts the same way as a disabled rule, except for the ability to count packets - return - return to the previous chain, from where the jump took place - set-priority - set priority specified by the new-priority parameter on the packets sent out through a link that is capable of transporting priority (VLAN or WMM-enabled wireless interface). Read more

Bridge NAT

This section describes specific bridge NAT options.

Sub-menu: `/interface bridge nat`

Property	Description
action (*accept* | *drop* | *jump* | *mark-packet* | *redirect* | *set-priority* | *arp-reply* | *dst-nat* | *log* | *passthrough* | *return* | *src-nat*; Default: **accept**)	Action to take if the packet is matched by the rule: - accept - accept the packet. No action, i.e., the packet is passed through without undertaking any action, and no more rules are processed in the relevant list/chain - arp-reply - send a reply to an ARP request (any other packets will be ignored by this rule) with the specified MAC address (only valid in dstnat chain) - drop - silently drop the packet (without sending the ICMP reject message) - dst-nat - change destination MAC address of a packet (only valid in dstnat chain) - jump - jump to the chain specified by the value of the jump-target argument - log - log the packet - mark - mark the packet to use the mark later - passthrough - ignore this rule and go on to the next one. Acts the same way as a disabled rule, except for the ability to count packets - redirect - redirect the packet to the bridge itself (only valid in dstnat chain) - return - return to the previous chain, from where the jump took place - set-priority - set priority specified by the new-priority parameter on the packets sent out through a link that is capable of transporting priority (VLAN or WMM-enabled wireless interface). Read more - src-nat - change source MAC address of a packet (only valid in srcnat chain)
to-arp-reply-mac-address (*MAC address*; Default:)	Source MAC address to put in Ethernet frame and ARP payload, when `action=arp-reply` is selected

to-dst-mac-address (*MAC address*; Default:)	Destination MAC address to put in Ethernet frames, when `action=dst-nat` is selected
to-src-mac-address (*MAC address*; Default:)	Source MAC address to put in Ethernet frames, when `action=src-nat` is selected

See also

- CRS1xx/2xx series switches
- CRS3xx, CRS5xx series switches, and CCR2116, CCR2216 routers
- Switch chip features
- MTU on RouterBOARD
- Layer2 misconfiguration
- Bridge VLAN Table
- Wireless VLAN Trunk
- VLANs on Wireless

CRS3xx, CRS5xx, CCR2116, CCR2216 switch chip features

Summary

The CCR3xx, CRS5xx series switches and CCR2116, CCR2216 routers have highly integrated switches with high-performance CPU and feature-rich packet processors. These devices can be designed into various Ethernet applications including unmanaged switch, Layer 2 managed switch, carrier switch, inter-VLAN router, and wired unified packet processor.

⚠ This article applies to CRS3xx, CRS5xx series switches, CCR2116, CCR2216 routers, and not to CRS1xx/CRS2xx series switches.

Features

Features	Description

Forwarding	• Configurable ports for switching or routing • Full non-blocking wire-speed switching • Large Unicast FDB for Layer 2 unicast forwarding • Forwarding Databases works based on IVL • Jumbo frame support • IGMP Snooping support • DHCP Snooping with Option 82

Routing	• Layer 3 Hardware Offloading:
	  ○ IPv4, IPv6 Unicast Routing
	  ○ Supported on Ethernet, Bridge, Bonding, and VLAN interfaces
	  ○ ECMP
	  ○ Blackholes
	  ○ Offloaded Fasttrack connections (applies only to certain switch models)
	  ○ Offloaded NAT for Fasttrack connections (applies only to certain switch models)
	  ○ Multiple MTU profiles

Spanning Tree Protocol	• STP • RSTP • MSTP

Mirroring	• Various types of mirroring:
	  ○ Port based mirroring
	  ○ VLAN based mirroring
	  ○ MAC based mirroring

VLAN	• Fully compatible with IEEE802.1Q and IEEE802.1ad VLAN
	• 4k active VLANs
	• Flexible VLAN assignment:
	  ○ Port based VLAN
	  ○ Protocol based VLAN
	  ○ MAC based VLAN
	• VLAN filtering
	• Ingress VLAN translation

Bonding	• Supports 802.3ad (LACP) and balance-xor modes • Up to 8 member ports per bonding interface • Hardware automatic failover and load balancing • MLAG

Traffic Shaping	• Ingress traffic limiting
	  ○ Port based
	  ○ MAC based
	  ○ IP based
	  ○ VLAN based
	  ○ Protocol based
	  ○ DSCP based
	• Port based egress traffic limiting
	• Traffic Storm Control

Port isolation	• Applicable for Private VLAN implementation

Access Control List	• Ingress ACL tables • Classification based on ports, L2, L3, L4 protocol header fields • ACL actions include filtering, forwarding and modifying of the protocol header fields

Models

This table clarifies the main differences between Cloud Router Switch models and CCR routers.

Model	Switch Chip	CPU	Cores	10G SFP+	10G Ethernet	25G SFP28	40G QSFP+	100G QSFP28	ACL rules	Unicast FDB entries	Jumbo Frame (Bytes)
netPower 15FR (CRS318-1Fi-15Fr-2S)	Marvell-98DX224S	800MHz	1	-	-	-	-	-	128	16,000	10218
netPower 16P (CRS318-16P-2S+)	Marvell-98DX226S	800MHz	1	2	-	-	-	-	128	16,000	10218
CRS310-1G-5S-4S+ (netFiber 9 /IN)	Marvell-98DX226S	800MHz	1	4	-	-	-	-	128	16,000	10218
CRS326-24G-2S+ (RM/IN)	Marvell-98DX3236	800MHz	1	2	-	-	-	-	128	16,000	10218
CRS328-24P-4S+	Marvell-98DX3236	800MHz	1	4	-	-	-	-	128	16,000	10218
CRS328-4C-20S-4S+	Marvell-98DX3236	800MHz	1	4	-	-	-	-	128	16,000	10218
CRS305-1G-4S+	Marvell-98DX3236	800MHz	1	4	-	-	-	-	128	16,000	10218
CRS309-1G-8S+	Marvell-98DX8208	800MHz	2	8	-	-	-	-	1024	32,000	10218
CRS317-1G-16S+	Marvell-98DX8216	800MHz	2	16	-	-	-	-	1024	128,000	10218
CRS312-4C+8XG	Marvell-98DX8212	650MHz	1	4 (combo ports)	8 + 4 (combo ports)	-	-	-	512	32,000	10218
CRS326-24S+2Q+	Marvell-98DX8332	650MHz	1	24	-	-	2	-	256	32,000	10218
CRS354-48G-4S+2Q+	Marvell-98DX3257	650MHz	1	4	-	-	2	-	170	32,000	10218
CRS354-48P-4S+2Q+	Marvell-98DX3257	650MHz	1	4	-	-	2	-	170	32,000	10218
CRS504-4XQ (IN/OUT)	Marvell-98DX4310	650MHz	1	-	-	-	-	4	1024	128,000	10218
CRS510-8XS-2XQ-IN	Marvell-98DX4310	650MHz	1	-	-	8	-	2	1024	128,000	10218
CRS518-16XS-2XQ	Marvell-98DX8525	650MHz	1	-	-	16	-	2	1024	128,000	10218
CCR2116-12G-4S+	Marvell-98DX3255	2000MHz	16	4	-	-	-	-	512	32,000	9570
CCR2216-1G-12XS-2XQ	Marvell-98DX8525	2000MHz	16	-	-	12	-	2	1024	128,000	9570

For L3 hardware offloading feature support and hardware limits, please refer to Feature Support and Device Support user manuals.

Abbreviations

- FDB - Forwarding Database
- MDB - Multicast Database
- SVL - Shared VLAN Learning
- IVL - Independent VLAN Learning
- PVID - Port VLAN ID
- ACL - Access Control List

- CVID - Customer VLAN ID
- SVID - Service VLAN ID

Port switching

In order to set up a port switching, check the Bridge Hardware Offloading page.

Currently, it is possible to create only one bridge with hardware offloading. Use the `hw=yes/no` parameter to select which bridge will use hardware offloading.

Bridge STP/RSTP/MSTP, IGMP Snooping and VLAN filtering settings don't affect hardware offloading, since RouterOS v6.42 Bonding interfaces are also hardware offloaded.

VLAN

Since RouterOS version 6.41, a bridge provides VLAN aware Layer2 forwarding and VLAN tag modifications within the bridge. This set of features makes bridge operation more like a traditional Ethernet switch and allows to overcome Spanning Tree compatibility issues compared to the configuration when tunnel-like VLAN interfaces are bridged. Bridge VLAN Filtering configuration is highly recommended to comply with STP (802.1D), RSTP (802.1w) standards and it is mandatory to enable MSTP (802.1s) support in RouterOS.

VLAN Filtering

VLAN filtering is described on the Bridge VLAN Filtering section.

VLAN setup examples

Below are describes some of the most common ways how to utilize VLAN forwarding.

Port-Based VLAN

The configuration is described on the Bridge VLAN FIltering section.

MAC Based VLAN

- The Switch Rule table is used for MAC Based VLAN functionality, see this table on how many rules each device supports.
- MAC-based VLANs will only work properly between switch ports and not between switch ports and CPU. When a packet is being forwarded to the CPU, the `pvid` property for the bridge port will be always used instead of `new-vlan-id` from ACL rules.
- MAC-based VLANs will not work for DHCP packets when DHCP snooping is enabled.

Enable switching on ports by creating a bridge with enabled hw-offloading:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
```

Add VLANs in the Bridge VLAN table and specify ports:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether2 untagged=ether7 vlan-ids=200,300,400
```

Add Switch rules which assign VLAN id based on MAC address:

```
/interface ethernet switch rule
add switch=switch1 ports=ether7 src-mac-address=A4:12:6D:77:94:43/FF:FF:FF:FF:FF:FF new-vlan-id=200
add switch=switch1 ports=ether7 src-mac-address=84:37:62:DF:04:20/FF:FF:FF:FF:FF:FF new-vlan-id=300
add switch=switch1 ports=ether7 src-mac-address=E7:16:34:A1:CD:18/FF:FF:FF:FF:FF:FF new-vlan-id=400
```

Protocol Based VLAN

- The Switch Rule table is used for Protocol Based VLAN functionality, see this table on how many rules each device supports.
- Protocol-based VLANs will only work properly between switch ports and not between switch ports and CPU. When a packet is being forwarded to the CPU, the `pvid` property for the bridge port will be always used instead of `new-vlan-id` from ACL rules.
- Protocol-based VLANs will not work for DHCP packets when DHCP snooping is enabled.

Enable switching on ports by creating a bridge with enabled hw-offloading:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Add VLANs in the Bridge VLAN table and specify ports:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether2 untagged=ether6 vlan-ids=200
add bridge=bridge1 tagged=ether2 untagged=ether7 vlan-ids=300
add bridge=bridge1 tagged=ether2 untagged=ether8 vlan-ids=400
```

Add Switch rules which assign VLAN id based on MAC protocol:

```
/interface ethernet switch rule
add mac-protocol=ip new-vlan-id=200 ports=ether6 switch=switch1
add mac-protocol=ipx new-vlan-id=300 ports=ether7 switch=switch1
add mac-protocol=0x80F3 new-vlan-id=400 ports=ether8 switch=switch1
```

VLAN Tunneling (Q-in-Q)

Since RouterOS v6.43 it is possible to use a provider bridge (IEEE 802.1ad) and Tag Stacking VLAN filtering, and hardware offloading at the same time. The configuration is described in the Bridge VLAN Tunneling (Q-in-Q) section.

Devices with switch chip Marvell-98DX3257 (e.g. CRS354 series) do not support VLAN filtering on 1Gbps Ethernet interfaces for other VLAN types (`0x88a8` and `0x9100`).

Ingress VLAN translation

It is possible to translate a certain VLAN ID to a different VLAN ID using ACL rules on an ingress port. In this example we create two ACL rules, allowing bidirectional communication. This can be done by doing the following.

Create a new bridge and add ports to it with hardware offloading:

```
/interface bridge
add name=bridge1 vlan-filtering=no
/interface bridge port
add interface=ether1 bridge=bridge1 hw=yes
add interface=ether2 bridge=bridge1 hw=yes
```

Add ACL rules to translate a VLAN ID in each direction:

```
/interface ethernet switch rule
add new-dst-ports=ether2 new-vlan-id=20 ports=ether1 switch=switch1 vlan-id=10
add new-dst-ports=ether1 new-vlan-id=10 ports=ether2 switch=switch1 vlan-id=20
```

Add both VLAN IDs to the bridge VLAN table:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=10
add bridge=bridge1 tagged=ether2 vlan-ids=20
```

Enable bridge VLAN filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```


Bidirectional communication is limited only between two switch ports. Translating VLAN ID between more ports can cause traffic flooding or incorrect forwarding between the same VLAN ports.

By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

(R/M)STP

CRS3xx, CRS5xx series switches, and CCR2116, CCR2216 routers are capable of running STP, RSTP, and MSTP on a hardware level. For more detailed information you should check out the Spanning Tree Protocol manual page.

Bonding

CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers support hardware offloading with bonding interfaces. Only `802.3ad` and `balance-xor` bonding modes are hardware offloaded, other bonding modes will use the CPU's resources. You can find more information about the bonding interfaces in the Bonding Interface section. If `802.3ad` mode is used, then LACP (Link Aggregation Control Protocol) is supported.

To create a hardware offloaded bonding interface, you must create a bonding interface with a supported bonding mode:

```
/interface bonding
add mode=802.3ad name=bond1 slaves=ether1,ether2
```

This interface can be added to a bridge alongside other interfaces:

```
/interface bridge
add name=bridge
/interface bridge port
add bridge=bridge interface=bond1 hw=yes
add bridge=bridge interface=ether3 hw=yes
add bridge=bridge interface=ether4 hw=yes
```

⚠ Do not add interfaces to a bridge that are already in a bond, RouterOS will not allow you to add an interface to bridge that is already a slave port for bonding.

Make sure that the bonding interface is hardware offloaded by checking the "H" flag:

```
/interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                                 BRIDGE                                    HW
 0   H bond1                                     bridge                                    yes
 1   H ether3                                    bridge                                    yes
 2   H ether4                                    bridge                                    yes
```

⚠ With HW-offloaded bonding interfaces, the built-in switch chip will always use Layer2+Layer3+Layer4 for a transmit hash policy, changing the transmit hash policy manually will have no effect.

Multi-chassis Link Aggregation Group

MLAG (Multi-chassis Link Aggregation Group) implementation in RouterOS allows configuring LACP bonds on two separate devices, while the client device believes to be connected on the same machine. This provides a physical redundancy in case of switch failure. All CRS3xx, CRS5xx series and CCR2116, CCR2216 devices can be configured with MLAG. Read here for more information.

L3 Hardware Offloading

Layer3 hardware offloading (otherwise known as IP switching or HW routing) will allow to offload some of the router features onto the switch chip. This allows reaching wire speeds when routing packets, which simply would not be possible with the CPU.

Offloaded feature set depends on the used chipset. Read here for more info.

Port isolation

Since RouterOS v6.43 is it possible to create a Private VLAN setup, an example can be found in the Switch chip port isolation manual page. Hardware offloaded bonding interfaces are not included in the switch port-isolation menu, but it is still possible to configure port-isolation individually on each secondary interface of the bonding.

⚠ Port isolation can be used with vlan-filtering bridge and it is possible to isolate ports that are members of the same VLAN. The isolation works per-port, it is not possible to isolate ports per-VLAN.

IGMP/MLD Snooping

CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers are capable of using IGMP/MLD Snooping on a hardware level. To see more detailed information, you should check out the IGMP/MLD snooping manual page.

DHCP Snooping and DHCP Option 82

CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers are capable of using DHCP Snooping with Option 82 on a hardware level. The switch will create a dynamic ACL rule to capture the DHCP packets and redirect them to the main CPU for further processing. To see more detailed information, please visit the DHCP Snooping and DHCP Option 82 manual page.

DHCP snooping will not work when hardware offloading bonding interfaces are created.

Controller Bridge and Port Extender

Controller Bridge (CB) and Port Extender (PE) is an IEEE 802.1BR standard implementation in RouterOS. It allows virtually extending the CB ports with a PE device and managing these extended interfaces from a single controlling device. Such configuration provides a simplified network topology, flexibility, increased port density, and ease of manageability. See more details on Controller Bridge and Port Extender manual.

Mirroring

Mirroring is a function that allows a network switch to duplicate all the data passing through it and send a copy to another specified port, known as the `mirror-target`. This feature is useful for setting up a tap device, which allows for analyzing network traffic using a separate device. You can set up mirroring in a simple way by designating source ports (see `mirror-egress` and `mirror-ingress` in `/interface/ethernet/switch/port`), or you can configure more advanced mirroring based on different criteria (see `mirror` in `/interface/ethernet/switch/rule`).

It is important to note that the `mirror-target` port must be on the same switch. You can check the device block diagram or navigate to the `/interface/ethernet` menu to identify which interfaces are connected where. When setting up the configration, it is not mandatory to add the `mirror-target` interface to the same hardware offloaded bridge where the source ports are set up. The `mirror-target` port can be a standalone interface (not configured as a bridge port), or it can be within a bridge setup. When using the `mirror-target` with a bridge, note that data and mirrored traffic may both travel on the same LAN. In such cases, consider employing RSPAN (Remote Switch Port Analyzer), where mirrored traffic is encapsulated into a separate VLAN before being transmitted over the network.

Additionally, you can set the `mirror-target` port to a special value "cpu", which means that the copied packets will be sent to the switch chip's CPU port.

Configuration examples

Port Based Mirroring

Starting from RouterOS version 7.15, it is possible to configure multiple source ports and selectively choose whether to mirror incoming traffic, outgoing traffic, or both. In this example, both incoming and outgoing traffic from the **ether2** interface will be copied and sent to the **ether3** interface for monitoring or analysis.

```
# Since RouterOS v7.15
/interface ethernet switch port
set ether2 mirror-egress=yes mirror-ingress=yes
/interface ethernet switch
set switch1 mirror-target=ether3

# Older RouterOS:
/interface ethernet switch
set switch1 mirror-source=ether2 mirror-target=ether3
```

VLAN Based Mirroring

Using ACL rules, it is possible to mirror packets from multiple interfaces using the `ports` setting. Additionally, you can specify more detailed criteria such as VLAN ID, MAC/IP address or TCP/UDP port. Only **ingress** packets are mirrored to `mirror-target` interface. This example will mirror incoming VLAN 11 traffic from the **ether2** interface, and send copies to the **ether3** interface. To use an ACL rule with a `vlan-id` matcher, you need to have bridge vlan-filtering enabled.

```
/interface bridge
set bridge1 vlan-filtering=yes
/interface ethernet switch
set switch1 mirror-target=ether3
/interface ethernet switch rule
add mirror=yes ports=ether1 switch=switch1 vlan-id=11
```

MAC Based Mirroring

This example will mirror incoming traffic with 64:D1:54:D9:27:E6 MAC destination or source address from the **ether1** interface, and send copies to the **ether3** interface.

```
/interface ethernet switch
set switch1 mirror-target=ether3
/interface ethernet switch rule
add mirror=yes ports=ether1 switch=switch1 dst-mac-address=64:D1:54:D9:27:E6/FF:FF:FF:FF:FF:FF
add mirror=yes ports=ether1 switch=switch1 src-mac-address=64:D1:54:D9:27:E6/FF:FF:FF:FF:FF:FF
```

IP Based Mirroring

This example will mirror incoming traffic with 192.168.88.0/24 IP destination or source address from the **ether1** interface, and send copies to the **ether3** interface.

```
/interface ethernet switch
set switch1 mirror-target=ether3 mirror-source=none
/interface ethernet switch rule
add mirror=yes ports=ether1 switch=switch1 src-address=192.168.88.0/24
add mirror=yes ports=ether1 switch=switch1 dst-address=192.168.88.0/24
```

There are other options as well, check the ACL section to find out all possible parameters that can be used to match packets.

Remote Switch Port Analyzer

This example will mirror incomming and outgoing traffic from the **ether2** interface, copies will be encapsulated in 802.1Q VLAN using the 999 as VLAN ID, and packets will be sent to the **ether3** interface. If the original traffic is already VLAN tagged, RSPAN will add another layer of VLAN tagging as an outer tag. This results in the mirrored traffic being tagged twice. If the `mirror-target` port is included in vlan-filtering bridge, it is not required to make the interface as tagged VLAN member under the `/interface/bridge/vlan` menu for the RSPAN.

```
/interface ethernet switch port
set ether2 mirror-egress=yes mirror-ingress=yes
/interface ethernet switch
set switch1 mirror-target=ether3 rspan=yes rspan-egress-vlan-id=999 rspan-ingress-vlan-id=999
```

Property Reference

Sub-menu: `/interface/ethernet/switch`

Property	Description
mirror-target (*cpu* / *name* / *none*; Default: **none**)	Selects a single mirroring target port. Packets from `mirror-egress` and `mirror-ingress` (`/interface/ethernet/switch/port`) and mirror (`/interface/ethernet/switch/rule`) will be sent to the selected port.
rspan (*no* / *yes*; Default: **no**)	Enables Remote Switch Port Analyzer (RSPAN) feature on `mirror-target`. Traffic marked for ingress or egress mirroring is carried over a specified remote analyzer VLAN - `rspan-egress-vlan-id` and `rspan-ingress-vlan-id`.
rspan-egress-vlan-id (*integer: 1..4095*; Default: **1**)	Selects the VLAN ID for marked egress traffic. Only applies when `rspan` is enabled.

rspan-ingress-vlan-id (*integer: 1..4095*; Default: **1**)	Selects the VLAN ID for marked ingress traffic. Only applies when `rspan` is enabled.

Sub-menu: `/interface/ethernet/switch/port`

Property	Description
mirror-egress (*no* / *yes*; Default: **no**)	Whether to send egress packet copy to the `mirror-target` port.
mirror-ingress (*no* / *yes*; Default: **no**)	Whether to send ingress packet copy to the `mirror-target` port.

Sub-menu: `/interface/ethernet/switch/rule`

Property	Description
mirror (*no* / *yes*; Default: **no**)	Whether to send a packet copy to `mirror-target` port.

Traffic Shaping

It is possible to limit ingress traffic that matches certain parameters with ACL rules and it is possible to limit ingress/egress traffic per port basis. The policer is used for ingress traffic, the shaper is used for egress traffic. The ingress policer controls the received traffic with packet drops. Everything that exceeds the defined limit will get dropped. This can affect the TCP congestion control mechanism on end hosts and achieved bandwidth can be actually less than defined. The egress shaper tries to queue packets that exceed the limit instead of dropping them. Eventually, it will also drop packets when the output queue gets full, however, it should allow utilizing the defined throughput better.

Port-based traffic police and shaper:

```
/interface ethernet switch port
set ether1 ingress-rate=10M egress-rate=5M
```

MAC-based traffic policer:

```
/interface ethernet switch rule
add ports=ether1 switch=switch1 src-mac-address=64:D1:54:D9:27:E6/FF:FF:FF:FF:FF:FF rate=10M
```

VLAN-based traffic policer:

```
/interface bridge
set bridge1 vlan-filtering=yes
/interface ethernet switch rule
add ports=ether1 switch=switch1 vlan-id=11 rate=10M
```

By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

Protocol-based traffic policer:

```
/interface ethernet switch rule
add ports=ether1 switch=switch1 mac-protocol=ipx rate=10M
```

There are other options as well, check the ACL section to find out all possible parameters that can be used to match packets.

The Switch Rule table is used for QoS functionality, see this table on how many rules each device supports.

Traffic Storm Control

Since RouterOS v6.42 it is possible to enable traffic storm control. A traffic storm can emerge when certain frames are continuously flooded on the network. For example, if a network loop has been created and no loop avoidance mechanisms are used (e.g. Spanning Tree Protocol), broadcast or multicast frames can quickly overwhelm the network, causing degraded network performance or even complete network breakdown. With CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers it is possible to limit broadcast, unknown multicast and unknown unicast traffic. Unknown unicast traffic is considered when a switch does not contain a host entry for the destined MAC address. Unknown multicast traffic is considered when a switch does not contain a multicast group entry in the `/interface bridge mdb` menu. Storm control settings should be applied to ingress ports, the egress traffic will be limited.

The storm control parameter is specified in percentage (%) of the link speed. If your link speed is 1Gbps, then specifying `storm-rate` as `10` will allow only 100Mbps of broadcast, unknown multicast and/or unknown unicast traffic to be forwarded.

Sub-menu: `/interface ethernet switch port`

Property	Description
limit-broadcasts (*yes* / *no*; Default: **yes**)	Limit broadcast traffic on a switch port.
limit-unknown-multicasts (*yes* / *no*; Default: **no**)	Limit unknown multicast traffic on a switch port.
limit-unknown-unicasts (*yes* / *no*; Default: **no**)	Limit unknown unicast traffic on a switch port.
storm-rate (*integer 0..100*; Default: **100**)	Amount of broadcast, unknown multicast and/or unknown unicast traffic is limited to in percentage of the link speed.

Devices with Marvell-98DX3236 switch chip cannot distinguish unknown multicast traffic from all multicast traffic. For example, CRS326-24G-2S+ will limit all multicast traffic when `limit-unknown-multicasts` and `storm-rate` is used. For other devices, for example, CRS317-1G-16S+ the `limit-unknown-multicasts` parameter will limit only unknown multicast traffic (addresses that are not present in `/interface bridge mdb`).

For example, to limit 1% (10Mbps) of broadcast and unknown unicast traffic on ether1 (1Gbps), use the following commands:

```
/interface ethernet switch port
set ether1 storm-rate=1 limit-broadcasts=yes limit-unknown-unicasts=yes
```

MPLS hardware offloading

Since RouterOS v6.41 it is possible to offload certain MPLS functions to the switch chip, the switch must be a (P)rovider router in a PE-P-PE setup in order to achieve hardware offloading. A setup example can be found in the Basic MPLS setup example manual page. The hardware offloading will only take place when LDP interfaces are configured as physical switch interfaces (e.g. Ethernet, SFP, SFP+).

Currently only `CRS317-1G-16S+` and `CRS309-1G-8S+` using RouterOS v6.41 and newer are capable of hardware offloading certain MPLS functions. `CRS317-1G-16S+` and `CRS309-1G-8S+` built-in switch chip is not capable of popping MPLS labels from packets, in a PE-P-PE setup you either have to use explicit null or disable TTL propagation in MPLS network to achieve hardware offloading.

The MPLS hardware offloading has been removed since RouterOS v7.

Switch Rules (ACL)

Access Control List contains ingress policy and egress policy engines. See this table on how many rules each device supports. It is an advanced tool for wire-speed packet filtering, forwarding and modifying based on Layer2, Layer3 and Layer4 protocol header field conditions.

⚠ ACL rules are checked for each received packet until a match has been found. If there are multiple rules that can match, then only the first rule will be triggered. A rule without any action parameters is a rule to accept the packet.

⚠ It is not required to set `mac-protocol` to certain IP version when using L3 or L4 matchers, however, it is recommended to set the `mac-protocol=ip` or `mac-protocol=ipv6` when filtering any IP packets.

⚠ When switch ACL rules are modified (e.g. added, removed, disabled, enabled, or moved), the existing switch rules will be inactive for a short time. This can cause some packet leakage during the ACL rule modifications.

Sub-menu: `/interface ethernet switch rule`

Property	Description
copy-to-cpu (*no* / *yes*; Default: **no**)	Clones the matching packet and sends it to the CPU.
disabled (*yes* / *no*; Default: **no**)	Enables or disables ACL entry.
dscp (*0..63*)	Matching the DSCP field of the packet (only applies to IPv4 packets).
dst-address (*IP address/Mask*)	Matching destination IPv4 address and mask, also matches the destination IP in ARP packets.
dst-address6 (*IPv6 address/Mask*)	Matching destination IPv6 address and mask.
dst-mac-address (*MAC address/Mask*)	Matching destination MAC address and mask.
dst-port (*0..65535*)	Matching destination protocol port number (applies to IPv4 and IPv6 packets if `mac-protocol` is not specified).
flow-label (*0..1048575*)	Matching IPv6 flow label.
mac-protocol (*802.2 / arp / homeplug-av / ip / ipv6 / ipx / lldp / loop-protect / mpls-multicast / mpls-unicast / packing-compr / packing-simple / pppoe / pppoe-discovery / rarp / service-vlan / vlan / or 0..65535 / or 0x0000-0xffff*)	Matching particular MAC protocol specified by protocol name or number
mirror (*no* / *yes*)	Clones the matching packet and sends it to the mirror-target port.
new-dst-ports (*ports*)	Changes the destination port as specified. An empty setting will drop the packet. A specified port will redirect the packet to it. When the parameter is not used, the packet will be accepted. Multiple "new-dst-ports" are not supported.
new-vlan-id (*0..4095*)	Changes the VLAN ID to the specified value. Requires `vlan-filtering=yes`.
new-vlan-priority (*0..7*)	Changes the VLAN priority (priority code point). Requires `vlan-filtering=yes`.
ports (*ports*)	Matching ports on which will the rule apply on received traffic.
protocol (*dccp / ddp / egp / encap / etherip / ggp / gre / hmp / icmp / icmpv6 / idpr-cmtp / igmp / ipencap / ipip / ipsec-ah / ipsec-esp / ipv6 / ipv6-frag / ipv6-nonxt / ipv6-opts / ipv6-route / iso-tp4 / l2tp / ospf / pim / pup / rdp / rspf / rsvp / sctp / st / tcp / udp / udp-lite / vmtp / vrrp / xns-idp / xtp / or 0..255*)	Matching particular IP protocol specified by protocol name or number. Only applies to IPv4 packets if `mac-protocol` is not specified. To match certain IPv6 protocols, use the `mac-protocol=ipv6` setting.
rate (*0..4294967295*)	Sets ingress traffic limitation (bits per second) for matched traffic.
redirect-to-cpu (*no* / *yes*)	Changes the destination port of a matching packet to the CPU.

src-address (*IP address/Mask*)	Matching source IPv4 address and mask, also matches the source IP in ARP packets.
src-address6 (*IPv6 address/Mask*)	Matching source IPv6 address and mask.
src-mac-address (*MAC address/Mask*)	Matching source MAC address and mask.
src-port (*0..65535*)	Matching source protocol port number (applies to IPv4 and IPv6 packets if `mac-protocol` is not specified).
switch (*switch group*)	Matching switch group on which will the rule apply.
traffic-class (*0..255*)	Matching IPv6 traffic class.
vlan-id (*0..4095*)	Matching VLAN ID. Requires `vlan-filtering=yes`.
vlan-header (*not-present* | *present*)	Matching VLAN header, whether the VLAN header is present or not. Requires `vlan-filtering=yes`.
vlan-priority (*0..7*)	Matching VLAN priority (priority code point).

Action parameters:

- copy-to-cpu
- redirect-to-cpu
- mirror
- new-dst-ports (can be used to drop packets)
- new-vlan-id
- new-vlan-priority
- rate

Layer2 condition parameters:

- dst-mac-address
- mac-protocol
- src-mac-address
- vlan-id
- vlan-header
- vlan-priority

Layer3 condition parameters:

- dscp
- protocol
- IPv4 conditions:
  - dst-address
  - src-address
- IPv6 conditions:
  - dst-address6
  - flow-label
  - src-address6
  - traffic-class

Layer4 condition parameters:

- dst-port
- src-port

For VLAN related matchers or VLAN related action parameters to work, you need to enable `vlan-filtering` on the bridge interface and make sure that hardware offloading is enabled on those ports, otherwise, these parameters will not have any effect.

When bridge interface `ether-type` is set to `0x8100`, then VLAN related ACL rules are relevant to frames tagged using regular/customer VLAN (TPID 0x8100), this includes `vlan-id` and `new-vlan-id`. When bridge interface `ether-type` is set to `0x88a8`, then ACL rules are relevant to frames tagged with 802.1ad service tag (TPID 0x88a8).

Port Security

It is possible to limit allowed MAC addresses on a single switch port. For example, to allow 64:D1:54:81:EF:8E MAC address on a switch port, start by switching multiple ports together, in this example 64:D1:54:81:EF:8E is going to be located behind **ether1**.

Create an ACL rule to allow the given MAC address and drop all other traffic on **ether1** (for ingress traffic):

```
/interface ethernet switch rule
add ports=ether1 src-mac-address=64:D1:54:81:EF:8E/FF:FF:FF:FF:FF:FF switch=switch1
add new-dst-ports="" ports=ether1 switch=switch1
```

Switch all required ports together, disable MAC learning and disable unknown unicast flooding on **ether1**:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes learn=no unknown-unicast-flood=no
add bridge=bridge1 interface=ether2 hw=yes
```

Add a static hosts entry for 64:D1:54:81:EF:8E (for egress traffic):

```
/interface bridge host
add bridge=bridge1 interface=ether1 mac-address=64:D1:54:81:EF:8E
```

Broadcast traffic will still be sent out from **ether1**. To limit broadcast traffic flood on a bridge port, you can use the `broadcast-flood` parameter to toggle it. Do note that some protocols depend on broadcast traffic, such as streaming protocols and DHCP.

Dual Boot

The “dual boot” feature allows you to choose which operating system you prefer to use on CRS3xx series switches, RouterOS or SwOS. Device operating system could be changed using:

- Command-line (`/system routerboard settings set boot-os=swos`)
- Winbox
- Webfig
- Serial Console

More details about SwOS are described here: SwOS manual

Configuring SwOS using RouterOS

Since RouterOS 6.43 it is possible to load, save and reset SwOS configuration, as well as upgrade SwOS and set an IP address for the CRS3xx series switches by using RouterOS.

- Save configuration with `/system swos save-config`

The configuration will be saved on the same device with `swos.config` as a filename, make sure you download the file from your device since the configuration file will be removed after a reboot.

- Load configuration with `/system swos load-config`
- Change password with `/system swos password`
- Reset configuration with `/system swos reset-config`
- Upgrade SwOS from RouterOS using `/system swos upgrade`

⚠ The upgrade command will automatically install the latest available SwOS primary backup version, make sure that your device has access to the Internet in order for the upgrade process to work properly. When the device is booted into SwOS, the version number will include the letter "p", indicating a primary backup version. You can then install the latest available SwOS secondary main version from the SwOS "Upgrade" menu.

Property	Description
address-acquisition-mode (*dhcp-only* | *dhcp-with-fallback* | *static*; Default: **dhcp-with-fallback**)	Changes address acquisition method: dhcp-only - uses only a DHCP client to acquire address dhcp-with-fallback - for the first 10 seconds will try to acquire address using a DHCP client. If the request is unsuccessful, then address falls back to static as defined by static-ip-address property static - address is set as defined by static-ip-address property
allow-from (*IP/Mask*; Default: **0.0.0.0/0**)	IP address or a network from which the switch is accessible. By default, the switch is accessible by any IP address.
allow-from-ports (*name*; Default:)	List of switch ports from which the device is accessible. By default, all ports are allowed to access the switch
allow-from-vlan (*integer: 0..4094*; Default: **0**)	VLAN ID from which the device is accessible. By default, all VLANs are allowed
identity (*name*; Default: **MikrotiK**)	Name of the switch (used for Mikrotik Neighbor Discovery protocol)
static-ip-address (*IP*; Default: **192.168.88.1**)	IP address of the switch in case address-acquisition-mode is either set to dhcp-with-fallback or static. By setting a static IP address, the address acquisition process does not change, which is DHCP with fallback by default. This means that the configured static IP address will become active only when there is going to be no DHCP servers in the same broadcast domain

See also

CRS Router

CRS3xx VLANs with Bonds

Basic VLAN switching

Bridge Hardware Offloading

Route Hardware Offloading

Spanning Tree Protocol

MTU on RouterBOARD

Layer2 misconfiguration

Bridge VLAN Table

Bridge IGMP/MLD snooping

Multi-chassis Link Aggregation Group

CRS1xx/2xx series switches

Summary

The Cloud Router Switch series are highly integrated switches with high-performance MIPS CPU and feature-rich packet processors. The CRS switches can be designed into various Ethernet applications including unmanaged switch, Layer 2 managed switch, carrier switch, and wireless/wired unified packet processing. See Cloud Router Switch configuration examples

> This article applies to CRS1xx and CRS2xx series switches and not to CRS3xx series switches. For CRS3xx series devices, read the CRS3xx, CRS5xx series switches and CCR2116, CCR2216 routers manual.

Features	Description
Forwarding	• Configurable ports for switching or routing • Full non-blocking wire-speed switching • Up to 16k MAC entries in Unicast FDB for Layer 2 unicast forwarding • Up to 1k MAC entries in Multicast FDB for multicast forwarding • Up to 256 MAC entries in Reserved FDB for control and management purposes • All Forwarding Databases support IVL and SVL • Configurable Port-based MAC learning limit • Jumbo frame support (CRS1xx: 4064 Bytes; CRS2xx: 9204 Bytes) • IGMP Snooping support

Mirroring	• Various types of mirroring:
	  ○ Port-based mirroring
	  ○ VLAN-based mirroring
	  ○ MAC-based mirroring
	• 2 independent mirroring analyzer ports

VLAN	• Fully compatible with IEEE802.1Q and IEEE802.1ad VLAN
	• 4k active VLANs
	• Flexible VLAN assignment:
	  ○ Port-based VLAN
	  ○ Protocol-based VLAN
	  ○ MAC-based VLAN
	• From any to any VLAN translation and swapping
	• 1:1 VLAN switching - VLAN to port mapping
	• VLAN filtering

Port Isolation and Leakage	• Applicable for Private VLAN implementation • 3 port profile types: Promiscuous, Isolated, and Community • Up to 28 Community profiles • Leakage profiles allow bypassing egress VLAN filtering
Trunking	• Supports static link aggregation groups • Up to 8 Port Trunk groups • Up to 8 member ports per Port Trunk group • Hardware automatic failover and load balancing

Quality of Service (QoS)	• Flexible QoS classification and assignment:
	  ○ Port-based
	  ○ MAC-based
	  ○ VLAN-based
	  ○ Protocol-based
	  ○ PCP/DEI based
	  ○ DSCP based
	  ○ ACL based
	• QoS remarking and remapping for QoS domain translation between a service provider and client networks
	• Overriding of each QoS assignment according to the configured priority

Shaping and Scheduling	• 8 queues on each physical port • Shaping per port, per queue, per queue group
Access Control List	• Ingress and Egress ACL tables • Up to 128 ACL rules (limited by RouterOS) • Classification based on ports, L2, L3, L4 protocol header fields • ACL actions include filtering, forwarding, and modifying the protocol header fields

Cloud Router Switch models

This table clarifies the main differences between Cloud Router Switch models.

Model	Switch Chip	CPU	Wireless	SFP+ port	Access Control List	Jumbo Frame (Bytes)
CRS105-5S-FB	QCA-8511	400MHz	-	-	+	9204

CRS106-1C-5S	QCA-8511	400MHz	-	-	+	9204
CRS112-8G-4S	QCA-8511	400MHz	-	-	+	9204
CRS210-8G-2S+	QCA-8519	400MHz	-	+	+	9204
CRS212-1G-10S-1S+	QCA-8519	400MHz	-	+	+	9204
CRS226-24G-2S+	QCA-8519	400MHz	-	+	+	9204
CRS125-24G-1S	QCA-8513L	600MHz	-	-	-	4064
CRS125-24G-1S-2HnD	QCA-8513L	600MHz	+	-	-	4064
CRS109-8G-1S-2HnD	QCA-8513L	600MHz	+	-	-	4064

Abbreviations and Explanations

CVID - Customer VLAN id: inner VLAN tag id of the IEEE 802.1ad frame

SVID - Service VLAN id: outer VLAN tag id of the IEEE 802.1ad frame

IVL - Independent VLAN learning - learning/lookup is based on both MAC addresses and VLAN IDs.

SVL - Shared VLAN learning - learning/lookup is based on MAC addresses - not on VLAN IDs.

TPID - Tag Protocol Identifier

PCP - Priority Code Point: a 3-bit field which refers to the IEEE 802.1p priority

DEI - Drop Eligible Indicator

DSCP - Differentiated services Code Point

Drop precedence - internal CRS switch QoS attribute used for packet enqueuing or dropping.

Port Switching

To set up port switching on CRS1xx/2xx series switches, check the Bridge Hardware Offloading page.

Dynamic reserved VLAN entries (VLAN4091; VLAN4090; VLAN4089; etc.) are created in the CRS switch when switched port groups are added when a hardware offloaded bridge is created. These VLANs are necessary for internal operation and have lower precedence than user-configured VLANs.

Multiple switch groups

The CRS1xx/2xx series switches allow you to use multiple bridges with hardware offloading, this allows you to easily isolate multiple switch groups. This can be done by simply creating multiple bridges and enabling hardware offloading.

Multiple hardware offloaded bridge configuration is designed as a fast and simple port isolation solution, but it limits a part of the VLAN functionality supported by the CRS switch chip. For advanced configurations use one bridge within the CRS switch chip for all ports, configure VLANs, and isolate port groups with port isolation profile configuration.

CRS1xx/2xx series switches can run multiple hardware offloaded bridges with (R)STP enabled, but it is not recommended since the device is not designed to run multiple (R)STP instances on a hardware level. To isolate multiple switch groups and have (R)STP enabled you should isolate port groups with port isolation profile configuration.

Global Settings

The CRS switch chip is configurable from the `/interface ethernet switch` console menu.

Sub-menu: `/interface ethernet switch`

Property	Description
name (*string value*; Default: **switch1**)	Name of the switch.
bridge-type (*customer-vid-used-as-lookup-vid | service-vid-used-as-lookup-vid*; Default: **customer-vid-used-as-lookup-vid**)	The bridge type defines which VLAN tag is used as Lookup-VID. Lookup-VID serves as the VLAN key for all VLAN-based lookups.
mac-level-isolation (*yes | no*; Default: **yes**)	Globally enables or disables MAC level isolation. Once enabled, the switch will check the source and destination MAC address entries and their `isolation-profile` from the unicast forwarding table. By default, the switch will learn MAC addresses and place them into a `promiscuous` isolation profile. Other isolation profiles can be used when creating static unicast entries. If the source or destination MAC address is located on a `promiscuous` isolation profile, the packet is forwarded. If both source and destination MAC addresses are located on the same `community1` or `community2` isolation profile, the packet is forwarded. The packet is dropped when the source and destination MAC address isolation profile is `isolated`, or when the source and destination MAC address isolation profiles are from different communities (e.g. source MAC address is `community1` and destination MAC address is `community2`). When MAC level isolation is globally disabled, the isolation is bypassed.
use-svid-in-one2one-vlan-lookup (*yes | no*; Default: **no**)	Whether to use service VLAN ID for 1:1 VLAN switching lookup.
use-cvid-in-one2one-vlan-lookup (*yes | no*; Default: **yes**)	Whether to use customer VLAN ID for 1:1 VLAN switching lookup.
multicast-lookup-mode (*dst-ip-and-vid-for-ipv4 | dst-mac-and-vid-always*; Default:**dst-ip-and-vid-for-ipv4**)	Lookup mode for IPv4 multicast bridging. • dst-mac-and-vid-always - For all packet types lookup key is the destination MAC and VLAN ID. • dst-ip-and-vid-for-ipv4 - For IPv4 packets lookup key is the destination IP and VLAN ID. For other packet types, the lookup key is the destination MAC and VLAN ID.
unicast-fdb-timeout (*time interval*; Default: **5m**)	Timeout for Unicast FDB entries.
override-existing-when-ufdb-full (*yes | no*; Default: **no**)	Enable or disable to override existing entry which has the lowest aging value when UFDB is full.

Property	Description
drop-if-no-vlan-assignment-on-ports (*ports*; Default: **none**)	Ports which drop frames if no MAC-based, Protocol-based VLAN assignment or Ingress VLAN Translation is applied.
drop-if-invalid-or-src-port--not-member-of-vlan-on-ports (*ports*; Default: **none**)	Ports that drop invalid and other port VLAN ID frames.
unknown-vlan-lookup-mode (*ivl* | *svl*; Default: **svl**)	Lookup and learning mode for packets with invalid VLAN.
forward-unknown-vlan (*yes* | *no*; Default: **yes**)	Whether to allow forwarding VLANs that are not members of the VLAN table.

Property	Description
bypass-vlan-ingress-filter-for (*protocols*; Default: **none**)	Protocols that are excluded from Ingress VLAN filtering. These protocols are not dropped if they have invalid VLAN. (arp, dhcpv4, dhcpv6, eapol, igmp, mld, nd, pppoe-discovery, ripv1)
bypass-ingress-port-policing-for (*protocols*; Default: **none**)	Protocols that are excluded from Ingress Port Policing. (arp, dhcpv4, dhcpv6, eapol, igmp, mld, nd, pppoe-discovery, ripv1)
bypass-l2-security-check-filter-for (*protocols* ; Default: **none**)	Protocols that are excluded from Policy rule security check. (arp, dhcpv4, dhcpv6, eapol, igmp, mld, nd, pppoe-discovery, ripv1)

Property	Description
ingress-mirror0 (*port* | *trunk,format*; Default: **none,modified**)	The first ingress mirroring analyzer port or trunk and mirroring format: • analyzer-configured - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the analyzer port. • modified - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the egress port. • original - Traffic is mirrored without any change to the original incoming packet format. But the service VLAN tag is stripped in the edge port.
ingress-mirror1 (*port* | *trunk,format*; Default: **none,modified**)	The second ingress mirroring analyzer port or trunk and mirroring format: • analyzer-configured - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the analyzer port. • modified - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the egress port. • original - Traffic is mirrored without any change to the original incoming packet format. But the service VLAN tag is stripped in the edge port.
ingress-mirror-ratio (*1/32768..1/1*; Default: **1/1**)	The proportion of ingress mirrored packets compared to all packets.
egress-mirror0 (*port* | *trunk,format*; Default: **none,modified**)	The first egress mirroring analyzer port or trunk and mirroring format: • analyzer-configured - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the analyzer port. • modified - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the egress port. • original - Traffic is mirrored without any change to the original incoming packet format. But the service VLAN tag is stripped in the edge port.

egress-mirror1 (*port* | *trunk,format*; Default: **none,modified**)	The second egress mirroring analyzer port or trunk and mirroring format: • analyzer-configured - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the analyzer port. • modified - The packet is the same as the packet to the destination. VLAN format is modified based on the VLAN configurations of the egress port. • original - Traffic is mirrored without any change to the original incoming packet format. But the service VLAN tag is stripped in the edge port.
egress-mirror-ratio (*1/32768..1/1*; Default: **1/1**)	Proportion of egress mirrored packets compared to all packets.
mirror-egress-if-ingress-mirrored (*yes* | *no*; Default: **no**)	When a packet is applied to both ingress and egress mirroring, only ingress mirroring is performed on the packet, if this setting is disabled. If this setting is enabled both mirroring types are applied.
mirror-tx-on-mirror-port (*yes* | *no*; Default: **no**)	
mirrored-packet-qos-priority (*0..7*; Default: **0**)	Remarked priority in mirrored packets.
mirrored-packet-drop-precedence (*drop* | *green* | *red* | *yellow*; Default: **green**)	Remarked drop precedence in mirrored packets. This QoS attribute is used for mirrored packet enqueuing or dropping.
fdb-uses (*mirror0* | *mirror1*; Default: **mirror0**)	Analyzer port used for FDB-based mirroring.
vlan-uses (*mirror0* | *mirror1*; Default: **mirror0**)	Analyzer port used for VLAN-based mirroring.

Port Settings

Sub-menu: `/interface ethernet switch port`

Property	Description
vlan-type (*edge-port* | *network-port*; Default: **network-port**)	Port VLAN type specifies whether VLAN ID is used in UFDB learning. The network port learns VLAN ID in UFDB, edge port does not - VLAN 0. It can be observed only in IVL learning mode.
isolation-leakage-profile-override (*yes* | *no*; Default: **!isolation-leakage-profile-override**) **isolation-leakage-profile** (*0..31*;)	Custom port profile for port isolation/leakage configurations. • Port-level isolation profile 0. Uplink port - allows the port to communicate with all ports in the device. • Port-level isolation profile 1. Isolated port - allows the port to communicate only with uplink ports. • Port-level isolation profile 2 - 31. Community port - allows communication among the same community ports and uplink ports.
learn-override (*yes* | *no*; Default: **!learn-override**) **learn-limit** (*1..1023*; Default: **!learn-limit**)	Enable or disable MAC address learning and set the MAC limit on the port. MAC learning limit is disabled by default when !learn-override and !learn-limit are set. Property learn-override is replaced with learn under `/interface bridge port` menu since RouterOS v6.42.
drop-when-ufdb-entry-src-drop (*yes* | *no*; Default: **yes**)	Enable or disable to drop packets when UFDB entry has action src-drop.
allow-unicast-loopback (*yes* | *no*; Default: **no**)	Unicast loopback on port. When enabled, it permits sending back when the source port and destination port are the same for known unicast packets.
allow-multicast-loopback (*yes* | *no*; Default: **no**)	Multicast loopback on port. When enabled, it permits sending back when the source port and destination port are the same for registered multicast or broadcast packets.
action-on-static-station-move (*copy-to-cpu* | *drop* | *forward* | *redirect-to-cpu*; Default: **forward**)	Action for packets when UFDB already contains a static entry with such MAC but with a different port.
drop-dynamic-mac-move (*yes* | *no*; Default: **no**)	Prevents MAC relearning until UFDB timeout if MAC is already learned on another port.

Property	Description
allow-fdb-based-vlan-translate (*yes* | *no*; Default: **no**)	Enable or disable MAC-based VLAN translation on the port.
allow-mac-based-service-vlan-assignment-for (*all-frames* | *none* | *tagged-frame-only* | *untagged-and-priority-tagged-frame-only*; Default: **none**)	Frame type for which applies MAC-based service VLAN translation.
allow-mac-based-customer-vlan-assignment-for (*all-frames* | *none* | *tagged-frame-only* | *untagged-and-priority-tagged-frame-only*; Default: **none**)	Frame type for which applies MAC-based customer VLAN translation.
default-customer-pcp (*0..7*; Default: **0**)	Default customer PCP of the port.
default-service-pcp (*0..7*; Default: **0**)	Default service PCP of the port.
pcp-propagation-for-initial-pcp (*yes* | *no*; Default: **no**)	Enables or disables PCP propagation for initial PCP assignment on ingress. • If the port vlan-type is Edge port, the service PCP is copied from the customer PCP. • If the port vlan-type is a Network port, the customer PCP is copied from the service PCP.
filter-untagged-frame (*yes* | *no*; Default: **no**)	Whether to filter untagged frames on the port.
filter-priority-tagged-frame (*yes* | *no*; Default: **no**)	Whether to filter tagged frames with priority on the port.
filter-tagged-frame (*yes* | *no*; Default: **no**)	Whether to filter tagged frames on the port.

Property	Description
egress-vlan-tag-table-lookup-key (*according-to-bridge-type* | *egress-vid*; Default: **egress-vid**)	Egress VLAN table (VLAN Tagging) lookup: • egress-vid - Lookup VLAN ID is CVID when Edge port is configured, SVID when Network port is configured. • according-to-bridge-type - Lookup VLAN ID is CVID when customer VLAN bridge is configured, SVID when service VLAN bridge is configured. The Customer tag is unmodified for Edge port in service VLAN bridge.
egress-vlan-mode (*tagged* | *unmodified* | *untagged*; Default: **unmodified**)	Egress VLAN tagging action on the port.
egress-pcp-propagation (*yes* | *no*; Default: **no**)	Enables or disables egress PCP propagation. • If the port vlan-type is Edge port, the service PCP is copied from the customer PCP. • If the port vlan-type is Network port, the customer PCP is copied from the service PCP.

Property	Description
ingress-mirror-to (*mirror0* | *mirror1* | *none*; Default: **none**)	Analyzer port for port-based ingress mirroring.
ingress-mirroring-according-to-vlan (*yes* | *no*; Default: **no**)	
egress-mirror-to (*mirror0* | *mirror1* | *none*; Default: **none**)	Analyzer port for port-based egress mirroring.

Property	Description
qos-scheme-precedence (*da-based | dscp-based | ingress-acl-based | pcp-based | protocol-based | sa-based | vlan-based*; Default: **pcp-based, sa-based, da-based, dscp-based, protocol-based, vlan-based**)	Specifies applied QoS assignment schemes on the ingress of the port. • da-based • dscp-based • ingress-acl-based • pcp-based • protocol-based • sa-based • vlan-based
pcp-or-dscp-based-qos-change-dei (*yes | no*; Default: **no**)	Enable or disable PCP or DSCP based DEI change on port.
pcp-or-dscp-based-qos-change-pcp (*yes | no*; Default: **no**)	Enable or disable PCP or DSCP based PCP change on port.
pcp-or-dscp-based-qos-change-dscp (*yes | no*; Default: **no**)	Enable or disable PCP or DSCP based DSCP change on port.
dscp-based-qos-dscp-to-dscp-mapping (*yes | no*; Default: **yes**)	Enable or disable DSCP to internal DSCP mapping on port.
pcp-based-qos-drop-precedence-mapping (*PCP/DEI-range:drop-precedence*; Default: **0-15:green**)	The new value of drop precedence for the PCP/DEI to drop precedence (drop | green | red | yellow) mapping. Multiple mappings are allowed separated by a comma e.g. "0-7:yellow,8-15:red".
pcp-based-qos-dscp-mapping (*PCP/DEI-range:DEI*; Default: **0-15:0**)	The new value of DSCP for the PCP/DEI to DSCP (0..63) mapping. Multiple mappings are allowed separated by a comma e.g. "0-7:25,8-15:50".
pcp-based-qos-dei-mapping (*PCP/DEI-range:DEI*; Default: **0-15:0**)	The new value of DEI for the PCP/DEI to DEI (0..1) mapping. Multiple mappings are allowed separated by a comma e.g. "0-7:0,8-15:1".
pcp-based-qos-pcp-mapping (*PCP/DEI-range:DEI*; Default: **0-15:0**)	The new value of PCP for the PCP/DEI to PCP (0..7) mapping. Multiple mappings are allowed separated by a comma e.g. "0-7:3,8-15:4".
pcp-based-qos-priority-mapping (*PCP/DEI-range:DEI*; Default: **0-15:0**)	The new value of internal priority for the PCP/DEI to priority (0..15) mapping. Multiple mappings are allowed separated by a comma e.g. "0-7:5,8-15:15".

Property	Description
priority-to-queue (*priority-range:queue*; Default: **0-15:0,1:1,2:2,3:3**)	Internal priority (0..15) mapping to queue (0..7) per port.
per-queue-scheduling (*Scheduling-type:Weight*; Default: **wrr-group0:1,wrr-group0:2,wrr-group0:4,wrr-group0:8,wrr-group0:16,wrr-group0:32,** **wrr-group0:64,wrr-group0:128**)	Set port to use either strict or weighted round robin policy for traffic shaping for each queue group, each queue is separated by a comma.

Property	Description

ingress-customer-tpid-override (*yes* / *no*; Default:!**ingress-customer-tpid-override**) **ingress-customer-tpid** (*0..10000*; Default: **0x8100**)	Ingress customer TPID override allows accepting specific frames with a custom customer tag TPID. The default value is for the tag of 802.1Q frames.
egress-customer-tpid-override (*yes* / *no*; Default: **!egress-customer-tpid-override**) **egress-customer-tpid** (*0..10000*; Default: **0x8100**)	Egress customer TPID override allows custom identification for egress frames with a customer tag. The default value is for the tag of 802.1Q frames.
ingress-service-tpid-override (*yes* / *no*; Default: **!ingress-service-tpid-override**) **ingress-service-tpid** (*0..10000*; Default: **0x88A8**)	Ingress service TPID override allows accepting specific frames with a custom service tag TPID. The default value is for the service tag of 802.1AD frames.
egress-service-tpid-override (*yes* / *no*; Default: **!egress-service-tpid-override**) **egress-service-tpid** (*0..10000*; Default: **0x88A8**)	Egress service TPID override allows custom identification for egress frames with a service tag. The default value is for the service tag of 802.1AD frames.

Property	Description
custom-drop-counter-includes (*counters*; Default: **none**)	Custom include to count dropped packets for switch port custom-drop-packet counter. • **device-loopback** • **fdb-hash-violation** • **exceeded-port-learn-limitation** • **dynamic-station-move** • **static-station-move** • **ufdb-source-drop** • **host-source-drop** • **unknown-host** • **ingress-vlan-filtered**
queue-custom-drop-counter0-includes (*counters*; Default: **none**)	Custom include to count dropped packets for switch port tx-queue-custom0-drop-packet and bytes for tx-queue-custom0-drop-byte counters. • **red** • **yellow** • **green** • **queue0** • **...** • **queue7**

queue-custom-drop-counter1-includes (*counters*; Default: **none**)	Custom include to count dropped packets for switch port tx-queue-custom1-drop-packet and bytes for tx-queue-custom1-drop-byte counters. • **red** • **yellow** • **green** • **queue0** • **...** • **queue7**
policy-drop-counter-includes (*counters*; Default: **none**)	Custom include to count dropped packets for switch port policy-drop-packet counter. • **ingress-policing** • **ingress-acl** • **egress-policing** • **egress-acl**

Forwarding Databases

Unicast FDB

The unicast forwarding database supports up to 16318 MAC entries.

Sub-menu: `/interface ethernet switch unicast-fdb`

Property	Description
action (*action*; Default: **forward**)	Action for UFDB entry: • dst-drop - Packets are dropped when their destination MAC matches the entry. • dst-redirect-to-cpu - Packets are redirected to the CPU when their destination MAC matches the entry. • forward - Packets are forwarded. • src-and-dst-drop - Packets are dropped when their source MAC or destination MAC matches the entry. • src-and-dst-redirect-to-cpu - Packets are redirected to CPU when their source MAC or destination MAC matches the entry. • src-drop - Packets are dropped when their source MAC matches the entry. • src-redirect-to-cpu - Packets are redirected to the CPU when their source MAC matches the entry.
disabled (*yes* | *no*; Default: **no**)	Enables or disables Unicast FDB entry.
isolation-profile (*community1* | *community2* | *isolated* | *promiscuous*; Default: **promiscuous**)	MAC level isolation profile.
mac-address (*MAC address*)	The action command applies to the packet when the destination MAC or source MAC matches the entry.
mirror (*yes* | *no*; Default: **no**)	Enables or disables mirroring based on source MAC or destination MAC.
port (*port*)	Matching port for the Unicast FDB entry.
qos-group (*none*; Default: **none**)	Defined QoS group from QoS group menu.

svl (*yes* | *no*; Default: **no**)	Unicast FDB learning mode: • Shared VLAN Learning (svl) - learning/lookup is based on MAC addresses - not on VLAN IDs. • Independent VLAN Learning (ivl) - learning/lookup is based on both MAC addresses and VLAN IDs.
vlan-id (*0..4095*)	Unicast FDB lookup/learning VLAN id.

Multicast FDB

CRS125 switch-chip supports up to 1024 entries in MFDB for multicast forwarding. For each multicast packet, destination MAC or destination IP lookup is performed in MFDB. MFDB entries are not automatically learned and can only be configured.

Sub-menu: `/interface ethernet switch multicast-fdb`

Property	Description
address (*X.X.X.X* | *XX:XX:XX:XX:XX:XX*)	Matching IP address or MAC address for multicast packets.
bypass-vlan-filter (*yes* | *no*; Default: **no**)	Allow to bypass VLAN filtering for matching multicast packets.
disabled (*yes* | *no*; Default: **no**)	Enables or disables Multicast FDB entry.
ports (*ports*)	Member ports for multicast traffic.
qos-group (*none*; Default: **none**)	Defined QoS group from QoS group menu.
svl (*yes* | *no*; Default: **no**)	Multicast FDB learning mode: • Shared VLAN Learning (svl) - learning/lookup is based on MAC addresses - not on VLAN IDs. • Independent VLAN Learning (ivl) - learning/lookup is based on both MAC addresses and VLAN IDs.
vlan-id (*0..4095*; Default: **0**)	Multicast FDB lookup VLAN ID. If the VLAN learning mode is IVL, VLAN id is lookup id, otherwise VLAN id = 0.

Reserved FDB

Cloud Router Switch supports 256 RFDB entries. Each RFDB entry can store either Layer2 unicast or multicast MAC address with specific commands.

Sub-menu: `/interface ethernet switch reserved-fdb`

Property	Description
action (*copy-to-cpu* | *drop* | *forward* | *redirect-to-cpu*; Default: **forward**)	Action for RFDB entry: • copy-to-cpu - Packets are copied to the CPU when their destination MAC matches the entry. • drop - Packets are dropped when their destination MAC matches the entry. • forward - Packets are forwarded when their destination MAC matches the entry. • redirect-to-cpu - Packets are redirected to CPU when their destination MAC matches the entry.
bypass-ingress-port-policing (*yes* | *no*; Default: **no**)	Allow to bypass Ingress Port Policer for matching packets.
bypass-ingress-vlan-filter (*yes* | *no*; Default: **no**)	Allow to bypass VLAN filtering for matching packets.
disabled (*yes* | *no*; Default: **no**)	Enables or disables Reserved FDB entry.
mac-address (*MAC address*; Default: **00:00:00:00:00:00**)	Matching MAC address for Reserved FDB entry.
qos-group (*none*; Default: **none**)	Defined QoS group from QoS group menu.

VLAN

VLAN Table

The VLAN table supports 4096 VLAN entries for storing VLAN member information as well as other VLAN information such as QoS, isolation, forced VLAN, learning, and mirroring.

Sub-menu: `/interface ethernet switch vlan`

Property	Description
disabled (*yes* | *no*; Default: **no**)	Indicate whether the VLAN entry is disabled. Only enabled entry is applied to the lookup process and forwarding decision.
flood (*yes* | *no*; Default: **no**)	Enables or disables forced VLAN flooding per VLAN. If the feature is enabled, the result of the destination MAC lookup in the UFDB or MFDB is ignored, and the packet is forced to flood in the VLAN.
ingress-mirror (*yes* | *no*; Default: **no**)	Enable the ingress mirror per VLAN to support the VLAN-based mirror function.
learn (*yes* | *no*; Default: **yes**)	Enables or disables source MAC learning for VLAN.
ports (*ports*)	Member ports of the VLAN.
qos-group (*none*; Default: **none**)	Defined QoS group from QoS group menu.
svl (*yes* | *no*; Default: **no**)	FDB lookup mode for lookup in UFDB and MFDB. • Shared VLAN Learning (svl) - learning/lookup is based on MAC addresses - not on VLAN IDs. • Independent VLAN Learning (ivl) - learning/lookup is based on both MAC addresses and VLAN IDs.
vlan-id (*0..4095*)	VLAN ID of the VLAN member entry.

Egress VLAN Tag

Egress packets can be assigned different VLAN tag formats. The VLAN tags can be removed, added, or remained as is when the packet is sent to the egress port (destination port). Each port has dedicated control of the egress VLAN tag format. The tag formats include:

- Untagged
- Tagged
- Unmodified

The Egress VLAN Tag table includes 4096 entries for VLAN tagging selection.

Sub-menu: `/interface ethernet switch egress-vlan-tag`

Property	Description
disabled (*yes* | *no*; Default: **no**)	Enables or disables Egress VLAN Tag table entry.
tagged-ports (*ports*)	Ports that are tagged in egress.
vlan-id (*0..4095*)	VLAN ID which is tagged in egress.

Ingress/Egress VLAN Translation

The Ingress VLAN Translation table allows for up to 15 entries for each port. One or multiple fields can be selected from the packet header for lookup in the Ingress VLAN Translation table. The S-VLAN or C-VLAN or both configured in the first matched entry are assigned to the packet.

Sub-menu: `/interface ethernet switch ingress-vlan-translation`

Sub-menu: `/interface ethernet switch egress-vlan-translation`

Property	Description
customer-dei (*0..1*; Default: **none**)	Matching DEI of the customer tag.
customer-pcp (*0..7*; Default: **none**)	Matching PCP of the customer tag.
customer-vid (*0..4095*; Default: **none**)	Matching the VLAN ID of the customer tag.
customer-vlan-format (*any* | *priority-tagged-or-tagged* | *tagged* | *untagged-or-tagged*; Default:**any**)	Type of frames with customer tag for which VLAN translation rule is valid.
disabled (*yes* | *no*; Default: **no**)	Enables or disables VLAN translation entry.
new-customer-vid (*0..4095*; Default: **none**)	The new customer VLAN ID replaces the matching customer VLAN ID. If set to 4095 and ingress VLAN translation is used, then traffic is dropped.
new-service-vid (*0..4095*; Default: **none**)	The new service VLAN ID replaces the matching service VLAN ID.
pcp-propagation (*yes* | *no*; Default: **no**)	Enables or disables PCP propagation. • If the port type is Edge, the customer PCP is copied from the service PCP. • If the port type is Network, the service PCP is copied from the customer PCP.
ports (*ports*)	Matching switch ports for VLAN translation rule.
protocol (*protocols*; Default: **none**)	Matching Ethernet protocol. *(only for Ingress VLAN Translation)*
sa-learning (*yes* | *no*; Default: **no**)	Enables or disables source MAC learning after VLAN translation. *(only for Ingress VLAN Translation)*
service-dei (*0..1*; Default: **none**)	Matching DEI of the service tag.
service-pcp (*0..7*; Default: **none**)	Matching PCP of the service tag.
service-vid (*0..4095*; Default: **none**)	Matching VLAN ID of the service tag.
service-vlan-format (*any* | *priority-tagged-or-tagged* | *tagged* | *untagged-or-tagged*; Default:**any**)	Type of frames with service tag for which VLAN translation rule is valid.

Below is a table of traffic that triggers a rule that has a certain VLAN format set, note that traffic that is tagged with VLAN ID 0 is a special case that is also taken into account.

Property	Description
any	Accepts: • Untagged traffic • Tagged traffic • Tagged traffic with priority set • VLAN 0 traffic • VLAN 0 traffic with priority set
priority-tagged-or-tagged	Accepts: • Tagged traffic • Tagged traffic with priority set • VLAN 0 traffic • VLAN 0 traffic with priority set
tagged	Accepts: • Tagged traffic • Tagged traffic with priority set

untagged-or-tagged	Accepts: • Untagged traffic • Tagged traffic • Tagged traffic with priority set

If `VLAN-format` is set to `any`, then `customer-vid`/`service-vid` set to `0` will trigger the switch rule with VLAN 0 traffic. In this case, the switch rule will be looking for untagged traffic or traffic with a VLAN 0 tag, and only `untagged-or-tagged` will filter out VLAN 0 traffic in this case.

Protocol Based VLAN

Protocol Based VLAN table is used to assign VID and QoS attributes to related protocol packets per port.

Sub-menu: `/interface ethernet switch protocol-based-vlan`

Property	Description
disabled (*yes* / *no*; Default: **no**)	Enables or disables Protocol Based VLAN entry.
frame-type (*ethernet* / *llc* / *rfc-1042*; Default: **ethernet**)	Encapsulation type of the matching frames.
new-customer-vid (*0..4095*; Default: **0**)	The new customer VLAN ID replaces the original customer VLAN ID for the specified protocol. If set to 4095, then traffic is dropped.
new-service-vid (*0..4095*; Default: **0**)	The new service VLAN ID replaces the original service VLAN ID for the specified protocol.
ports (*ports*)	Matching switch ports for Protocol-based VLAN rule.
protocol (*protocol*; Default: **0**)	Matching protocol for Protocol-based VLAN rule.
qos-group (*none*; Default: **none**)	Defined QoS group from QoS group menu.
set-customer-vid-for (*all* / *none* / *tagged* / *untagged-or-priority-tagged*; Default: **all**)	Customer VLAN ID assignment command for different packet types.
set-qos-for (*all* / *none* / *tagged* / *untagged-or-priority-tagged*; Default: **none**)	Frame type for which QoS assignment command applies.
set-service-vid-for (*all* / *none* / *tagged* / *untagged-or-priority-tagged*; Default: **all**)	Service VLAN ID assignment command for different packet types.

MAC Based VLAN

MAC Based VLAN table is used to assign VLAN based on the source MAC.

Sub-menu: `/interface ethernet switch mac-based-vlan`

Property	Description
disabled (*yes* / *no*; Default: **no**)	Enables or disables MAC Based VLAN entry.
new-customer-vid (*0..4095*; Default: **0**)	The new customer VLAN ID replaces the original service VLAN ID for matched packets. If set to 4095, then traffic is dropped.
new-service-vid (*0..4095*; Default: **0**)	The new service VLAN ID replaces the original service VLAN ID for matched packets.
src-mac-address (*MAC address*)	Matching source MAC address for MAC based VLAN rule.

⚠ All CRS1xx/2xx series switches support up to 1024 MAC Based VLAN table entries.

1:1 VLAN Switching

1:1 VLAN switching can be used to replace the regular L2 bridging for matched packets. When a packet hits a 1:1 VLAN switching table entry, the destination port information in the entry is assigned to the packet. The matched destination information in the UFDB and MFDB entry no longer applies to the packet.

Sub-menu: `/interface ethernet switch one2one-vlan-switching`

Property	Description
customer-vid (*0..4095*; Default: **0**)	Matching customer VLAN id for 1:1 VLAN switching.
disabled (*yes* / *no*; Default: **no**)	Enables or disables 1:1 VLAN switching table entry.
dst-port (*port*)	Destination port for matched 1:1 VLAN switching packets.
service-vid (*0..4095*; Default: **0**)	Matching customer VLAN id for 1:1 VLAN switching.

Port Isolation/Leakage

The CRS switches support flexible multi-level isolation features, which can be used for user access control, traffic engineering and advanced security and network management. The isolation features provide an organized fabric structure allowing user to easily program and control the access by port, MAC address, VLAN, protocol, flow, and frame type. The following isolation and leakage features are supported:

- Port-level isolation
- MAC-level isolation
- VLAN-level isolation
- Protocol-level isolation
- Flow-level isolation
- Free combination of the above

Port-level isolation supports different control schemes on the source port and destination port. Each entry can be programmed with access control for either the source port or the destination port.

- When the entry is programmed with source port access control, the entry is

applied to the ingress packets.

- When the entry is programmed with destination port access control, the entry

is applied to the egress packets.

Port leakage allows bypassing egress VLAN filtering on the port. A leaky port is allowed to access other ports for various applications such as security, network control, and management. Note: When both isolation and leakage are applied to the same port, the port is isolated.

Sub-menu: `/interface ethernet switch port-isolation`

Sub-menu: `/interface ethernet switch port-leakage`

Property	Description
disabled (*yes* / *no*; Default: **no**)	Enables or disables port isolation/leakage entry.
flow-id (*0..63*; Default: **none**)	
forwarding-type (*bridged; routed*; Default: **bridged,routed**)	Matching traffic forwarding type on Cloud Router Switch.
mac-profile (*community1* / *community2* / *isolated* / *promiscuous*; Default: **none**)	Matching MAC isolation/leakage profile.

port-profile (*0..31*; Default: **none**)	Matching Port isolation/leakage profile.
ports (*ports*; Default: **none**)	Isolated/leaked ports.
protocol-type (*arp; nd; dhcpv4; dhcpv6; ripv1*; Default: **arp,nd, dhcpv4,dhcpv6,ripv1**)	Included protocols for isolation/leakage.
registration-status (*known; unknown*; Default: **known,unknown**)	Registration status for matching packets. Known are present in UFDB and MFDB, and unknown are not.
traffic-type (*unicast; multicast; broadcast*; Default: **unicast, multicast,broadcast**)	Matching traffic type.
type (*dst* | *src*; Default: **src**)	Lookup type of the isolation/leakage entry: • src - Entry applies to ingress packets of the ports. • dst - Entry applies to egress packets of the ports.
vlan-profile (*community1* | *community2* | *isolated* | *promiscuous*; Default: **none**)	Matching VLAN isolation/leakage profile.

Trunking

The Trunking in the Cloud Router Switches provides static link aggregation groups with hardware automatic failover and load balancing. IEEE802.3ad and IEEE802.1ax compatible Link Aggregation Control Protocol is not supported. Up to 8 Trunk groups are supported with up to 8 Trunk member ports per Trunk group. CRS Port Trunking calculates transmit-hash based on all following parameters: L2 src-dst MAC + L3 src-dst IP + L4 src-dst Port.

Sub-menu: `/interface ethernet switch trunk`

Property	Description
disabled (*yes* | *no*; Default: **no**)	Enables or disables port trunking entry.
member-ports (*ports*)	Member ports of the Trunk group.
name (*string value*; Default: **trunkX**)	Name of the Trunk group.

Quality of Service

Shaper

Traffic shaping restricts the rate and burst size of the flow which is transmitted out from the interface. The shaper is implemented by a token bucket. If the packet exceeds the maximum rate or the burst size, which means not enough token for the packet, the packet is stored to buffer until there is enough token to transmit it.

Sub-menu: `/interface ethernet switch shaper`

Property	Description
burst (*integer*; Default: **100k**)	Maximum data rate which can be transmitted while the burst is allowed.
disabled (*yes* | *no*; Default: **no**)	Enables or disables traffic shaper entry.
meter-unit (*bit* | *packet*; Default: **bit**)	Measuring units for traffic shaper rate.
port (*port*)	Physical port for traffic shaper.
rate (*integer*; Default: **1M**)	Maximum data rate limit.

target (*port* | *queueX* | *wrr-groupX*; Default: **port**)	Three levels of shapers are supported on each port (including CPU port): • Port level - Entry applies to the port of the switch-chip. • WRR group level - Entry applies to one of the 2 Weighted Round Robin queue groups (wrr-group0, wrr-group1) on the port. • Queue level - Entry applies to one of the 8 queues (queue0 - queue7) on the port.

Ingress Port Policer

Sub-menu: `/interface ethernet switch ingress-port-policer`

Property	Description
burst (*integer*; Default: **100k**)	Maximum data rate which can be transmitted while the burst is allowed.
disabled (*yes* | *no*; Default: **no**)	Enables or disables ingress port policer entry.
meter-len (*layer-1* | *layer-2* | *layer-3*; Default: **layer-1**)	Packet classification which sets the packet byte length for metering. • layer-1 - includes entire layer-2 frame + FCS + inter-packet gap + preamble. • layer-2 - includes layer-2 frame + FCS. • layer-3 - includes only layer-3 + ethernet padding without layer-2 header and FCS.
meter-unit (*bit* | *packet*; Default: **bit**)	Measuring units for traffic ingress port policer rate.
new-dei-for-yellow (*0..1* | *remap*; Default: **none**)	Remarked DEI for exceeded traffic if yellow-action is remark.
new-dscp-for-yellow (*0..63* | *remap*; Default: **none**)	Remarked DSCP for exceeded traffic if yellow-action is remark.
new-pcp-for-yellow (*0..7* | *remap*; Default: **none**)	Remarked PCP for exceeded traffic if yellow-action is remark.
packet-types (*packet-types*; Default: **all types from description**)	Matching packet types for which ingress port policer entry is valid.
port (*port*)	Physical port or trunk for ingress port policer entry.
rate (*integer*)	Maximum data rate limit.
yellow-action (*drop* | *forward* | *remark*; Default: **drop**)	Performed action for exceeded traffic.

QoS Group

The global QoS group table is used for VLAN-based, Protocol-based, and MAC-based QoS group assignment configuration.

Sub-menu: `/interface ethernet switch qos-group`

Property	Description
dei (*0..1*; Default: **none**)	The new value of DEI for the QoS group.
disabled (*yes* | *no*; Default: **no**)	Enables or disables protocol QoS group entry.
drop-precedence (*drop* | *green* | *red* | *yellow*; Default: **green**)	Drop precedence is an internal QoS attribute used for packet enqueuing or dropping.
dscp (*0..63*; Default: **none**)	The new value of DSCP for the QoS group.
name (*string value*; Default: **groupX**)	Name of the QoS group.
pcp (*0..7*; Default: **none**)	The new value of PCP for the QoS group.
priority (*0..15*; Default: **0**)	Internal priority is a local significance of priority for classifying traffic to different egress queues on a port. (1 is highest, 15 is lowest)

DSCP QoS Map

The global DSCP to QOS mapping table is used for mapping from the DSCP of the packet to new QoS attributes configured in the table.

Sub-menu: `/interface ethernet switch dscp-qos-map`

Property	Description
dei (*0..1*)	The new value of DEI for the DSCP to QOS mapping entry.
drop-precedence (*drop* | *green* | *red* | *yellow*)	The new value of Drop precedence for the DSCP to QOS mapping entry.
pcp (*0..7*)	The new value of PCP for the DSCP to QOS mapping entry.
priority (*0..15*)	The new value of internal priority for the DSCP to QOS mapping entry.

DSCP To DSCP Map

The global DSCP to DSCP mapping table is used for mapping from the packet's original DSCP to the new DSCP value configured in the table.

Sub-menu: `/interface ethernet switch dscp-to-dscp`

Property	Description
new-dscp (*0..63*)	The new value of DSCP for the DSCP to DSCP mapping entry.

Policer QoS Map

Sub-menu: `/interface ethernet switch policer-qos-map`

Property	Description
dei-for-red (*0..1*; Default: **0**)	Policer DEI remapping value for red packets.
dei-for-yellow (*0..1*; Default: **0**)	Policer DEI remapping value for yellow packets.
dscp-for-red (*0..63*; Default: **0**)	Policer DSCP remapping value for red packets.
dscp-for-yellow (*0..63*; Default: **0**)	Policer DSCP remapping value for yellow packets.
pcp-for-red (*0..7*; Default: **0**)	Policer PCP remapping value for red packets.
pcp-for-yellow (*0..7*; Default: **0**)	Policer PCP remapping value for yellow packets.

Access Control List

Access Control List contains of ingress policy and egress policy engines and allows configuration of up to 128 policy rules (limited by RouterOS). It is an advanced tool for wire-speed packet filtering, forwarding, shaping, and modifying based on Layer2, Layer3, and Layer4 protocol header field conditions.

See the Summary section for Access Control List supported Cloud Router Switch devices.

Due to hardware limitations, it is not possible to match broadcast/multicast traffic on specific ports. You should use port isolation, drop traffic on ingress ports, or use VLAN filtering to prevent certain broadcast/multicast traffic from being forwarded.

Sub-menu: `/interface ethernet switch acl`

ACL condition part for MAC-related fields of packets.

Property	Description
disabled (*yes* | *no*; Default: **no**)	Enables or disables ACL entry.
table (*egress* | *ingress*; Default: **ingress**)	Selects the policy table for incoming or outgoing packets.
invert-match (*yes* | *no*; Default: **no**)	Inverts the whole ACL rule matching.
src-ports (*ports,trunks*)	Matching physical source ports or trunks.
dst-ports (*ports,trunks*)	Matching physical destination ports or trunks. It is not possible to match broadcast/multicast traffic on the egress port due to a hardware limitation.
mac-src-address (*MAC address/Mask*)	Source MAC address and mask.
mac-dst-address (*MAC address/Mask*)	Destination MAC address and mask.
dst-addr-registered (*yes* | *no*)	Defines whether to match packets with registered state - packets whose destination MAC address is in UFDB/MFDB/RFDB. Valid only in the egress table.
mac-protocol (*802.2 | arp | homeplug-av | ip | ip-or-ipv6 | ipv6 | ipx | lldp | loop-protect | mpls-multicast | mpls-unicast | non-ip | packing-compr | packing-simple | pppoe | pppoe-discovery | rarp | service-vlan | vlan or integer: 0..65535 decimal format or 0x0000-0xffff hex format*)	Ethernet payload type (MAC-level protocol) • **802.2** - 802.2 Frames (0x0004) • **arp** - Address Resolution Protocol (0x0806) • **homeplug-av** - HomePlug AV MME (0x88E1) • **ip** - Internet Protocol version 4 (0x0800) • **ip-or-ipv6** - IPv4 or IPv6 (0x0800 or 0x86DD) • **ipv6** - Internet Protocol Version 6 (0x86DD) • **ipx** - Internetwork Packet Exchange (0x8137) • **lldp** - Link Layer Discovery Protocol (0x88CC) • **loop-protect** - Loop Protect Protocol (0x9003) • **mpls-multicast** - MPLS multicast (0x8848) • **mpls-unicast** - MPLS unicast (0x8847) • **non-ip** - Not Internet Protocol version 4 (not 0x0800) • **packing-compr** - Encapsulated packets with compressed IP packing (0x9001) • **packing-simple** - Encapsulated packets with simple IP packing (0x9000) • **pppoe** - PPPoE Session Stage (0x8864) • **pppoe-discovery** - PPPoE Discovery Stage (0x8863) • **rarp** - Reverse Address Resolution Protocol (0x8035) • **service-vlan** - Provider Bridging (IEEE 802.1 ad) & Shortest Path Bridging IEEE 802.1aq (0x88A8) • **vlan** - VLAN-tagged frame (IEEE 802.1Q) and Shortest Path Bridging IEEE 802.1aq with NNI compatibility (0x8100)
drop-precedence (*drop* | *green* | *red* | *yellow*)	Matching internal drop precedence. Valid only in the egress table.
custom-fields	

ACL condition part for VLAN-related fields of packets.

Property	Description
lookup-vid (*0..4095*)	VLAN id used in lookup. It can be changed before reaching the egress table.

service-vid (*0-4095*)	Matching service VLAN id.
service-pcp (*0..7*)	Matching service PCP.
service-dei (*0..1*)	Matching service DEI.
service-tag (*priority-tagged* | *tagged* | *tagged-or-priority-tagged* | *untagged*)	Format of the service tag.
customer-vid (*0-4095*)	Matching customer VLAN ID.
customer-pcp (*0..7*)	Matching customer PCP.
customer-dei (*0..1*)	Matching customer DEI.
customer-tag (*priority-tagged* | *tagged* | *tagged-or-priority-tagged* | *untagged*)	Format of the customer tag.
priority (*0..15*)	Matching internal priority. Valid only in the egress table.

ACL condition part for IPv4 and IPv6 related fields of packets.

Property	Description
ip-src (*IPv4/0..32*)	Matching source IPv4 address.
ip-dst (*IPv4/0..32*)	Matching destination IPv4 address.
ip-protocol (*tcp* | *udp* | *udp-lite* | *other*)	IP protocol type.
src-l3-port (*0-65535*)	Matching Layer3 source port.
dst-l3-port (*0-65535*)	Matching Layer3 destination port.
ttl (*0* | *1* | *max* | *other*)	Matching TTL field of the packet.
dscp (*0..63*)	Matching DSCP field of the packet.
ecn (*0..3*)	Matching ECN field of the packet.
fragmented (*yes* | *no*)	Whether to match fragmented packets.
first-fragment (*yes* | *no*)	YES matches not fragmented and the first fragments, NO matches other fragments.
ipv6-src (*IPv6/0..128*)	Matching source IPv6 address.
ipv6-dst (*IPv6/0..128*)	Matching destination IPv6 address.
mac-isolation-profile (*community1* | *community2* | *isolated* | *promiscuous*)	Matches isolation profile based on UFDB. Valid only in the egress policy table.
src-mac-addr-state (*dynamic-station-move* | *sa-found* | *sa-not-found* | *static-station-move*)	Defines whether to match packets with registered state - packets whose destination MAC address is in UFDB/MFDB/RFDB. Valid only in the egress policy table.
flow-id (*0..63*)	

ACL rule action part.

Property	Description
action (*copy-to-cpu* | *drop* | *forward* | *redirect-to-cpu* | *send-to-new-dst-ports*; Default: **forward**)	• copy-to-cpu - Packets are copied to the CPU if they match the ACL conditions. • drop - Packets are dropped if they match the ACL conditions. • forward - Packets are forwarded if they match the ACL conditions. • redirect-to-cpu - Packets are redirected to the CPU if they match the ACL conditions. • send-to-new-dst-ports - Packets are sent to new destination ports if they match the ACL conditions.
new-dst-ports (*ports,trunks*)	If the action is "send-to-new-dst-ports", then this property sets which ports/trunks are the new destinations.

mirror-to (*mirror0* | *mirror1*)	Mirroring destination for ACL packets.
policer (*policer*)	Applied ACL Policer for ACL packets.
src-mac-learn (*yes* | *no*)	Whether to learn the source MAC of the matched ACL packets. Valid only in the ingress policy table.
new-service-vid (*0..4095*)	New service VLAN ID for ACL packets.
new-service-pcp (*0..7*)	New service PCP for ACL packets.
new-service-dei (*0..1*)	New service DEI for ACL packets.
new-customer-vid (*0..4095*)	New customer VLAN ID for ACL packets. If set to 4095, then traffic is dropped.
new-customer-pcp (*0..7*)	New customer PCP for ACL packets.
new-customer-dei (*0..1*)	New customer DEI for ACL packets.
new-dscp (*0..63*)	New DSCP for ACL packets.
new-priority (*0..15*)	New internal priority for ACL packets.
new-drop-precedence (*drop* | *green* | *red* | *yellow*)	New internal drop precedence for ACL packets.
new-registered-state (*yes* | *no*)	Whether to modify packet status. YES sets packet status to registered, NO - unregistered. Valid only in the ingress policy table.
new-flow-id (*0..63*)	

Filter bypassing part for ACL packets.

Property	Description
attack-filter-bypass (*yes* | *no*; Default: **no**)	
ingress-vlan-filter-bypass (*yes* | *no*; Default: **no**)	Allows bypassing ingress VLAN filtering in the VLAN table for matching packets. This applies only to the ingress policy table.
egress-vlan-filter-bypass (*yes* | *no*; Default: **no**)	Allows bypassing egress VLAN filtering in the VLAN table for matching packets. This applies only to the ingress policy table.
isolation-filter-bypass (*yes* | *no*; Default: **no**)	Allows bypassing the Isolation table for matching packets. This applies only to the ingress policy table.
egress-vlan-translate-bypass (*yes* | *no*; Default: **no**)	Allows bypassing egress VLAN translation table for matching packets.

ACL Policer

Sub-menu: `/interface ethernet switch acl policer`

Property	Description
name (*string*; Default: **policerX**)	Name of the Policer used in ACL.
yellow-rate (*integer*)	Maximum data rate limit for packets with yellow drop precedence.
yellow-burst (*integer*; Default: **0**)	Maximum data rate which can be transmitted while the burst is allowed for packets with yellow drop precedence.
red-rate (*integer*); Default: **0**)	Maximum data rate limit for packets with red drop precedence.
red-burst (*integer*; Default: **0**)	Maximum data rate which can be transmitted while the burst is allowed for packets with red drop precedence.
meter-unit (*bit* | *packet*; Default: **bit**)	Measuring units for ACL traffic rate.

meter-len (*layer-1* | *layer-2* | *layer-3*; Default: **layer-1**)	Packet classification which sets the packet byte length for metering. • layer-1 - includes entire layer-2 frame + FCS + inter-packet gap + preamble. • layer-2 - includes layer-2 frame + FCS. • layer-3 - includes only layer-3 + ethernet padding without layer-2 header and FCS.
color-awareness (*yes* | *no*; Default: **no**)	YES makes the policer to take into account pre-colored drop precedence, NO - ignores drop precedence.
bucket-coupling (*yes* | *no*; Default: **no**)	
yellow-action (*drop* | *forward* | *remark*; Default: **drop**)	Performed action for exceeded traffic with yellow drop precedence.
new-dei-for-yellow (*0..1* | *remap*)	New DEI for yellow drop precedence packets.
new-pcp-for-yellow (*0..7* | *remap*)	New PCP for yellow drop precedence packets.
new-dscp-for-yellow (*0..63* | *remap*)	New DSCP for yellow drop precedence packets.
red-action (*drop* | *forward* | *remark*; Default: **drop**)	Performed action for exceeded traffic with red drop precedence.
new-dei-for-red (*0..1* | *remap*)	New DEI for red drop precedence packets.
new-pcp-for-red (*0..7* | *remap*)	New PCP for red drop precedence packets.
new-dscp-for-red (*0..63* | *remap*)	New DSCP for red drop precedence packets.

See also

- CRS1xx/2xx series switches examples
- CRS Router
- CRS1xx/2xx VLANs with Trunks
- Basic VLAN switching
- Bridge Hardware Offloading
- Spanning Tree Protocol
- IGMP Snooping
- DHCP Snooping and Option 82
- MTU on RouterBOARD
- Layer2 misconfiguration

L3 Hardware Offloading

Introduction

Layer 3 Hardware Offloading (**L3HW**, otherwise known as IP switching or HW routing) allows to offload some router features onto the switch chip. This allows reaching wire speeds when routing packets, which would simply not be possible with the CPU.

Switch Configuration

To enable Layer 3 Hardware Offloading, set `l3-hw-offloading=yes` for the switch:

```
/interface/ethernet/switch set 0 l3-hw-offloading=yes
```

Switch Port Configuration

Layer 3 Hardware Offloading can be configured for each physical switch port. For example:

```
/interface/ethernet/switch/port set sfp-sfpplus1 l3-hw-offloading=yes
```

Note that l3hw settings for switch and ports are different:

- Setting `l3-hw-offloading=no` for the switch completely disables offloading - all packets will be routed by CPU.

- However, setting `l3-hw-offloading=no` for a switch port only disables hardware routing from/to this particular port. Moreover, the port can still participate in Fastrack connection offloading.

To enable full hardware routing, enable l3hw on all switch ports:

```
/interface/ethernet/switch set 0 l3-hw-offloading=yes
/interface/ethernet/switch/port set [find] l3-hw-offloading=yes
```

To make all packets go through the CPU first, and offload only the Fasttrack connections, disable l3hw on all ports but keep it enabled on the switch chip itself:

```
/interface/ethernet/switch set 0 l3-hw-offloading=yes
/interface/ethernet/switch/port set [find] l3-hw-offloading=no
```

Packets get routed by the hardware only if both source and destination ports have `l3-hw-offloading=yes`. If at least one of them has `l3-hw-offloading=no`, packets will go through the CPU/Firewall while offloading only the Fasttrack connections.

The next example enables hardware routing on all ports but the upstream port (sfp-sfpplus16). Packets going to/from sfp-sfpplus16 will enter the CPU and, therefore, subject to Firewall/NAT processing.

```
/interface/ethernet/switch set 0 l3-hw-offloading=yes
/interface/ethernet/switch/port set [find] l3-hw-offloading=yes
/interface/ethernet/switch/port set sfp-sfpplus16 l3-hw-offloading=no
```

The existing connections may be unaffected by the `l3-hw-offloading` setting change.

L3HW Settings

Basic Settings

The L3HW Settings menu has been introduced in RouterOS version 7.6.

Sub-menu: `/interface ethernet switch l3hw-settings`

Property	Description
autorestart (*yes* / *no*; Default: **no**)	Automatically restarts the l3hw driver in case of an error. Otherwise, if an error occurs, `l3-hw-offloading` gets disabled, and the error code is displayed in the switch settings and #monitor. Autorestart does not work for system failures, such as OOM (Out Of Memory).
fasttrack-hw (*yes* / *no*; Default: **yes** (if supported))	Enables or disables FastTrack HW Offloading. Keep it enabled unless HW TCAM memory reservation is required, e.g., for dynamic switch ACL rules creation. Not all switch chips support FastTrack HW Offloading (see **hw-supports-fasttrack**).
ipv6-hw (*yes* / *no*; Default: **no**)	Enables or disables IPv6 Hardware Offloading. Since IPv6 routes occupy a lot of HW memory, enable it only if IPv6 traffic speed is significant enough to benefit from hardware routing.
icmp-reply-on-error (*yes* / *no*; Default: **yes**)	Since the hardware cannot send ICMP messages, the packet must be redirected to the CPU to send an ICMP reply in case of an error (e.g., "Time Exceeded", "Fragmentation required", etc.). Enabling icmp-reply-on-error helps with network diagnostics but may open potential vulnerabilities for DDoS attacks. Disabling icmp-reply-on-error silently drops the packets on the hardware level in case of an error.

Read-Only Properties

Property	Description
hw-supports-fasttrack (*yes* / *no*)	Indicates if the hardware (switch chip) supports FastTrack HW Offloading.

Advanced Settings

This menu allows tweaking l3hw settings for specific use cases.

> It is NOT recommended to change the advanced L3HW settings unless instructed by MikroTik Support or MikroTik Certified Routing Engineer. Applying incorrect settings may break the L3HW operation.

Sub-menu: `/interface ethernet switch l3hw-settings` advanced

Property	Description
route-queue-limit-high (*number*; Default: **256**)	The switch driver stops route indexing when route-queue-size (see #monitor) exceeds this value. Lowering this value leads to faster route processing but increases the lag between a route's appearance in RouterOS and hardware memory. Setting route-queue-limit-high=0 disables route indexing when there are any routes in the processing queue - the most efficient CPU usage but the longest delay before hardware offloading. Useful when there are static routes only. Not recommended together with routing protocols (such as BGP or OSPF) when there are frequent routing table changes.
route-queue-limit-low (*number*; Default: **0**)	Re-enable route indexing when route-queue-size drops down to this value. Must not exceed the high limit. Setting route-queue-limit-low=0 tells the switch driver to process all pending routes before the next hw-offloading attempt. While this is the desired behavior, it may completely block the hw-offloading under a constant BGP feed.
shwp-reset-counter (*number*; Default: **128**)	Reset the Shortest HW Prefix (see ipv4-shortest-hw-prefix / ipv6-shortest-hw-prefix in #monitor) and try the full route table offloading after this amount of changes in the routing table. At a partial offload, when the entire routing table does not fit into the hardware memory and shorter prefixes are redirected to the CPU, there is no need to try offloading route prefixes shorter than SHWP since those will get redirected to the CPU anyway, theoretically. However, significant changes to the routing table may lead to a different index layout and, therefore, a different amount of routes that can be hw-offloaded. That's why it is recommended to do the full table re-indexing occasionally. Lowering this value may allow more routes to be hw-offloaded but increases CPU usage and vice-versa. Setting shwp-reset-counter=0 always does full re-indexing after each routing table change. This setting is used only during Partial Offloading and has no effect when ipv4-shortest-hw-prefix=0 (and ipv6, respectively).
partial-offload-chunk (*number*; Default: **1024**, min: 16)	The minimum number of routes for incremental adding in Partial Offloading. Depending on the switch chip model, routes are offloaded either as-is (each routing entry in RouterOS corresponds to an entry in the hardware memory) or getting indexed, and the index entries are the ones that are written into the hardware memory. This setting is used only for the latter during Partial Offloading. Depending on index fragmentation, a single IPv4 route addition can occupy from -3 to +6 LPM blocks of HW memory (some route addition may lower the amount of required HW memory thanks to index defragmentation). Hence, it is impossible to predict the exact number of routes that may fit in the hardware memory. The switch driver uses a binary split algorithm to find the maximum number of routes that fit in the hardware. Let's imagine 128k routes, all of them not fitting into the hardware memory. The algorithm halves the number and tries offloading 64k routes. Let's say offloading succeeded. In the next iteration, the algorithm picks 96k, let's say it fails; then 80k - fails again, 72k - succeeds, 76k, etc. until the difference between succeeded and failed numbers drops below the partial-offload-chunk value. Lowering the partial-offload-chunk value increases the number of hw-offloaded routes but also raises CPU usage and vice-versa.
route-index-delay-min (*time*; Default: **1s**)	The minimum delay between route processing and its offloading. The delay allows processing more routes together and offloading them at once, saving CPU usage. It also makes offloading the entire routing table faster by reducing the per-route processing work. On the other hand, it slows down the offloading of an individual route. If an additional route is received during the delay, the latter resets to the route-index-delay-min value. Adding more and more routes within the delay keeps resetting the timer until the route-index-delay-max is reached.
route-index-delay-max (*time*; Default: **10s**)	The maximum delay between route processing and its offloading. When the maximum delay is reached, the processed routes get offloaded despite more routes pending. However, route-queue-limit-high has higher priority than this, meaning that the indexing/offloading gets paused anyway when a certain queue size is reached.

neigh-keepalive-interval (*time*; Default: **15s**, min: 5s)	Neighbor (host) keepalive interval. When a host (IP neighbor) gets hw-offloaded, all traffic from/to it is routed by the switch chip, and RouterOS may think the neighbor is inactive and delete it. To prevent that, the switch driver must keep the offloaded neighbors alive by sending periodical refreshes to RouterOS.
neigh-discovery-interval (*time*; Default: **1m37s**, min: 30s)	Unfortunately, switch chips do not provide per-neighbor stats. Hence, the only way to check if the offloaded host is still active is by sending occasional ARP (IPv4) / Neighbor Discovery (IPv6) requests to the connected network. Increasing the value lowers the broadcast traffic but may leave inactive hosts in hardware memory for longer. Neighbor discovery is triggered within the neighbor keepalive work. Hence, the discovery time is rounded up to the next keepalive session. Choose a value for neigh-discovery-interval not dividable by neigh-keepalive-interval to send ARP/ND requests in various sessions, preventing broadcast bursts.
neigh-discovery-burst-limit (*number*; Default: **64**)	The maximum number of ARP/ND requests that can be sent at once.
neigh-discovery-burst-delay (*time*; Default: **300ms**, min: 10ms)	The delay between ARP/ND subsequent bursts if the number of requests exceeds neigh-discovery-burst-limit.

Some settings only apply to certain switch models.

Monitor

The L3HW Monitor feature has been introduced in RouterOS version 7.10. It allows monitoring of switch chip and driver stats related to L3HW.

```
/interface/ethernet/switch/l3hw-settings/monitor
        ipv4-routes-total: 99363
           ipv4-routes-hw: 61250
          ipv4-routes-cpu: 38112
  ipv4-shortest-hw-prefix: 24
              ipv4-hosts: 87
        ipv6-routes-total: 15
           ipv6-routes-hw: 11
          ipv6-routes-cpu: 4
  ipv6-shortest-hw-prefix: 0
              ipv6-hosts: 7
         route-queue-size: 118
     fasttrack-ipv4-conns: 2031
   fasttrack-hw-min-speed: 0
             nexthop-cap: 8192
           nexthop-usage: 93
```

Stats

Property	Description
ipv4-routes-total	The total number of IPv4 routes handled by the switch driver.

ipv4-routes-hw	The number of hardware-offloaded IPv4 routes (a.k.a. hardware routes)
ipv4-routes-cpu	The number of IPv4 routes redirected to the CPU (a.k.a. software routes)
ipv4-shortest-hw-prefix	*Shortest Hardware Prefix (SHWP)* for IPv4. If the entire IPv4 routing table does not fit into the hardware memory, *partial offloading* is applied, where the longest prefixes are hw-offloaded while the shorter ones are redirected to the CPU. This field shows the shortest route prefix (/x) that is offloaded to the hardware memory. All prefixes shorter than this are processed by the CPU. "`ipv4-shortest-hw-prefix=0`" means the entire IPv4 routing table is offloaded to the hardware memory.
ipv4-hosts	The number of hardware-offloaded IPv4 hosts (/32 routes)
ipv6-routes-total [1]	The total number of IPv6 routes handled by the switch driver.
ipv6-routes-hw [1]	The number of hardware-offloaded IPv6 routes (a.k.a. hardware routes)
ipv6-routes-cpu [1]	The number of IPv6 routes redirected to the CPU (a.k.a. software routes)
ipv6-shortest-hw-prefix [1]	*Shortest Hardware Prefix (SHWP)* for IPv6. If the entire IPv6 routing table does not fit into the hardware memory, *partial offloading* is applied, where the longest prefixes are hw-offloaded while the shorter ones are redirected to the CPU. This field shows the shortest route prefix (/x) that is offloaded to the hardware memory. All prefixes shorter than this are processed by the CPU. "`ipv6-shortest-hw-prefix=0`" means the entire IPv6 routing table is offloaded to the hardware memory.
ipv6-hosts [1]	The number of hardware-offloaded IPv6 hosts (/128 routes)
route-queue-size	The number of routes in the queue for processing by the switch chip driver. Under normal working conditions, this field is 0, meaning that all routes are processed by the driver.
fasttrack-ipv4-conns [2]	The number of hardware-offloaded FastTrack connections.
fasttrack-hw-min-speed [2]	When the hardware memory for storing FastTrack is full, this field shows the minimum speed (in bytes per second) of a hw-offloaded FastTrack connection. Slower connections are routed by the CPU.

[1] *IPv6 stats appear only when IPv6 hardware routing is enabled (`ipv6-hw=yes`)*

[2] *FastTrack stats appear only when hardware offloading of FastTrack connections is enabled (`fasttrack-hw=yes`)*

Advanced Monitor

An enhanced version of Monitor with extra telemetry data for advanced users. Advanced Monitor contains all data from the basic monitor plus the fields listed below.

```
/interface/ethernet/switch/l3hw-settings/advanced> monitor once
       ipv4-routes-total: 29968
          ipv4-routes-hw: 29957
         ipv4-routes-cpu: 11
  ipv4-shortest-hw-prefix: 0
              ipv4-hosts: 3
       ipv6-routes-total: 4
          ipv6-routes-hw: 0
         ipv6-routes-cpu: 4
  ipv6-shortest-hw-prefix: 0
              ipv6-hosts: 0
        route-queue-size: 0
        route-queue-rate: 0
     route-process-rate: 0
    fasttrack-ipv4-conns: 0
    fasttrack-queue-size: 0
    fasttrack-queue-rate: 0
   fasttrack-process-rate: 0
   fasttrack-hw-min-speed: 0
   fasttrack-hw-offloaded: 0
    fasttrack-hw-unloaded: 0
                lpm-cap: 54560
              lpm-usage: 31931
           lpm-bank-cap: 2728
          lpm-bank-usage: 46,0,0,0,2589,2591,1983,0,2728,2728,2728,2728,2728,2728,2728,2728,2728,170,0,0
                 pbr-cap: 8192
              pbr-usage: 0
            pbr-lpm-bank: 3
              nat-usage: 0
            nexthop-cap: 8192
          nexthop-usage: 85
```

Stats

Property	Description
route-queue-rate	The rate at which routes are added to the queue for the switch driver processing. In other words, the growth rate of route-queue-size (routes per second)
route-process-rate	The rate at which previously queued routes are processed by the switch driver. In other words, the shrink rate of route-queue-size (routes per second)
fasttrack-queue-size	The number of FastTrack connections in the queue for processing by the switch chip driver.
fasttrack-queue-rate	The rate at which FastTrack connections are added to the queue for the switch driver processing. In other words, the growth rate of fasttrack-queue-size (connections per second)
fasttrack-process-rate	The rate at which previously queued FastTrack connections are processed by the switch driver. In other words, the shrink rate of fasttrack-queue-size (connections per second)
fasttrack-hw-offloaded	The number of FastTrack connections offloaded to the hardware. The counter resets every second (or every monitor interval).
fasttrack-hw-unloaded	The number of FastTrack connections unloaded from the hardware (redirected to software routing). The counter resets every second (or every monitor interval).
lpm-cap	The size of the LPM hardware table (LPM = Longest Prefix Match). LPM stores route indexes for hardware routing. Not every switch chip model uses LPM. Others use TCAM.
lpm-usage	The number of used LPM blocks. lpm-usage / lpm-cap = usage percentage.

lpm-bank-cap	LPM memory is organized in banks - special memory units. The bank size depends on the switch chip model. This value shows the size of a single bank (in LPM blocks). lpm-cap / lpm-bank-cap = the number of banks (usually, 20).
lpm-bank-usage	Per-bank LPM usage (in LPM blocks)
pbr-cap	The size of the Policy-Based Routing (PBR) hardware table. PBR is used for NAT offloading of FastTrack connections.
pbr-usage	The number of used PBR entries. pbr-usage / pbr-cap = usage percentage.
pbr-lpm-bank	PBR shares LPM memory banks with routing tables. This value shows the LPM bank index shared with PBR (0 = the first bank).
nat-usage	The number of used NAT hardware entries (for FastTrack connections).

Interface Lists

It is impossible to use interface lists directly to control `l3-hw-offloading` because an interface list may contain virtual interfaces (such as VLAN) while the `l3-hw-offloading` setting must be applied to physical switch ports only. For example, if there are two VLAN interfaces (vlan20 and vlan30) running on the same switch port (trunk port), it is impossible to enable hardware routing on vlan20 but keep it disabled on vlan30.

However, an interface list may be used as a port selector. The following example demonstrates how to enable hardware routing on LAN ports (ports that belong to the "LAN" interface list) and disable it on WAN ports:

```
:foreach i in=[/interface/list/member/find where list=LAN] do={
    /interface/ethernet/switch/port set [/interface/list/member/get $i interface] l3-hw-offloading=yes
}

:foreach i in=[/interface/list/member/find where list=WAN] do={
    /interface/ethernet/switch/port set [/interface/list/member/get $i interface] l3-hw-offloading=no
}
```

Please take into account that since interface lists are not directly used in hardware routing control., **modifying the interface list also does not automatically reflect in l3hw changes**. For instance, adding a switch port to the "LAN" interface list does not automatically enable `l3-hw-offloading` on it. The user has to rerun the above script to apply the changes.

MTU

The hardware supports up to 8 MTU profiles, meaning that the user can set up to 8 different MTU values for interfaces: the default 1500 + seven custom ones.

> It is recommended to disable `l3-hw-offloading` while changing the MTU/L2MTU values on the interfaces.

MTU Change Example

```
/interface/ethernet/switch set 0 l3-hw-offloading=no
/interface set sfp-sfpplus1 mtu=9000 l2mtu=9022
/interface set sfp-sfpplus2 mtu=9000 l2mtu=9022
/interface set sfp-sfpplus3 mtu=10000 l2mtu=10022
/interface/ethernet/switch set 0 l3-hw-offloading=yes
```

Layer 2 Dependency

Layer 3 hardware processing lies on top of Layer 2 hardware processing. Therefore, L3HW offloading requires L2HW offloading on the underlying interfaces. The latter is enabled by default, but there are some exceptions. For example, CRS3xx devices support only one hardware bridge. If there are multiple bridges, others are processed by the CPU and are not subject to L3HW.

Another example is ACL rules. If a rule redirects traffic to the CPU for software processing, then hardware routing (L3HW) is not triggered:

ACL rule to disable hardware processing on a specific port

```
/interface/ethernet/switch/rule/add switch=switch1 ports=ether1 redirect-to-cpu=yes
```

It is recommended to turn off L3HW offloading during L2 configuration.

To make sure that Layer 3 is in sync with Layer 2 on both the software and hardware sides, we recommend disabling L3HW while configuring Layer 2 features. The recommendation applies to the following configuration:

- adding/removing/enabling/disabling bridge;
- adding/removing switch ports to/from the bridge;
- bonding switch ports / removing bond;
- changing VLAN settings;
- changing MTU/L2MTU on switch ports;
- changing ethernet (MAC) addresses.

In short, disable `l3-hw-offloading` while making changes under `/interface/bridge/` and `/interface/vlan/`:

Layer 2 Configuration Template

```
/interface/ethernet/switch set 0 l3-hw-offloading=no

/interface/bridge
# put bridge configuration changes here

/interface/vlan
# define/change VLAN interfaces

/interface/ethernet/switch set 0 l3-hw-offloading=yes
```

MAC telnet and RoMON

There is a limitation for MAC telnet and RoMON when L3HW offloading is enabled on **98DX8xxx**, **98DX4xxx,** or **98DX325x** switch chips. Packets from these protocols are dropped and do not reach the CPU, thus access to the device will fail.

If MAC telnet or RoMON are desired in combination with L3HW, certain ACL rules can be created to force these packets to the CPU.

For example, if MAC telnet access on sfp-sfpplus1 and sfp-sfpplus2 is needed, you will need to add this ACL rule. It is possible to select even more interfaces with the `ports` setting.

```
/interface ethernet switch rule
add dst-port=20561 ports=sfp-sfpplus1,sfp-sfpplus2 protocol=udp redirect-to-cpu=yes switch=switch1
```

For example, if RoMON access on sfp-sfpplus2 is needed, you will need to add this ACL rule.

```
/interface ethernet switch rule
add mac-protocol=0x88BF ports=sfp-sfpplus2 redirect-to-cpu=yes switch=switch1
```

Inter-VLAN Routing

Since L3HW depends on L2HW, and L2HW is the one that does VLAN processing, Inter-VLAN *hardware* routing requires a hardware bridge underneath. Even if a particular VLAN has only one tagged port member, the latter must be a bridge member. Do not assign a VLAN interface directly on a switch port! Otherwise, L3HW offloading fails and the traffic will get processed by the CPU:

~~`/interface/vlan add interface=ether2 name=vlan20 vlan-id=20`~~

Assign the VLAN interface to the bridge instead. This way, VLAN configuration gets offloaded to the hardware, and, with L3HW enabled, the traffic is subject to inter-VLAN hardware routing.

VLAN Configuration Example

```
/interface/ethernet/switch set 0 l3-hw-offloading=no
/interface/bridge/port add bridge=bridge interface=ether2
/interface/bridge/vlan add bridge=bridge tagged=bridge,ether2 vlan-ids=20
/interface/vlan add interface=bridge name=vlan20 vlan-id=20
/ip/address add address=192.0.2.1/24 interface=vlan20
/interface/bridge set bridge vlan-filtering=yes
/interface/ethernet/switch set 0 l3-hw-offloading=yes
```

For Inter-VLAN routing, the bridge interface must be a tagged member of every routable `/interface/bridge/vlan/` entry.

L3HW MAC Address Range Limitation (DX2000/DX3000 series only)

Marvell Prestera DX2000 and DX3000 switch chips have a hardware limitation that allows configuring only the last (least significant) octet of the MAC address for each interface. The other five (most significant) octets are configured globally and, therefore, must be equal for all interfaces (switch ports, bridge, VLANs). In other words, the MAC addresses must be in the format "**XX:XX:XX:XX:XX:??**", where:

- "**XX:XX:XX:XX:XX**" part is common for all interfaces.
- "**??**" is a variable part.

This requirement applies only to Layer 3 (routing). Layer 2 (bridging) does not use the switch's ethernet addresses. Moreover, it does not apply to bridge ports because they use the bridge's MAC address.

The requirement for common five octets applies to:

- Standalone switch ports (not bridge members) with hardware routing enabled (`l3-hw-offloading=yes`).
- Bridge itself.
- VLAN interfaces (those that use the bridge's MAC address by default).

Route Configuration

Suppressing HW Offload

By default, all the routes are participating to be hardware candidate routes. To further fine-tune which traffic to offload, there is an option for each route to disable/enable **`suppress-hw-offload`**.

For example, if we know that the majority of traffic flows to the network where servers are located, we can enable offloading only to that specific destination:

```
/ip/route set [find where static && dst-address!="192.168.3.0/24"] suppress-hw-offload=yes
```

Now only the route to 192.168.3.0/24 has H-flag, indicating that it will be the only one eligible to be selected for HW offloading:

```
[admin@MikroTik] > /ip/route print where static
Flags: A - ACTIVE; s - STATIC, y - COPY; H - HW-OFFLOADED
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#     DST-ADDRESS       GATEWAY         D
0 As  0.0.0.0/0         172.16.2.1      1
1 As  10.0.0.0/8        10.155.121.254  1
2 AsH 192.168.3.0/24    172.16.2.1      1
```

H-flag does not indicate that the route is actually HW offloaded, it indicates only that the route can be selected to be HW offloaded.

Routing Filters

For dynamic routing protocols like OSFP and BGP, it is possible to suppress HW offloading using routing filters. For example, to suppress HW offloading on all OSFP instance routes, use "**`suppress-hw-offload yes`**" property:

```
/routing/ospf/instance
set [find name=instance1] in-filter-chain=ospf-input
/routing/filter/rule
add chain="ospf-input" rule="set suppress-hw-offload yes; accept"
```

Offloading Fasttrack Connections

Firewall filter rules have **`hw-offload`** option for Fasttrack, allowing fine-tuning connection offloading. Since the hardware memory for Fasttrack connections is very limited, we can choose what type of connections to offload and, therefore, benefit from near-the-wire-speed traffic. The next example offloads only TCP connections while UDP packets are routed via the CPU and do not occupy HW memory:

```
/ip/firewall/filter
add action=fasttrack-connection chain=forward connection-state=established,related hw-offload=yes protocol=tcp
add action=fasttrack-connection chain=forward connection-state=established,related hw-offload=no
add action=accept chain=forward connection-state=established,related
```

Stateless Hardware Firewall

While connection tracking and stateful firewalling can be performed only by the CPU, the hardware can perform stateless firewalling via switch rules (ACL). The next example prevents (on a hardware level) accessing a MySQL server from the ether1, and redirects to the CPU/Firewall packets from ether2 and ether3:

```
/interface ethernet switch rule
add switch=switch1 dst-address=10.0.1.2/32 dst-port=3306 ports=ether1 new-dst-ports=""
add switch=switch1 dst-address=10.0.1.2/32 dst-port=3306 ports=ether2,ether3 redirect-to-cpu=yes
```

Switch Rules (ACL) vs. Fasttrack HW Offloading

Some firewall rules may be implemented both via switch rules (ACL) and CPU Firewall Filter + Fasttrack HW Offloading. Both options grant near-the-wire-speed performance. So the question is which one to use?

First, not all devices support Fasttrack HW Offloading, and without HW offloading, Firewall Filter uses only software routing, which is dramatically slower than its hardware counterpart. Second, even if Fasttrack HW Offloading is an option, a rule of thumb is:

> Always use Switch Rules (ACL), if possible.

Switch rules share the hardware memory with Fastrack connections. However, hardware resources are allocated for each Fasttrack connection while a single ACL rule can match multiple connections. For example, if you have a guest WiFi network connected to sfp-sfpplus1 VLAN 10 and you don't want it to access your internal network, simply create an ACL rule:

```
/interface/ethernet/switch/rule
add switch=switch1 ports=sfp-sfpplus1 vlan-id=10 dst-address=10.0.0.0/8 new-dst-ports=""
```

The matched packets will be dropped on the hardware level. It is much better than letting *all* guest packets to the CPU for Firewall filtering.

Of course, ACL rules cannot match everything. For instance, ACL rules cannot filter connection states: accept established, drop others. That is where Fasttrack HW Offloading gets into action - redirect the packets to the CPU by default for firewall filtering, then offload the established Fasttrack connections. However, disabling `l3-hw-offloading` for the entire switch, port is not the only option.

Define ACL rules with **`redirect-to-cpu=yes`** instead of setting `l3-hw-offloading=no` of the switch port for narrowing down the traffic that goes to the CPU.

Configuration Examples

Inter-VLAN Routing with Upstream Port Behind Firewall/NAT

This example demonstrates how to benefit from near-to-wire-speed inter-VLAN routing while keeping Firewall and NAT running on the upstream port. Moreover, Fasttrack connections to the upstream port get offloaded to hardware as well, boosting the traffic speed close to wire-level. Inter-VLAN traffic is fully routed by the hardware, not entering the CPU/Firewall, and, therefore, not occupying the hardware memory of Fasttrack connections.

We use the **CRS317-1G-16S+** model with the following setup:

- sfp1-sfp4 - bridged ports, VLAN ID 20, untagged
- sfp5-sfp8 - bridged ports, VLAN ID 30, untagged
- sfp16 - the upstream port
- ether1 - management port

Setup interface lists for easy access:

Interface Lists

```
/interface list
add name=LAN
add name=WAN
add name=MGMT

/interface list member
add interface=sfp-sfpplus1 list=LAN
add interface=sfp-sfpplus2 list=LAN
add interface=sfp-sfpplus3 list=LAN
add interface=sfp-sfpplus4 list=LAN
add interface=sfp-sfpplus5 list=LAN
add interface=sfp-sfpplus6 list=LAN
add interface=sfp-sfpplus7 list=LAN
add interface=sfp-sfpplus8 list=LAN
add interface=sfp-sfpplus16 list=WAN
add interface=ether1 list=MGMT
```

Bridge Setup

```
/interface bridge
add name=bridge vlan-filtering=yes

/interface bridge port
add bridge=bridge interface=sfp-sfpplus1 pvid=20
add bridge=bridge interface=sfp-sfpplus2 pvid=20
add bridge=bridge interface=sfp-sfpplus3 pvid=20
add bridge=bridge interface=sfp-sfpplus4 pvid=20
add bridge=bridge interface=sfp-sfpplus5 pvid=30
add bridge=bridge interface=sfp-sfpplus6 pvid=30
add bridge=bridge interface=sfp-sfpplus7 pvid=30
add bridge=bridge interface=sfp-sfpplus8 pvid=30

/interface bridge vlan
add bridge=bridge tagged=bridge untagged=sfp-sfpplus1,sfp-sfpplus2,sfp-sfpplus3,sfp-sfpplus4 vlan-ids=20
add bridge=bridge tagged=bridge untagged=sfp-sfpplus5,sfp-sfpplus6,sfp-sfpplus7,sfp-sfpplus8 vlan-ids=30
```

Routing requires dedicated VLAN interfaces. For standard L2 VLAN bridging (without inter-VLAN routing), the next step can be omitted.

VLAN Interface Setup for Routing

```
/interface vlan
add interface=bridge name=vlan20 vlan-id=20
add interface=bridge name=vlan30 vlan-id=30

/ip address
add address=192.168.20.17/24 interface=vlan20 network=192.168.20.0
add address=192.168.30.17/24 interface=vlan30 network=192.168.30.0
```

Configure management and upstream ports, a basic firewall, NAT, and enable hardware offloading of Fasttrack connections:

Firewall Setup

```
/ip address
add address=192.168.88.1/24 interface=ether1
add address=10.0.0.17/24 interface=sfp-sfpplus16

/ip route
add gateway=10.0.0.1

/ip firewall filter
add action=fasttrack-connection chain=forward connection-state=established,related hw-offload=yes
add action=accept chain=forward connection-state=established,related

/ip firewall nat
add action=masquerade chain=srcnat out-interface-list=WAN
```

At this moment, all routing still is performed by the CPU. Enable hardware routing on the switch chip:

Enable Layer 3 Hardware Offloading

```
# Enable full hardware routing on LAN ports
:foreach i in=[/interface/list/member/find where list=LAN] do={
    /interface/ethernet/switch/port set [/interface/list/member/get $i interface] l3-hw-offloading=yes
}

# Disable full hardware routing on WAN or Management ports
:foreach i in=[/interface/list/member/find where list=WAN or list=MGMT] do={
    /interface/ethernet/switch/port set [/interface/list/member/get $i interface] l3-hw-offloading=no
}

# Activate Layer 3 Hardware Offloading on the switch chip
/interface/ethernet/switch/set 0 l3-hw-offloading=yes
```

Results:

- Within the same VLAN (e.g., sfp1-sfp4), traffic is forwarded by the hardware on Layer 2 *(L2HW)*.
- Inter-VLAN traffic (e.g. sfp1-sfp5) is routed by the hardware on Layer 3 *(L3HW)*.
- Traffic from/to the WAN port gets processed by the CPU/Firewall first. Then Fasttrack connections get offloaded to the hardware *(Hardware-Accelerated L4 Stateful Firewall)*. NAT applies both on CPU- and HW-processed packets.
- Traffic to the management port is protected by the Firewall.

Typical Misconfiguration

Below are typical user errors in configuring Layer 3 Hardware Offloading.

VLAN interface on a switch port or bond

```
/interface/vlan
add name=vlan10 vlan-id=10 interface=sfp-sfpplus1
add name=vlan20 vlan-id=20 interface=bond1
```

VLAN interface must be set on the bridge due to Layer 2 Dependency. Otherwise, L3HW will not work. The correct configuration is:

```
/interface/bridge/port
add bridge=bridge1 interface=sfp-sfpplus1 frame-types=admit-only-vlan-tagged
add bridge=bridge1 interface=bond1 frame-types=admit-only-vlan-tagged

/interface/bridge/vlan
add bridge=bridge1 tagged=bridge1,sfp-sfpplus1 vlan-ids=10
add bridge=bridge1 tagged=bridge1,bond1 vlan-ids=20

/interface/vlan
add name=vlan10 vlan-id=10 interface=bridge1
add name=vlan20 vlan-id=20 interface=bridge1
```

Not adding the bridge interface to /interface/bridge/vlan/

For Inter-VLAN routing, the bridge interface itself needs to be added to the tagged members of the given VLANs. In the next example, Inter-VLAN routing works between VLAN 10 and 11, but packets are NOT routed to VLAN 20.

```
/interface bridge vlan
add bridge=bridge1 vlan-ids=10 tagged=bridge1,sfp-sfpplus1
add bridge=bridge1 vlan-ids=11 tagged=bridge1 untagged=sfp-sfpplus2,sfp-sfpplus3
add bridge=bridge1 vlan-ids=20 tagged=sfp-sfpplus1 untagged=sfp-sfpplus4,sfp-sfpplus5
```

The above example does not always mean an error. Sometimes, you may want the device to act as a simple L2 switch in some/all VLANs. Just make sure you set such behavior on purpose, not due to a mistake.

Creating multiple bridges

The devices support only one hardware bridge. If there are multiple bridges created, only one gets hardware offloading. While for L2 that means software forwarding for other bridges, in the case of L3HW, multiple bridges may lead to undefined behavior.

Instead of creating multiple bridges, create one and segregate L2 networks with VLAN filtering.

Using ports that do not belong to the switch

Some devices have two switch chips or the management port directly connected to the CPU. For example, **CRS312-4C+8XG** has an **ether9** port connected to a separate switch chip. Trying to add this port to a bridge or involve it in the L3HW setup leads to unexpected results. Leave the management port for management!

```
[admin@crs312] /interface/ethernet/switch> print
Columns: NAME, TYPE, L3-HW-OFFLOADING
# NAME     TYPE              L3-HW-OFFLOADING
0 switch1  Marvell-98DX8212  yes
1 switch2  Atheros-8227      no

[admin@crs312] /interface/ethernet/switch> port print
Columns: NAME, SWITCH, L3-HW-OFFLOADING, STORM-RATE
 # NAME         SWITCH   L3-HW-OFFLOADING  STORM-RATE
 0 ether9       switch2
 1 ether1       switch1  yes                      100
 2 ether2       switch1  yes                      100
 3 ether3       switch1  yes                      100
 4 ether4       switch1  yes                      100
 5 ether5       switch1  yes                      100
 6 ether6       switch1  yes                      100
 7 ether7       switch1  yes                      100
 8 ether8       switch1  yes                      100
 9 combo1       switch1  yes                      100
10 combo2       switch1  yes                      100
11 combo3       switch1  yes                      100
12 combo4       switch1  yes                      100
13 switch1-cpu  switch1                           100
14 switch2-cpu  switch2
```

Relying on Fasttrack HW Offloading too much

Since Fasttrack HW Offloading offers near-the-wire-speed performance at zero configuration overhead, the users are tempted to use it as the default solution. However, the number of HW Fasttrack connections is very limited, leaving the other traffic for the CPU. Try using the hardware routing as much as possible, reduce the CPU traffic to the minimum via switch ACL rules, and then fine-tune which Fasttrack connections to offload with firewall filter rules.

Trying to offload slow-path connections

Using certain configurations (e.g. enabling bridge "**use-ip-firewall**" setting, creating bridge nat/filter rules) or running specific features like sniffer or torch can disable RouterOS FastPath, which will affect the ability to properly FastTrack and HW offload connections. If HW offloaded Fasttrack is required, make sure that there are no settings that disable the FastPath and verify that connections are getting the "H" flag or use the L3HW monitor command to see the amount of HW offloaded connections.

L3HW Feature Support

- **HW** - the feature is supported and offloaded to the hardware.
- **CPU** - the feature is supported but performed by software (CPU)
- **N/A** - the feature is not available together with L3HW. Layer 3 hardware offloading must be completely disabled (**switch** `l3-hw-offloading=no`) to make this feature work.
- **FW** - the feature requires `l3-hw-offloading=no` for a given **switch port**. On the **switch** level, `l3-hw-offloading=yes`.

Feature	Support	Comments	Release
IPv4 Unicast Routing	HW		7.1
IPv6 Unicast Routing	HW	`/interface/ethernet/switch/l3hw-settings/set ipv6-hw=yes`	7.6
IPv4 Multicast Routing	CPU		
IPv6 Multicast Routing	CPU		

ECMP	HW	Multipath routing	7.1
Blackholes	HW	`/ip/route add dst-address=10.0.99.0/24 blackhole`	7.1
gateway=<interface_name>	CPU/HW	`/ip/route add dst-address=10.0.0.0/24 gateway=ether1` This works only for directly connected networks. Since HW does not know how to send ARP requests, CPU sends an ARP request and waits for a reply to find out the DST MAC address on the first received packet of the connection that matches a DST IP address. After DST MAC is determined, HW entry is added and all further packets will be processed by the switch chip.	7.1
BRIDGE	HW	IP Routing from/to hardware-offloaded bridge interface.	7.1
VLAN	HW	Routing between VLAN interfaces that are created on hardware-offloaded bridge interface with vlan-filtering.	7.1
Bonding	HW	`/interface/bonding`	7.1
IPv4 Firewall	FW	Users must choose either HW-accelerated routing or firewall. Firewall rules get processed by the CPU. ***Fasttrack*** connections get offloaded to HW.	7.1
IPv4 NAT	FW	NAT rules applied to the offloaded ***Fasttrack*** connections get processed by HW too.	7.1
MLAG	N/A		
VRF	N/A	Only the **main** routing table gets offloaded. If VRF is used together with L3HW and packets arrive on a switch port with `l3-hw-offloading=yes`, packets can be incorrectly routed through the main routing table. To avoid this, disable L3HW on needed switch ports or use ACL rules to redirect specific traffic to the CPU.	
VRRP	N/A		
Controller Bridge and Port Extender	N/A		
VXLAN	CPU		
MTU	HW	The hardware supports up to 8 MTU profiles.	7.1
QinQ and tag-stacking	CPU	Stacked VLAN interfaces will lose HW offloading, while other VLANs created directly on the bridge interface can still use HW offloading.	

Only the devices listed in the table below support L3 HW Offloading.

L3HW Device Support

Only the devices listed in the table below support L3 HW Offloading.

CRS3xx: Switch DX3000 and DX2000 Series

The devices below are based on **Marvell 98DX224S, 98DX226S**, or **98DX3236** switch chip models. These devices do not support Fasttrack or NAT connection offloading.

> The **98DX3255** and **98DX3257** models are exceptions, which have a feature set of the DX8000 rather than the DX3000 series.

Model	Switch Chip	Release	IPv4 Route Prefixes[1]	IPv6 Route Prefixes[2]	Nexthops	ECMP paths per prefix[3]
CRS305-1G-4S+	98DX3236	7.1	13312	3328	4K	8
CRS310-1G-5S-4S+	98DX226S	7.1	13312	3328	4K	8
CRS310-8G+2S+	98DX226S	7.1	13312	3328	4K	8
CRS318-1Fi-15Fr-2S	98DX224S	7.1	13312	3328	4K	8
CRS318-16P-2S+	98DX226S	7.1	13312	3328	4K	8
CRS326-24G-2S+	98DX3236	7.1	13312	3328	4K	8
CRS328-24P-4S+	98DX3236	7.1	13312	3328	4K	8
CRS328-4C-20S-4S+	98DX3236	7.1	13312	3328	4K	8

[1] *Since the total amount of routes that can be offloaded is limited, prefixes with higher netmask are preferred to be forwarded by hardware (e.g., /32, /30, /29, etc.), any other prefixes that do not fit in the HW table will be processed by the CPU. Directly connected hosts are offloaded as /32 (IPv4) or /128 (IPv6) route prefixes. The number of hosts is also limited by max-neighbor-entries in IP Settings / IPv6 Settings.*

[2] *IPv4 and IPv6 routing tables share the same hardware memory.*

[3] *If a route has more paths than the hardware ECMP limit (X), only the first X paths get offloaded.*

CRS3xx, CRS5xx: Switch DX8000 and DX4000 Series

The devices below are based on **Marvell 98DX8xxx**, **98DX4xxx** switch chips, or **98DX325x** model.

Model	Switch Chip	Release	IPv4 Routes [1]	IPv4 Hosts [7]	IPv6 Routes[8]	IPv6 Hosts[7]	Nexthops	Fasttrack connections [2,3,4]	NAT entries [2,5]
CRS317-1G-16S+	**98DX8216**	7.1	120K - 240K	64K	30K - 40K	32K	8K	4.5K	4K
CRS309-1G-8S+	**98DX8208**	7.1	16K - 36K	16K	4K - 6K	8K	8K	4.5K	3.9K
CRS312-4C+8XG	**98DX8212**	7.1	16K - 36K	16K	4K - 6K	8K	8K	2.25K	2.25K
CRS326-24S+2Q+	**98DX8332**	7.1	16K - 36K	16K	4K - 6K	8K	8K	2.25K	2.25K
CRS354-48G-4S+2Q+, CRS354-48P-4S+2Q+	**98DX3257** [6]	7.1	16K - 36K	16K	4K - 6K	8K	8K	2.25K	2.25K
CRS504-4XQ	**98DX4310**	7.1	60K - 120K	64K	15K - 20K	32K	8K	4.5K	4K
CRS510-8XS-2XQ	98DX4310	7.3	60K - 120K	64K	15K - 20K	32K	8K	4.5K	4K
CRS518-16XS-2XQ	98DX8525	7.3	60K - 120K	64K	15K - 20K	32K	8K	4.5K	4K

[1] *Depends on the complexity of the routing table. Whole-byte IP prefixes (/8, /16, /24, etc.) occupy less HW space than others (e.g., /22). Starting with **RouterOS v7.3**, when the Routing HW table gets full, only routes with longer subnet prefixes are offloaded (/30, /29, /28, etc.) while the CPU processes the shorter prefixes. In RouterOS v7.2 and before, Routing HW memory overflow led to undefined behavior. Users can fine-tune what routes to offload via routing filters (for dynamic routes) or suppressing hardware offload of static routes. IPv4 and IPv6 routing tables share the same hardware memory.*

[2] *When the HW limit of Fasttrack or NAT entries is reached, other connections will fall back to the CPU. MikroTik's smart connection offload algorithm ensures that the connections with the most traffic are offloaded to the hardware.*

[3] *Fasttrack connections share the same HW memory with ACL rules. Depending on the complexity, one ACL rule may occupy the memory of 3-6 Fasttrack connections.*

[4] *MPLS shares the HW memory with Fasttrack connections. Moreover, enabling MPLS requires the allocation of the entire memory region, which could otherwise store up to 768 (0.75K) Fasttrack connections. The same applies to the Bridge Port Extender. However, MPLS and BPE may use the same memory region, so enabling them both doesn't double the limitation of Fasttrack connections.*

[5] *If a Fasttrack connection requires Network Address Translation, a hardware NAT entry is created. The hardware supports both SRCNAT and DSTNAT.*

[6] *The switch chip has a feature set of the DX8000 series.*

[7] *DX4000/DX8000 switch chips store directly connected hosts, IPv4 /32, and IPv6 /128 route entries in the FDB table rather than the routing table. The HW memory is shared between regular FDB L2 entries (MAC), IPv4, and IPv6 addresses. The number of hosts is also limited by max-neighbor-entries in IP Settings / IPv6 Settings.*

[8] *IPv4 and IPv6 routing tables share the same hardware memory.*

CCR2000

Model	Switch Chip	Release	IPv4 Routes	IPv4 Hosts	IPv6 Routes	IPv6 Hosts	Nexthops	Fasttrack connections	NAT entries
CCR2116-12G-4S+	**98DX3255** [1]	7.1	16K - 36K	16K	4K - 6K	8K	8K	2.25K	2.25K
CCR2216-1G-12XS-2XQ	98DX8525	7.1	60K - 120K	64K	15K - 20K	32K	8k	4.5K	4K

[1] *The switch chip has a feature set of the DX8000 series.*

MACsec

Overview

The MACsec (Media Access Control Security) protocol is a standard security technology employed in Ethernet networks to ensure the confidentiality, integrity, and authenticity of data transmitted over the physical medium. MACsec is defined by IEEE standard 802.1AE.

MACsec utilizes GCM-AES-128 encryption over Ethernet and secures all LAN traffic, including DHCP, ARP, LLDP, and higher-layer protocols.

> ⚠ RouterOS MACsec implementation is in the early stage, it **does not support** dynamic key management via Dot1x (manual key configuration is required) and hardware-accelerated encryption (maximum throughput is highly limited by the device CPU).

Basic Configuration Example

Imagine Host1 ether1 is connected to Switch ether1 and Host2 ether1 is connected to Switch ether2. In this example, we will create two MACsec interface pairs and use a bridge to create a secure Layer2 connection between both end devices.

First, configure MACsec interfaces on Host1 and Host2. We can specify only the Ethernet interface and RouterOS will automatically generate the Connectivity Association Key (CAK) and connectivity association name (CKN). Use the `print` command to see the values:

```
# Host1
/interface macsec
add interface=ether1 name=macsec1

[admin@Host2] /interface/macsec print
Flags: I - inactive, X - disabled, R - running
 0   name="macsec1" mtu=1468 interface=ether1 status="negotiating" cak=71a7c363794da400dbde595d3926b0e9
     ckn=f2c4660060169391d29d8db8a1f06e5d4b84a128bad06ad43ea2bd4f7d21968f profile=default

# Host2
/interface macsec
add interface=ether1 name=macsec1

[admin@Host2] /interface/macsec print
Flags: I - inactive, X - disabled, R - running
 0   name="macsec1" mtu=1468 interface=ether1 status="negotiating" cak=dc47d94291d19a6bb26a0c393a1af9a4
     ckn=e9bd0811dad1e56f06876aa7715de1855f1aee0baf5982ac8b508d4fc0f162d9 profile=default
```

On the Switch device, to enable MACsec we need to configure the matching CAK and CKN values for the appropriate Ethernet interface:

```
# Switch
/interface macsec
add comment=Host1 cak=71a7c363794da400dbde595d3926b0e9
ckn=f2c4660060169391d29d8db8a1f06e5d4b84a128bad06ad43ea2bd4f7d21968f interface=ether1 name=macsec1
add comment=Host2 cak=dc47d94291d19a6bb26a0c393a1af9a4
ckn=e9bd0811dad1e56f06876aa7715de1855f1aee0baf5982ac8b508d4fc0f162d9 interface=ether2 name=macsec2
```

Once the pre-shared keys are successfully exchanged, the MACsec Key Agreement (MKA) protocol is activated. MKA is responsible for ensuring the continuity of MACsec on the link and determines which side becomes the key server in a point-to-point connection. The key server generates a Secure Association Key (SAK) that is shared exclusively with the device on the other end of the link. This SAK is used to secure all data traffic passing through the link. Periodically, the key server generates a new randomly-created SAK and shares it over the point-to-point link to maintain MACsec functionality.

In RouterOS, the MACsec interface can be configured like any Ethernet interface. It can be used as a routable interface with an IP address, or placed inside a bridge. On Host1 and Host2 we will add an IP address from the same network. On Switch, we will use a bridge.

```
# Host1
/ip address
add address=192.168.10.10/24 interface=macsec1

# Host2
/ip address
add address=192.168.10.20/24 interface=macsec1

# Switch
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=macsec1
add bridge=bridge1 interface=macsec2
```

Last, confirm that Host1 can reach Host2 using a ping.

```
[admin@Host1] > ping 192.168.10.20
 SEQ HOST                                     SIZE TTL TIME       STATUS
   0 192.168.10.20                              56  64 1ms438us
   1 192.168.10.20                              56  64 818us
   2 192.168.10.20                              56  64 791us
   3 192.168.10.20                              56  64 817us
   4 192.168.10.20                              56  64 783us
   sent=5 received=5 packet-loss=0% min-rtt=783us avg-rtt=929us max-rtt=1ms438us
```

Property Reference

Interface settings

Sub-menu: `/interface/macsec`

Configuration settings for the MACsec interface.

Property	Description
cak (*string*; Default:)	A 16-byte pre-shared connectivity association key (CAK). To enable MACsec, configure the matching CAK and CKN on both ends of the link. When not specified, RouterOS will automatically generate a random value.
ckn (*string*; Default:)	A 32-byte connectivity association name (CKN). To enable MACsec, configure the matching CAK and CKN on both ends of the link. When not specified, RouterOS will automatically generate a random value.
comment (*string*; Default:)	Short description of the interface.
disabled (*yes* | *no*; Default: **no**)	Changes whether the interface is disabled.
interface (*name*; Default:)	Ethernet interface name where MACsec is created on, limited to one MACsec interface per Ethernet.
mtu (*integer*; Default: **1468**)	Sets the maximum transmission unit. The `l2mtu` will be set automatically according to the associated `interface` (subtracting 32 bytes corresponding to the MACsec encapsulation). The `l2mtu` cannot be changed.
name (*string*; Default: **macsec1**)	Name of the interface.
profile (*name*; Default: **default**)	Sets MACsec profile, used for determining the key server in a point-to-point connection.
status (*read-only: disabled |initializing | invalid | negotiating | open-encrypted*)	Shows the current MACsec interface status.

Profile settings

Sub-menu: `/interface/macsec/profile`

Configuration settings for the MACsec profile.

Property	Description
name (*string*; Default:)	Name of the profile.
server-priority (*integer: 0..255*; Default: **10**)	Sets the priority for determining the key server in a point-to-point connection, a lower value means higher priority. In case of a priority match, the interface with the lowest MAC address will be acting as a key server.

MACVLAN

Overview

The MACVLAN provides a means to create multiple virtual network interfaces, each with its own unique Media Access Control (MAC) address, attached to a physical network interface. This technology is utilized to address specific network requirements, such as obtaining multiple IP addresses or establishing distinct PPPoE client connections from a single physical Ethernet interface while using different MAC addresses. Unlike traditional VLAN (Virtual LAN) interfaces, which rely on Ethernet frames tagged with VLAN identifiers, MACVLAN operates at the MAC address level, making it a versatile and efficient solution for specific networking scenarios.

> RouterOS MACVLAN interfaces are not supported by Container, as it exclusively utilizes VETH (Virtual Ethernet) interfaces for its networking.

Basic Configuration Example

Picture a scenario where the ether1 interface connects to your ISP, and your router needs to lease two IP addresses, each with a distinct MAC address. Traditionally, this would require the use of two physical Ethernet interfaces and an additional switch. However, a more efficient solution is to create a virtual MACVLAN interface.

To create a MACVLAN interface, select the needed Ethernet interface. A MAC address will be automatically assigned if not manually specified:

```
/interface macvlan
add interface=ether1 name=macvlan1

/interface macvlan print
Flags: R - RUNNING
Columns: NAME, MTU, INTERFACE, MAC-ADDRESS, MODE
#   NAME       MTU  INTERFACE  MAC-ADDRESS        MODE
0 R macvlan1  1500  ether1     76:81:BF:68:69:83  bridge
```

Now, a DHCP client can be created on ether1 and macvlan1 interfaces:

```
/ip dhcp-client
add interface=ether1
add interface=macvlan1
```

Property Reference

Sub-menu: `/interface/macvlan`

Configuration settings for the MACVLAN interface.

Property	Description

arp (*disabled* | *enabled* | *local-proxy-arp* | *proxy-arp* | *reply-only*; Default: **enabled**)	Address Resolution Protocol setting • `disabled` - the interface will not use ARP • `enabled` - the interface will use ARP • `local-proxy-arp` - the router performs proxy ARP on the interface and sends replies to the same interface • `proxy-arp` - the router performs proxy ARP on the interface and sends replies to other interfaces • `reply-only` - the interface will only reply to requests originating from matching IP address/MAC address combinations, which are entered as static entries in the IP/ARP table. No dynamic entries will be automatically stored in the IP/ARP table. Therefore, for communications to be successful, a valid static entry must already exist.
arp-timeout (*auto* | *integer*; Default: **auto**)	Sets for how long the ARP record is kept in the ARP table after no packets are received from IP. Value `auto` equals to the value of `arp-timeout` in `/ip/settings/`, default is 30s.
comment (*string*; Default:)	Short description of the interface.
disabled (*yes* | *no*; Default: **no**)	Changes whether the interface is disabled.
interface (*name*; Default:)	Name of the interface on top of which MACVLAN will work. MACVLAN interfaces can be created on Ethernet or VLAN interfaces, adding VLAN on MACVLAN is not supported.
loop-protect (*on* | *off* | *default*; Default: **default**)	Enables or disables loop protect on the interface, the **default** works as turned off.
loop-protect-disable-time (*time interval* | *0*; Default: **5m**)	Sets how long the selected interface is disabled when a loop is detected. **0** - forever.
loop-protect-send-interval (*time interval*; Default: **5s**)	Sets how often loop protect packets are sent on the selected interface.
mac-address (*MAC*; Default:)	Static MAC address of the interface. A randomly generated MAC address will be assigned when not specified.
mode (*private* | *bridge*; Default: **bridge**)	Sets MACVLAN interface mode: • `private` - does not allow communication between MACVLAN instances on the same parent **interface**. • `bridge` - allows communication between MACVLAN instances on the same parent **interface**.
mtu (*integer*; Default: **1500**)	Sets Layer 3 Maximum Transmission Unit. For the MACVLAN interface, it cannot be higher than the parent **interface**.
name (*string*; Default:)	Interface name.

Quality of Service (QoS)

Overview

This document defines **Quality of Service (QoS)** usage in RouterOS based on **Marvell Prestera DX switch chips** (CRS3xx, CRS5xx series switches, and CCR2116, CCR2216 routers).

QoS is a set of features in network switches that allow network administrators to prioritize traffic and allocate network resources to ensure that important data flows smoothly and with low latency.

The primary function of QoS in network switches is to manage network traffic in a way that meets the specific requirements of different types of network applications. For example, voice and video data require low latency and minimal packet loss to ensure high-quality communication, while file transfers and other data applications can tolerate higher levels of latency and packet loss.

QoS works by identifying the type of traffic flowing through the switch and assigning it a priority level based on its requirements. The switch can then use this information to alter packet headers and prioritize the flow of traffic, ensuring that higher-priority traffic is given preferential treatment over lower-priority traffic.

Planned QoS implementation phases:

1. **QoS Marking.** QoS profile matching by ingress packet headers, then egress header alternation according to the assigned QoS profiles (introduced in RouterOS v7.10).
2. **QoS Enforcement**. Avoid or resolve congestion based on the assigned QoS profiles and traffic shaping (introduced in RouterOS v7.13 for 98DX224S, 98DX226S, and 98DX3236 switch chips, and extended to all capable switch chips starting from RouterOS v7.15).
3. **QoS Policy**. Assign QoS profiles via ACL rules (introduced in RouterOS v7.15).
4. Extra QoS Features: **WRED** (Weighted Random Early Detection), **ECN** notification, and processing (introduced in RouterOS v7.15 to capable switch chips).
5. Traffic shaping (introduced per-queue traffic shaping in RouterOS v7.15).

QoS Terminology

These terms will be used throughout the article.

- **QoS** - Quality of Service.
- **ACL** - Access Control List, a set of switch rules used to filter network traffic based on specified criteria.
- **AQM** - Active Queue Management.
- **DSCP** - Differentiated Services Code Point, a 6-bit field in the IP header used to prioritize network traffic.
- **ECN** - Explicit Congestion Notification.
- **PCP** - Priority Code Point, a 3-bit field in the VLAN header used to prioritize traffic within a VLAN.
- **WRED** - Weighted Random Early Detection.
- **`/in/eth/sw/`** a shortcut for **`/interface/ethernet/switch/`**. The shortcut works in CLI, too.

QoS Device Support

Model	Switch Chip	QoS Profiles	QoS Maps	Tx Managers	WRED	ECN	Port/Queue Usage Stats
CCR2116-12G-4S+	98DX3255	1024	12	15	✓	✓	Unreliable [1]
CCR2216-1G-12XS-2XQ	98DX8525	1024	12	15	✓	✓	Max fill [2]
CRS305-1G-4S+	98DX3236	128	1	8			Current values
CRS309-1G-8S+	98DX8208	1024	12	15	✓	✓	Unreliable
CRS310-1G-5S-4S+	98DX226S	128	1	8			Current values
CRS312-4C+8XG	98DX8212	1024	12	15	✓	✓	Unreliable
CRS317-1G-16S+	98DX8216	1024	12	15	✓	✓	Unreliable
CRS318-1Fi-15Fr-2S	98DX224S	128	1	8			Current values
CRS318-16P-2S+	98DX226S	128	1	8			Current values
CRS326-24G-2S+	98DX3236	128	1	8			Current values
CRS326-24S+2Q+	98DX8332	1024	12	15	✓	✓	Unreliable
CRS328-24P-4S+	98DX3236	128	1	8			Current values
CRS328-4C-20S-4S+	98DX3236	128	1	8			Current values
CRS354-48G-4S+2Q+	98DX3257	1024	12	15	✓	✓	Unreliable
CRS504-4XQ	98DX4310	1024	12	15	✓	✓	Max fill
CRS510-8XS-2XQ	98DX4310	1024	12	15	✓	✓	Max fill
CRS518-16XS-2XQ	98DX8525	1024	12	15	✓	✓	Max fill

[1] Due to hardware limitations, some switch chip models may break traffic flow while accessing QoS port/queue usage data.

*[2] The device gathers max queue fill statistics instead of displaying the current usage values. Use the **reset-counters** command to reset those stats.*

Applications and Usage Examples

Basic Configuration Example

In this example, we define just one QoS level - VoIP (IP Telephony) on top of the standard "Best Effort" class. Let's imagine that we have a CRS326-24G-2S+ device where:

- all ports are bridged and using vlan-filtering;
- sfp-sfpplus1 is a VLAN trunk connected to another switch;
- ether1-ether9 are dedicated ports for IP phones;
- ether10-ether24 are standard ports for host connection;

First, we need to define QoS profiles. Defined `dscp` and `pcp` values will be used in forwarded packets on egress:

```
/interface ethernet switch qos profile
add dscp=46 name=voip pcp=5 traffic-class=5
```

Port-based QoS profile assignment on dedicated ports for IP phones applies to ingress traffic. Other Ethernet ports will use the default `qos-profile` (where `dscp=0` and `pcp=0`):

```
/interface ethernet switch qos port
set ether1 profile=voip
set ether2 profile=voip
set ether3 profile=voip
set ether4 profile=voip
set ether5 profile=voip
set ether6 profile=voip
set ether7 profile=voip
set ether8 profile=voip
set ether9 profile=voip
```

Starting from **RouterOS v7.13**, QoS port settings moved from `/interface/ethernet/switch/port` to **`/interface/ethernet/switch/qos/port`**. The "qos-" prefix from the respective fields have been removed (since all fields are qos-related anyway).

The trunk port receives both types of QoS traffic. We need to create VLAN priority mapping with the QoS profile and enable `qos-trust-l2` to differentiate them:

```
/interface ethernet switch qos map vlan
add pcp=5 profile=voip

/interface ethernet switch port
set sfp-sfpplus1 trust-l2=trust
```

Finally, enable QoS hardware offloading for the above settings to start working:

```
/interface ethernet switch
set switch1 qos-hw-offloading=yes
```

It is possible to verify the port QoS settings with `print` command:

```
[admin@MikroTik] /interface/ethernet/switch/qos/port print
Columns: NAME, SWITCH, PROFILE, MAP, TRUST-L2, TRUST-L3
 # NAME          SWITCH   PROFILE  MAP      TRUST-L2  TRUST-L3  TX-MANAGER
 0 ether1        switch1  voip     default  ignore    ignore    default
 1 ether2        switch1  voip     default  ignore    ignore    default
 2 ether3        switch1  voip     default  ignore    ignore    default
 3 ether4        switch1  voip     default  ignore    ignore    default
 4 ether5        switch1  voip     default  ignore    ignore    default
 5 ether6        switch1  voip     default  ignore    ignore    default
 6 ether7        switch1  voip     default  ignore    ignore    default
 7 ether8        switch1  voip     default  ignore    ignore    default
 8 ether9        switch1  voip     default  ignore    ignore    default
 9 ether10       switch1  default  default  ignore    ignore    default
10 ether11       switch1  default  default  ignore    ignore    default
11 ether12       switch1  default  default  ignore    ignore    default
12 ether13       switch1  default  default  ignore    ignore    default
13 ether14       switch1  default  default  ignore    ignore    default
14 ether15       switch1  default  default  ignore    ignore    default
15 ether16       switch1  default  default  ignore    ignore    default
16 ether17       switch1  default  default  ignore    ignore    default
17 ether18       switch1  default  default  ignore    ignore    default
18 ether19       switch1  default  default  ignore    ignore    default
19 ether20       switch1  default  default  ignore    ignore    default
20 ether21       switch1  default  default  ignore    ignore    default
21 ether22       switch1  default  default  ignore    ignore    default
22 ether23       switch1  default  default  ignore    ignore    default
23 ether24       switch1  default  default  ignore    ignore    default
24 sfp-sfpplus1  switch1  default  default  trust     ignore    default
25 sfp-sfpplus2  switch1  default  default  ignore    ignore    default
26 switch1-cpu   switch1
```

Now incoming packets on ports ether1-ether9 are marked with a Priority Code Point (PCP) value of 5 and a Differentiated Services Code Point (DSCP) value of 46, and incoming packets on ports ether10-ether24 are marked with PCP and DSCP values of 0. When packets are incoming to sfp-sfpplus1 port, any packets with a PCP value of 5 or higher will retain their PCP value of 5 and DSCP value of 46, while all other packets will be marked with PCP and DSCP values of 0.

Dante

Starting from RouterOS v7.15, all MikroTik QoS-Capable devices comply with Dante**.**

Dante hardware use the following DSCP / Diffserv priority values for traffic prioritization.

Dante Priority	Usage	DSCP Label	DSCP Value
High	Time critical PTP events	CS7	56
Medium	Audio, PTP	EF	46
Low	(reserved)	CS1	8
None	Other traffic	BE	0

The example assumes that the switch is using its default configuration, which includes a default "bridge" interface and all Ethernet interfaces added as bridge ports, and any of these interfaces could be used for Dante.

First, create QoS Profiles to match Dante traffic classes, there is already a pre-existing "default" profile that corresponds to Dante's None priority.

```
/interface/ethernet/switch/qos/profile
add name=dante-ptp dscp=56 pcp=7 traffic-class=7
add name=dante-audio dscp=46 pcp=5 traffic-class=5
add name=dante-low dscp=8 pcp=1 traffic-class=0
```

Then, create a QoS mapping to match QoS profiles based on DSCP values.

```
/interface/ethernet/switch/qos/map/ip
add dscp=56 profile=dante-ptp
add dscp=46 profile=dante-audio
add dscp=8 profile=dante-low
```

Configure hardware queues to enforce QoS on Dante traffic.

```
/interface/ethernet/switch/qos/tx-manager/queue
set [find where traffic-class>=2] schedule=strict-priority
set [find where traffic-class<2] schedule=low-priority-group weight=1
```

Dante's High and Medium priority traffic is scheduled in strict order. The devices transmits time-critical PTP packets until queue7 gets empty, then proceed with audio (queue5). Low and other traffic gets transmitted only when PTP and audio queues are empty. Since Dante does not define priority order between Low and Other traffic (usually, CS1 has lower priority than Best Effort), and the Low traffic class is reserved for future use anyway, we treat both traffic types equally by putting both into the same group with the same weight. Feel free to change the CS1/BE traffic scheduling according to the requirements if some Dante hardware in your network uses the low-priority traffic class.

The next step is to enable trust mode for incoming Layer3 packets (IP DSCP field):

```
/interface/ethernet/switch/qos/port
set [find] trust-l3=keep
```

Finally, enable QoS hardware offloading for the above settings to start working:

```
/interface ethernet switch
set switch1 qos-hw-offloading=yes
```

When using Dante in multicast mode, it is beneficial to enable IGMP snooping on the switch. This feature directs traffic only to ports with subscribed devices, preventing unnecessary flooding. Additionally, enabling an IGMP querier (if not already enabled on another device in the same LAN), adjusting query intervals, and activating fast-leave can further optimize multicast performance.

```
/interface/bridge
set [find name=bridge] igmp-snooping=yes multicast-querier=yes query-interval=60s

/interface/bridge/port
set [find] fast-leave=yes
```

QoS Marking

Understanding Map ranges

In order to avoid defining all possible PCP and DSCP mappings, RouterOS allows setting the *minimal* PCP and DSCP values for QoS Profile mapping.

In the following example, PCP values 0-2 use the default QoS profile, 3-4 - streaming, 5 - voip, and 6-7 - control.

```
/interface ethernet switch qos map vlan
add pcp=3 profile=streaming
add pcp=5 profile=voip
add pcp=6 profile=control
```

Since the `pcp` the parameter identifies the *minimum* value, all packets with a higher PCP value match too. If such behavior is undesired, add mapping for higher values. The next example sets voip profile for pcp=5 only. Packets with PCP values 6 or 7 are reset back to the default profile.

```
/interface ethernet switch qos map vlan
add pcp=5 profile=voip
add pcp=6 profile=default
```

Understanding Port, Profile, and Map relation

Each switch port has Layer2 and Layer3 trust settings that will change how ingress packets are classified into QoS profiles and what PCP and DSCP values will be used. Below are tables that describe all possible options:

qos-trust-l2	qos-trust-l3	Behavior
ignore	**ignore**	The port is considered untrusted. Both headers are ignored, and the port's **profile** is forced to all ingress packets. This is the default setting.
ignore	**trust**	Trust the Layer 3 header. Use the DSCP field from the IP header of ingress packets for QoS profile lookup (see `/in/eth/sw/qos/map/ip`). If the lookup fails (no QoS profiles are mapped to the given DSCP value), the **default** QoS profile is used (not the switch port's QoS profile). The switch port's **profile** field is used only for non-IP traffic.
ignore	**keep**	Trust the Layer 3 header. Use the DSCP field from the IP header of ingress packets for QoS profile lookup (see `/in/eth/sw/qos/map/ip`). If the lookup fails, the **default** QoS profile is used. The switch port's **profile** field is used only for non-IP traffic. If the forwarded/routed packet is VLAN-tagged, its PCP value is set from the selected QoS profile. However, the original DSCP value of the packet is kept intact.
trust	**ignore**	Trust the Layer 2 header, but ignore L3. If an ingress packet is VLAN-tagged, use the PCP field from the VLAN header for QoS profile lookup (see `/in/eth/sw/qos/map/vlan`). If the lookup fails (no QoS profiles are mapped to the given PCP value), the **default** QoS profile is used. The switch port's **profile** field is used only for untagged traffic.
trust	**trust**	Trust both headers, but Layer 3 has higher precedence. In the case of an IP packet, use the DSCP field for QoS profile lookup (see `/in/eth/sw/qos/map/ip`). If the DSCP-to-QoS lookup fails, use the **default** profile. If the packet is not an IP packet but is VLAN-tagged, use the PCP field from the VLAN header for QoS profile lookup (see `/in/eth/sw/qos/map/vlan`). If the VLAN-to-QoS lookup fails, use the **default** QoS profile. Non-IP untagged packets use the switch port's **profile**.
trust	**keep**	The same as **trust+trust**, but the original DSCP value is preserved in forwarded/routed packets.
keep	**ignore**	Trust the Layer 2 header but ignore L3. If an ingress packet is VLAN-tagged, use the PCP field from the VLAN header for QoS profile lookup (see `/in/eth/sw/qos/map/vlan`). If the lookup fails (no QoS profiles are mapped to the given PCP value), the **default** QoS profile is used. The switch port's **profile** field is used only for untagged traffic. If the packet is VLAN-tagged on both ingress and egress, the original PCP value is kept.
keep	**trust**	Trust both headers, but Layer 3 has higher precedence. In the case of an IP packet, use the DSCP field for QoS profile lookup (see `/in/eth/sw/qos/map/ip`). If the DSCP-to-QoS lookup fails, use the **default** profile. If the packet is not an IP packet but is VLAN-tagged, use the PCP field from the VLAN header for QoS profile lookup (see `/in/eth/sw/qos/map/vlan`). If the VLAN-to-QoS lookup fails, use the **default** QoS profile. Non-IP untagged packets use the switch port's **profile**. If the packet is VLAN-tagged on both ingress and egress, the original PCP value is kept. The DSCP value in forwarded/routed packets is set from the selected QoS profile.
keep	**keep**	Trust both headers, but Layer 3 has higher precedence. In the case of an IP packet, use the DSCP field for QoS profile lookup (see `/in/eth/sw/qos/map/ip`). If the DSCP-to-QoS lookup fails, use the **default** profile. If the packet is not an IP packet but is VLAN-tagged, use the PCP field from the VLAN header for QoS profile lookup (see `/in/eth/sw/qos/map/vlan`). If the VLAN-to-QoS lookup fails, use the **default** QoS profile. Non-IP untagged packets use the switch port's **profile**. Keep both the original PCP and/or DSCP values intact in cases of VLAN-tagged and/or IP packets, respectively.

Port settings		The selected QoS profile and the source for PCP / DSCP field values in forwarded/routed packets											
qos-trust-l2	qos-trust-l3	VLAN-Tagged IP			Untagged IP			VLAN-Tagged Non-IP			Untagged Non-IP		
		QoS Profile	PCP	DSCP	QoS Profile	PCP [1]	DSCP	QoS Profile	PCP	DSCP	QoS Profile	PCP [1]	DSCP
ignore	ignore	profile	profile	profile	profile	profile	profile	profile	profile	-	profile	profile	-
ignore	trust	map/ip	map/ip	map/ip	map/ip	map/ip	map/ip	profile	profile	-	profile	profile	-
ignore	keep	map/ip	map/ip	original	map/ip	map/ip	original	profile	profile	-	profile	profile	-
trust	ignore	map/vlan	map/vlan	map/vlan	profile	profile	profile	map/vlan	map/vlan	-	profile	profile	-
trust	trust	map/ip	map/ip	map/ip	map/ip	map/ip	map/ip	map/vlan	map/vlan	-	profile	profile	-

trust	keep	map/ip	map/ip	original	map/ip	map/ip	original	map/vlan	map/vlan	-	profile	profile	-
keep	ignore	map/vlan	original	map/vlan	profile	profile	profile	map/vlan	original	-	profile	profile	-
keep	trust	map/ip	original	map/ip	map/ip	profile	map/ip	map/vlan	original	-	profile	profile	-
keep	keep	map/ip	original	original	map/ip	profile	original	map/vlan	original	-	profile	profile	-

[1] *applies only when ingress traffic is untagged, but the egress needs to be VLAN-tagged.*

QoS Marking via Switch Rules (ACL)

Starting from **RouterOS v7.15**, it is possible to assign QoS profiles via Switch Rules (ACL).

Sub-menu: `/interface/ethernet/switch/rule`

New/Changed Properties	Description
new-qos-profile (*name*)	The name of the QoS profile to assign to the matched packets.
keep-qos-fields (*yes* | *no*; Default: **no**)	Should the original values of QoS fields (PCP, DSCP) be kept (*yes*), or replace them with the ones from the assigned QoS profile (*no*)? Relevant only if **new-qos-profile** is set.
new-vlan-priority (*0..7*)	Deprecated and should be replaced with the respective **new-qos-profile**. Kept for backward compatibility. Relevant only if qos-hw-offloading=no.

The following example assigns a QoS profile based on the source MAC address.

```
/interface ethernet switch rule
add new-qos-profile=stream ports=ether1,ether2 src-mac-address=00:01:02:00:00:00/FF:FF:FF:00:00:00
switch=switch1
add new-qos-profile=voip ports=ether1,ether2 src-mac-address=04:05:06:00:00:00/FF:FF:FF:00:00:00 switch=switch1
```

QoS Enforcement

Hardware Queues

Each switch port has eight hardware transmission (tx) queues (queue0..queue7). Each queue corresponds to a traffic class (tc0..tc7) set by a QoS profile. Each ingress packet gets assigned to a QoS profile, which, in turn, determines the traffic class for tx queue selection on the egress port.

Hardware queues are of variable size - set by the Transmission Manager. Moreover, multiple ports and/or queues can share resources with each other (so-called *Shared Buffers*). For example, a device with 25 ports has memory (buffers) to queue 1200 packets in total. If we split the resources equally, each port gets 48 exclusive buffers with a maximum of 6 packets per queue (48/8) - which is usually insufficient to absorb even a short burst of traffic. However, choosing to share 50% of the buffers leaves each port with 24 exclusive buffers (3 per queue), but at the same time, a single queue can grow up to 603 buffers (3 exclusive + 600 shared).

RouterOS allows enabling/disabling the shared pool for each queue individually - for example, to prevent low-priority traffic from consuming the entire hardware memory. In addition, port buffer limits may prevent a single low-speed port from consuming the entire shared pool. See QoS Settings and Transmission Manager for details.

> The default, best-effort (PCP=0, DSCP=0) traffic class is 1, while the lowest priority (PCP=1) has traffic class 0.

Hardware Resources

The hardware (switch chips) has limited resources (memory). There are two main hardware resources that are relevant to QoS:

- *Packet descriptors* - contain packet control information (target port, header alternation, etc).
- *Data buffers* - memory chunks containing the actual payload. Buffer size depends on the switch chip model. Usually - 256 bytes.

One packet descriptor may use multiple buffers (depending on the payload size); buffers may be shared by multiple descriptors - in cases of multicast /broadcast. If the hardware does not have enough free descriptors or buffers, the packet gets dropped (*tail-drop*).

Hardware resources can be limited per destination type (multicast/unicast), per port, and per each tx queue. If any limits are reached, no more packets can be enqueued for transmission, and further packets get dropped.

RouterOS obscures low-level hardware information, allowing to set resource limits either in terms of packets or a percentage of the total amount. RouterOS automatically calculates the required hardware descriptor and buffer count based on the user-specified packet limit and port's MTU. Moreover, RouterOS comes with preconfigured hardware resources, so there is no need to do a manual configuration in common QoS environments.

> Changing any hardware resource allocation parameter in runtime results in a temporary device halt when no packets can be enqueued nor transmitted. Temporary packet loss is expected while the device is forwarding traffic.

Resource Saving

Since reallocating hardware resources in runtime is not an option, RouterOS cannot automatically free queue buffers reserved for inactive ports. Those buffers remain unused. However, if the user knows that the specific ports will *never* be used (e.g., stay physically disconnected), the respective queue resources can be manually freed by introducing the "offline" tx-manager with minimum resources:

```
/interface/ethernet/switch/qos/tx-manager/
add name=offline comment="use this for always disconnected ports"

/interface/ethernet/switch/qos/tx-manager/queue/
set [find where tx-manager=offline] queue-buffers=1 use-shared-buffers=no

/interface/ethernet/switch/qos/port
set [find where !running] tx-manager=offline
```

Traffic Prioritization

The hardware provides two types of traffic transmission prioritization:

- **Strict Priority** - traffic from higher queues is always transmitted first;
- **Weighted Priority Groups** - multiple queues participate in packet transmission scheduling at the same time.

Strict priority queues are straightforward. If the highest priority queue (Q7) has packets, those are transmitted first. When Q7 is empty, packets from Q6 get transmitted, and so on. The packets from the lowest priority queue (Q0) are transmitted only if all other queues are empty.

The downside of strict prioritization is increased latency in lower queues while "overprioritizing" higher queues. Suppose the acceptable latency of TC5 is 20ms, TC3 - 50ms. Traffic appearing in Q5 gets immediately transmitted due to the strict priority of the queue, adding extra latency to *every* packet in the lower queues (Q4..Q0). A packet burst in Q5 (e.g., a start of a voice call) may temporarily "paralyze" Q3, increasing TC3 latencies over the acceptable 50ms (or even causing packet drops due to full queue) while TC5 packets get transmitted at <1ms (way below the 20ms limit). Slightly sacrificing TC5 latency by transmitting TC3 packets in between would make everybody happy. That *Weighted Priority Groups* are for.

Weighted Priority Groups schedule traffic for transmission from multiple queues (group members) in a weighted round-robin manner. A queue's weight sets the number of packets transmitted from the queue in each round. For example, if Q2, Q1, and Q0 are the group members, and their weights are 3, 2, and 1, respectively, the scheduler transmits 3 packets from Q2, 2 - from Q1, and 1 - from Q0. The actual Tx order is "Q2, Q1, Q0, Q2, Q1, Q2" - for even fairer scheduling.

There are two hardware groups: `low-priority-group` and `high-priority-group`. There is a strict priority ordering between the two groups: the low-priority-group is transmitting only when *all* queues in the high-priority-group are empty. However, it is possible to use only one group for all queues.

The default (built-in) RouterOS queue setup is listed below. Q3-Q5 share the bandwidth within the high-priority group, where packets are transmitted while Q6 and Q7 are empty. Q0-Q2 are the members of the low-priority-group, where packets are transmitted while Q3-Q7 are empty.

```
[admin@MikroTik] /interface/ethernet/switch/qos/tx-manager/queue> print
Columns: TX-MANAGER, TRAFFIC-CLASS, SCHEDULE, WEIGHT, QUEUE-BUFFERS, USE-SHARED-BUFFERS
#  TX-MANAGER  TRAFFIC-CLASS  SCHEDULE             WEIGHT  QUEUE-BUFFERS  USE-SHARED-BUFFERS
0  default     0              low-priority-group   1       auto           no
1  default     1              low-priority-group   2       auto           yes
2  default     2              low-priority-group   3       auto           yes
3  default     3              high-priority-group  3       auto           yes
4  default     4              high-priority-group  4       auto           yes
5  default     5              high-priority-group  5       auto           yes
6  default     6              strict-priority              auto           yes
7  default     7              strict-priority              auto           yes
```

It is recommended that all group members are adjacent to each other.

Active Queue Management (AQM)

Weighted Random Early Detection (WRED)

WRED is a per-queue congestion control mechanism that signals congestion events to the end-points by dropping packets. WRED relies on the existence of rate throttling mechanisms in the end-points that react to packet loss, such as TCP/IP. WRED uses a randomized packet drop algorithm in an attempt to anticipate congestion events and respond to them by throttling traffic rates before the congestion actually happens. The randomness property of WRED prevents throughput collapse related to the global synchronization of TCP flows.

WRED can be enabled/disabled per each queue in each Tx Manager. Disable WRED for lossless traffic! Also, there is no reason to enable WRED on high-speed ports where congestion should not happen in the first place.

The behavior is controlled via *WRED margins*. WRED margin is the distance to the queue/pool buffer limit (cap) - where a random packet drop begins. A different margin can be applied to queues that use or don't use shared buffers. A queue that uses a shared pool may set a bigger WRED margin due to a higher overall cap (queue buffers + shared pool). RouterOS automatically chooses the actual WRED margin values according to queue or shared pool capacities. The user may shift the margins in one way or another via QoS Settings.

For example, if **queue1-packet-cap=96**, and WRED margin is 32 (assuming **use-shared-buffers=no**), then:

- first 64 packets are always enqueued (96 - 32 = 64).
- WRED kicks in starting from 65th packet; the chance of 65th packet to be dropped is 1/32 or roughly 3%; the formula: (65 - 64) / 32;
- the drop chance of 72th packet is 25%: (72 - 64) / 32 = 8/32 = 1/4;
- half of the newly enquining packets are dropped when the queue fill level reaches 80 packets: (80 - 64) / 32 = 16/32 = 1/2;
- 75% of the packets are dropped at the fill level 88: (88 - 64) / 32 = 24/32 = 3/4.

When WRED is enabled (**wred=yes**), the queue cap is rounded to the applied WRED margin. In the above example, setting **wred-queue-margin=64** raises **queue1-packet-cap** to 128. That, in turn, may lower the resources available to other queues, such as shared buffers. A much safer option is to raise **wred-shared-queue-margin** which may reduce the shared buffers available for the affected queue but not the shared pool itself. For example, if: "**wred-shared-queue-margin=256, use-shared-buffers=yes, wred=yes, shared-pool-index=0, queue2-packet-cap=30,** and **shared-pool0-packet-cap=900"**, queue2 can use up to 768 buffers (30+900=930, rounded down by the scale of 256), and WRED starting at 512, while the other ports/queues still may use the remaining 162 buffers (30+900-768) of the shared pool.

Choosing a WRED margin value is a tradeoff between congestion anticipation and burst absorption. Setting a higher WRED margin may lead to earlier traffic rate throttling and, therefore, resolve congestion. On the other hand, a high margin leads to packet drops in limited traffic bursts that could be absorbed by the queue buffers and transformed losslessly if WRED didn't kick in. For instance, initiating a remote database connection usually starts with heavier traffic ("packet burst") at the initialization phase; then, the traffic rate drops down to a "reasonable" level. Any packet drop during the initialization phase leads to nothing but a slower database connection due to the need for retransmission. Hence, lowering the WRED margin or entirely disabling WRED on such traffic is advised. The opposite case is video streaming. Early congestion detection helps select a comfortable streaming rate without losing too much bandwidth on retransmission or/and "overshooting" by sacrificing the quality level by too much.

Use Switch Rules (ACL) or other QoS Marking techniques to differentiate traffic and put packets into queues with desired WRED settings.

The following script only applies WRED to TCP/IP traffic by redirecting it to queue2. UDP and other packets are left in queue1 - since their end-points usually cannot respond to early drops. Queue1 and queue2 are scheduled equally - without prioritizing one queue over another.

```
/interface/ethernet/switch/qos/profile
add name=tcp-wred traffic-class=2 pcp=0 dscp=0

# move TCP traffic to queue2
/interface/ethernet/switch/rule
add new-qos-profile=tcp-wred ports=ether1,ether2,ether3,ether4 protocol=tcp switch=switch1

# set the same scheduling priority (weight) between queue1 and queue2
# apply WRED only to queue2 - TCP traffic
/interface/ethernet/switch/qos/tx-manager/queue/
set [find where traffic-class=1] weight=2 schedule=low-priority-group use-shared-buffers=yes shared-pool-
index=0 wred=no
set [find where traffic-class=2] weight=2 schedule=low-priority-group use-shared-buffers=yes shared-pool-
index=0 wred=yes
```

Explicit Congestion Notification (ECN)

Some switch chips can perform ECN marking of IP packets on the hardware level, according to RFC 3168. Hardware ECN marking is based on the WRED mechanism, but instead of dropping IP packets, they are marked with CE (Congestion Experienced, binary 11) in the ECN field (two least significant bits in IPv4/TOS or IPv6/TrafficClass octet). Only ECN-Capable IP packets may be marked - those with the ECN field value of ECT(1) or ECT(0) (binary 01 or 10, respectively). Not ECN-Capable Transport packets (ECN=00) never get marked. If a packet already has the CE mark (ECN=11), it never gets cleared, even if the device does not experience congestion.

Set **ecn=yes** in Tx Manager to enable ECN marking. The per-queue ECN setting is unavailable due to hardware limitations. ECN and WRED share the same queue fill threshold: **wred-shared-queue-margin** (see QoS Settings).

ECN marking mechanism requires the respective Tx queues to use shared buffers (**use-shared-buffers=yes**).

The packet receives the CE mark if all conditions below are met:

1. The packet is either IPv4 or IPv6.
2. The ECN field value in IP header is either ECT(1) or ECT(0).
3. Egress port's Tx Manager has **ecn=yes**.
4. The assigned Tx Queue uses shared buffers (**use-shared-buffers=yes**).
5. The Tx Queue detects congestion via WRED margins.

Property Reference

Switch settings

Sub-menu: `/interface/ethernet/switch`

Switch QoS settings (in addition to the existing ones).

Property	Description
qos-hw-offloading (*yes* / *no*; Default: **no**)	Allows enabling QoS for the given switch chip (if the latter supports QoS).

When you enable QoS, turning off the **qos-hw-offloading** setting will not completely revert to the previous functionality. It is recommended to reboot the device after disabling it.

Port settings

Sub-menu: `/interface/ethernet/switch/qos/port`

Starting from **RouterOS v7.13**, QoS port settings moved from `/interface/ethernet/switch/port` to **`/interface/ethernet/switch/qos/port`**. The "qos-" prefix from the respective fields has been removed (since all fields are qos-related anyway).

Switch port QoS settings. Assigns a QoS profile to ingress packets on the given port. The assigned profile can be changed via match rules if the port is considered trusted.

By default, ports are untrusted and receive the default QoS profile (Best-Effort, PCP=0, DSCP=0), where priority fields are cleared from the egress packets.

Property	Description
egress-rate-queue0 .. egress-rate-queue7 (*integer: 0..18446744073709551615*; Default **!egress-rate-queuex**)	Sets egress traffic limitation (bits per second) for specific output queue. It is possible to specify the limit using suffixes like k, M, or G to represent kbps, Mbps, or Gbps. This setting can be combined with the overall per-port limit **egress-rate** (see **`/in/eth/sw/port`**).
map (*name*; Default: **default**)	Allows user-defined QoS priority-to-profile mapping in the case of a trusted port or host (see **`/in/eth/sw/qos/map`**).
profile (*name*; Default: **default**)	The name of the QoS profile to assign to the ingress packets by default (see **`/in/eth/sw/qos/profile`**).
trust-l2 (*ignore* | *trust* | *keep*; Default: **ignore**)	Whenever to trust the Layer 2 headers of the incoming packets (802.1p PCP field): • **ignore** - ignore L2 header; use the port's **profile** value for all incoming packets; • **trust** - use PCP field of VLAN-tagged packets for QoS profile lookup in **map**. Untagged packets use the port's **profile** value. Forwarded VLAN or priority-tagged packets receive the PCP value from the selected QoS profile (overwriting the original value). • **keep** - trust but keep the original PCP value in forwarded packets.
trust-l3 (*ignore* | *trust* | *keep*; Default: **ignore**)	Whenever to trust the Layer 3 headers of the incoming packets (IP DSCP field): • **ignore** - ignore L3 header; use either L2 header or the port's **profile** (depends on **trust-l2**). • **trust** - use DSCP field of IP packets for QoS profile lookup in **map**. Forwarded/routed IP packets receive the DSCP value from the selected QoS profile (overwriting the original value). • **keep** - trust but keep the original DSCP value in forwarded/routed packets.
tx-manager (*name*; Default: **default**)	The name of the Transmission Manager that is responsible for enqueuing and transmitting packets *from* the given port (see **`/in/eth/sw/qos/tx-manager`**).

L3 trust mode has higher precedence than L2 unless *trust-l3=ignore* or the packet does not have an IP header.

Forwarded/routed packets obtain priority field values (PCP, DSCP) from the selected QoS profile, overwriting the original values unless the respective trust mode is set to **keep**.

Commands.

Command	Description
print	Print the above properties in a human-friendly format.
print stats	Print port statistics: total and per-queue transmitted/dropped packets/bytes.
reset-counters	Reset all counters in port statistics to zero.
print usage	Print queue usage/resources.

Port Stats

Example

```
[admin@Mikrotik] /interface/ethernet/switch/qos/port> print stats where name=ether2
                  name:     ether2
             tx-packet:      2 887
               tx-byte:  3 938 897
          drop-packet:      1 799
            drop-byte:  2 526 144
      tx-queue0-packet:         50
      tx-queue1-packet:      1 871
      tx-queue3-packet:        774
      tx-queue5-packet:        192
        tx-queue0-byte:      3 924
        tx-queue1-byte:  2 468 585
        tx-queue3-byte:  1 174 932
        tx-queue5-byte:    291 456
    drop-queue1-packet:      1 799
      drop-queue1-byte:  2 526 144
```

Property	Description
name	Port name.
tx-packet	The total number of packets transmitted via this port.
tx-byte	The total number of bytes transmitted via this port.
drop-packet	The total number of packets should have been transmitted via this port but were dropped due to a lack of resources (e. g., queue buffers) or QoS Enforcement.
drop-byte	The total number of bytes should have been transmitted via this port but were dropped.
tx-queue0-packet .. **tx-queue7-packet**	The number of packets transmitted via this port from the respective queue.
tx-queue0-byte .. **tx-queue7-byte**	The number of bytes transmitted via this port from the respective queue.
drop-queue0-packet .. **drop-queue7-packet**	The number of packets dropped from the respective queue (or not enqueued at all due to lack of resources).
drop-queue0-byte .. **drop-queue7-byte**	The number of bytes dropped from the respective queue.

Port Resources/Usage

Due to hardware limitations, some switch chip models may break traffic flow while accessing QoS port `usage` data. Use port `usage` for diagnostics/troubleshooting only. For monitoring, use QoS `monitor` or Port `stats` instead.

Example

```
[admin@crs326] /interface/ethernet/switch/qos/port> print usage where name=ether2
                name:  ether2
          packet-cap:     136
          packet-use:       5
            byte-cap:  35 840
            byte-use:   9 472
    queue0-packet-cap:     130
    queue0-packet-use:       1
    queue1-packet-cap:       5
    queue1-packet-use:       4
    queue3-packet-cap:      65
    queue3-packet-use:       2
      queue0-byte-cap:  24 576
      queue0-byte-use:     256
      queue1-byte-cap:   7 680
      queue1-byte-use:   6 144
      queue3-byte-cap:  14 080
      queue3-byte-use:   3 072
```

Property	Description
name	Port name.
packet-cap	Port's packet capacity. The maximum number of packets that can be enqueued for transmission via the port.
packet-use [1]	Port's packet usage. The number of packets that are currently enqueued in all port's queues.
byte-cap	Port's byte capacity (buffer size). The maximum number of bytes that can be enqueued for transmission via the port.
byte-use [1]	Port's byte usage. The size of hardware buffers (in bytes) that are currently allocated for packets the enqueued packets. Since the buffers are allocated by blocks (usually - 256B each), the allocated buffer size can be bigger than the actual payload.
queue0-packet-cap .. **queue7-packet-cap** [2]	Queue capacity (in packets). The maximum number of packets that can be enqueued in the respective queues.
queue0-packet-use .. **queue7-packet-use** [2]	Queue packet usage. The number of enqueued packets in the respective queues.
queue0-byte-cap .. **queue7-byte-cap** [2]	Queue buffer capacity (in bytes). The maximum number of bytes that can be enqueued in the respective queues. Only the queues in use are printed.
queue0-byte-use .. **queue7-byte-use** [2]	Queue buffer usage (in bytes). The size of hardware buffers (in bytes) that are currently allocated for packets in the respective queues.
queue0-byte-max .. queue7-byte-max [2]	Maximum queue buffer fill level (in bytes). Available only on devices that provide the queue statistics service. Use the **reset-counters** command to reset values.

[1] *Port's packet/byte usage can exceed the capacity if **Shared Buffers** are enabled.*

[2] *Only the queues in use are printed.*

QoS Menu

Sub-menu: `/interface/ethernet/switch/qos`

Almost the entire QoS HW configuration is located under **`/in/eth/sw/qos`**. Such an approach allows storing all QoS-related configuration items in one place, easy monitoring and exporting (`/in/eth/sw/qos/export`).

QoS entries have two major flags:

- **H** - Hardware-offloaded.
- **I** - Inactive.

QoS Settings

Sub-menu: `/interface/ethernet/switch/qos/settings`

Property	Description
multicast-buffers-percentage (*integer: 1..90*; Default: **10**)	Maximum amount of packet buffers for multicast/broadcast traffic (% of the total buffer memory).
shared-buffers-percentage (*integer: 0..90*; Default: **40**)	Maximum amount of packet buffers that are shared between ports (% of the total buffer memory). Setting it to 0 disables buffer sharing. The remaining buffer memory is split between the ports.
shared-buffers-color (*all | green-only | yellow-and-green*; Default: **all**)	Restricts shared buffer usage for specific traffic colors only.
shared-pool0-percentage .. shared-pool7-percentage (*integer: 0..100*; Default: **auto**)	If the device supports multiple shared buffer pools, these settings allows adjusting the size of each pool (% of the *shared* buffer memory, where 100% means all shared buffers allocated by the **shared-buffers-percentage** setting). For example, if shared-buffers-percentage=40 and shared-pool0-percentage=50, the shared pool #0 (the first one) receives 20% of the total buffer memory (50% of 40% or "0.5 * 0.4 = 0.2"). Auto mode tries to equally allocate available resources between pools that uses auto setting, and provides at least a minimum of 10% of the total shared buffer size if the sum of other manually configured pools are exceeded. The default setting (auto).
treat-yellow-as (*green | red*; Default: **red**)	For devices that support only two-color traffic marking (red/green). This setting allows using the same QoS profiles for the devices with two- and three-color traffic marking.
wred-queue-margin (*low | medium | high;* Default: **medium**)	A relative amount of packets below a queue cap ("`queueX-packet-cap`" or "`queueX-byte-cap`") to start a random tail drop. This margin is applied only to queues with enabled Weighed Random Early Detection (**wred=yes**) that do NOT use shared buffers (**use-shared-buffers=no)**. The higher the queue buffer fill level, the higher the packet drop chance. The *low* margin means the random tail drop starts later; the *high* - sooner.
wred-shared-queue-margin (*low | medium | high;* Default: **medium**)	Similar to **wred-queue-margin** but applies to queues that use shared buffers (**use-shared-buffers=yes)**. Also affects ECN marking.

QoS Monitor

Command: `/interface/ethernet/switch/qos/monitor`

Example

```
[admin@CCR2216] /interface/ethernet/switch/qos monitor
         total-packet-cap: 32768
         total-packet-use: 0
         total-buffer-cap: 32768
         total-buffer-use: 0
     multicast-packet-cap: 3276
     multicast-packet-use: 0
     multicast-buffer-cap: 3276
     multicast-buffer-use: 0
  shared-pool0-packet-cap: 13107
  shared-pool0-packet-use: 0
  shared-pool0-buffer-cap: 13107
  shared-pool0-buffer-use: 0
```

Monitors hardware QoS resources.

Property	Description
total-packet-cap *(integer)*	Total packet capacity. The maximum number of hardware packet descriptors that the device can store is all queues.
total-packet-use *(integer)*	Total packet usage. The current number of packet descriptors residing in the hardware memory.
total-buffer-cap *(integer)*	Total tx memory capacity (in tx queue buffers). Buffer size depends on the switch chip model (usually - 256 bytes).
total-buffer-use *(integer)*	Total tx memory usage. The current number of buffers occupied by the packets in all tx queues.
multicast-packet-cap *(integer)*	Multicast packet capacity. The maximum number of hardware packet descriptors that can be used by multicast /broadcast traffic. Depends on the **multicast-buffers-percentage** setting.
multicast-packet-use *(integer)*	Multicast packet usage. The hardware makes a copy of the packet descriptor for each multicast destination.
shared-packet-cap (integer)	Shared packet capacity. The maximum number of hardware packet descriptors that can be shared between ports and tx queues. Depends on the **shared-buffers-percentage** setting.
shared-packet-use (integer)	Shared packet usage. The current number of shared packet descriptors used by all tx queues.
shared-buffer-cap *(integer)*	Shared tx memory capacity (in tx queue buffers). Depends on the **shared-buffers-percentage** setting.
shared-buffer-use *(integer)*	Shared tx memory usage. The current number of shared buffers occupied by the packets in all tx queues.
shared-pool0-packet-cap .. shared-pool7-packet-cap (integer)	Shared packet capacity of the each shared pool. Only the shared pools in use are displayed. These fields are omitted if the device does not support multiple shared pools.
shared-pool0-packet-use .. shared-pool7-packet-use (integer)	Per-pool shared packet usage. Only the shared pools in use are displayed. These fields are omitted if the device does not support multiple shared pools.

QoS Profile

Sub-menu: `/interface/ethernet/switch/qos/profile`

QoS profiles determine priority field values (PCP, DSCP) for the forwarded/routed packets. Congestion avoidance/resolution is based on QoS profiles. Each packet gets a QoS profile assigned based on the ingress switch port QoS settings (see `/in/eth/sw/port`).

Property	Description
color (*green* | *yellow* | *red*; Default: **green**)	Trafic color for color-aware drop precedence management. Leave the default value (green) for color-blind drop precedence management.

dscp (*integer: 0..63*; Default: **0**)	IPv4/IPv6 DSCP field value for the egress packets assigned to the QoS profile.
name (*string*; Default:)	The user-defined name of the QoS profile.
pcp (*integer: 0..7*; Default: **0**)	VLAN priority value (IEEE 802.1q PCP - Priority Code Point). Used only if the egress packets assigned to the QoS profile are VLAN-tagged (have the 802.1q header). The value can be further altered via the QoS Egress Map.
traffic-class (*integer: 0..7*; Default: **0**)	The traffic class determines the packet priority and the egress queue (see **tx-manager**). The queue number is usually the same as the traffic class (packets with tc0 go into queue0, tc1 - queue1, ... tc7 - queue7). Unlike pcp, where 0 means the default priority but 1 - the lowest one (and further customizable), traffic classes are strictly ordered. TC0 always selects the lowest priority, etc.

QoS Mapping

Sub-menu: `/interface/ethernet/switch/qos/map`

Priority-to-profile mapping table(-s) for trusted packets. All switch chips have one built-in map - **default**. In addition, some models allow the user to define custom mapping tables and assign different maps to various switch ports via the **qos-map** property:

- devices based on **Marvell Prestera 98DX224S, 98DX226S**, or **98DX3236** switch chip models support only one map - default.
- devices based on **Marvell Prestera 98DX8xxx**, **98DX4xxx** switch chips, or **98DX325x** model devices support up to 12 maps (the default + 11 user-defined).

Property	Description
name (*string*; Default:)	The user-defined name of the mapping table.

VLAN Map

Sub-menu: `/interface/ethernet/switch/qos/map/vlan`

Matches VLAN priorities (802.1p PCP/DEI fields) to QoS profiles. By default, all values are matched to the default QoS profile.

Property	Description
dei-only (yes / *no*; Default: **no**)	Map only packets with DEI (formerly CFI) bit set in the VLAN header.
map (*name*; Default: **default**)	The name of the mapping table.
profile (*name*; Default:)	The name of the QoS profile to assign to the matched packets.
pcp (*integer: 0..7*; Default: **0**)	*Minimum* VLAN priority (PCP) value for the lookup.

DSCP Map

Sub-menu: `/interface/ethernet/switch/qos/map/ip`

Matches DSCP values to QoS profiles.

Property	Description
dscp (*integer: 0..63*; Default: **0**)	*Minimum* DSCP value for the lookup.
map (*name*; Default: **default**)	The name of the mapping table. If not set, the standard (built-in) mapping table gets altered.
profile (*name*; Default:)	The name of the QoS profile to assign to the matched packets.

Transmission Manager

Sub-menu: `/interface/ethernet/switch/qos/tx-manager`

Transmission (Tx) Manager controls packet enqueuing for transmission and packet tx order. Different switch ports can be assigned to different Tx managers. The maximum number of hardware Tx managers depends on the switch chip model (usually - 8).

Property	Description
ecn (*yes* | *no*; Default: **no)**	Enables/disables ECN marking of the transmitted packets.
name (*string*; Default:)	The use-defined name of the Tx Manager

Transmission Queue Scheduler

Sub-menu: `/interface/ethernet/switch/qos/tx-manager/queue`

Each port has eight Tx queues. The assigned Tx Manager controls packet enqueuing and schedules transmission orders. Each queue can have either strict priority (where packets with the highest traffic class are always transmitted first) or grouped together for a weighted round-robin tx schedule.

Creating a Tx Manager automatically creates all eight respective queue schedulers.

> Changing any properties of Tx manager or queues completely halts traffic enqueueing and transmission during the offload process. Temporary packet loss is expected while the device is forwarding traffic.

Property	Description
tx-manager (*name*; *read-only*)	The linked Tx Manager
traffic-class (*integer: 0..7*; *read-only*)	The traffic class (tc0..tc7) and the respective port queue (queue0..queue7) that the scheduler controls.
schedule (*strict-priority | high-priority-group | low-priority-group*)	• **strict-priority** - packets in the respective queue are always scheduled before moving to lower traffic classes. Packets with lower traffic classes are not transmitted until the current queue is empty. • **high-priority-group** - all queues in the group are scheduled together by using a weighted round-robin principle. For example, if TC5 has weight 4, TC4 - 3, and TC3 - 2, then the scheduler transmits 4 packets from queue5, 3 packets from Q4, and 2 packets from Q3 in a single round. To achieve lower latency, each round is "sliced" between all queues in the group. In other words, the packet order in each round of the above example is "Q5, Q4, Q3, Q5, Q4, Q3, Q5, Q4, Q5". • **low-priority-group** - similar logic to the high-priority-group, but the low-priority-group is scheduled only when all queues in the high-priority-group are empty.
weight *(integer: 0..255;* Default: **1**)	The weight value for the traffic class if it is a member of a schedule group. The field is not used in the case of strict priority schedule.
queue-buffers (*integer*; Default: **auto**)	Per-queue buffer pool. The maximum number of packets that can be assigned to the queue (per each assigned port).
use-shared-buffers *(yes | no)*	Allow the queue to use the shared buffer pool when **queue-buffers** are full. If the queue is full and the shared buffers are disabled, the packet gets dropped. If the shared buffers are enabled, the queue may use up to **shared-packet-cap** or **shared-poolX-packet-cap** (see QoS Settings for details) packets from the shared pool.
shared-pool-index *(integer;* Default: **0**)	The shared pool index for the queue to use. Relevant only if **use-shared-buffers=yes** and the device supports multiple shared pools.
wred (*yes* | *no*; Default: **no)**	Enables/disables Weighted Random Early Detection for the given queue.

Switch Chip Features

Introduction

There are several types of switch chips on Routerboards and they have different sets of features. Most of them (from now on "Other") have only the basic "Port Switching" feature, but there are a few with more features:

Feature	QCA8337	Atheros8327	Atheros8316	Atheros8227	Atheros7240	IPQ-PPE	ICPlus175D	MT7621, MT7531	RTL8367	88E6393X	88E6191X, 88E6190	98PX1012
Port Switching	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no
Port Mirroring	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	no
TX limit [1]	yes	yes	yes	yes	yes	no	no	yes	yes	yes	yes	no
RX limit [1]	yes	yes	no	no	no	no	no	yes	yes	yes	yes	no
Host table	2048 entries	2048 entries	2048 entries	1024 entries	2048 entries	2048 entries	2048 entries [2]	2048 entries	2048 entries	16k entries	16k entries	no
Vlan table	4096 entries	4096 entries	4096 entries	4096 entries	16 entries	no	no	4096 entries [3]	4096 entries [3]	4096 entries [3]	4096 entries [3]	no
Rule table	92 rules	92 rules	32 rules	no	no	no	no	no	no	256	no	no

Notes

1. For QCA8337, Atheros8327, Atheros8316, Atheros8227, and Atheros7240 the Tx/Rx rate limits can be changed with `bandwidth` property on "`/interface ethernet`" menu, see more details in the Ethernet manual. For RTL8367, 88E6393X, 88E6191X, 88E6190, MT7621 and MT7531 Tx /Rx rate limit can be changed with `egress-rate` and `ingress-rate` properties on "`/interface ethernet switch port`" menu.
2. MAC addresses are learned up to the specified number, but the content of a switch host table is not available in RouterOS and static host configuration is not supported.
3. Bridge HW vlan-filtering was added in the RouterOS 7.1rc1 (for RTL8367) and 7.1rc5 (for MT7621) versions. The switch does not support other `ether-type` 0x88a8 or 0x9100 (only 0x8100 is supported) and no `tag-stacking`. Using these features will disable HW offload.

⚠ Cloud Router Switch (CRS) series devices have highly advanced switch chips built-in, they support a wide variety of features. For more details about switch chip capabilities on CRS1xx/CRS2xx series devices check the CRS1xx/CRS2xx series switches manual, for CRS3xx series devices check the CRS3xx, CRS5xx series switches, and CCR2116, CCR2216 routers manual.

RouterBoard	Switch-chip description

C52iG-5HaxD2HaxD-TC (hAP ax^2), C53UiG+5HPaxD2HPaxD (hAP ax^3), Chateau ax series	IPQ-PPE (ether1-ether5)
cAPGi-5HaxD2HaxD (cAP ax)	IPQ-PPE (ether1-ether2)
L009 series	88E6190 (ether2-ether8, sfp1)
RB5009 series	88E6393X (ether1-ether8, sfp-sfpplus1)
CCR2004-16G-2S+	88E6191X (ether1-ether8); 88E6191X (ether9-ether16);
RB4011iGS+	RTL8367 (ether1-ether5); RTL8367 (ether6-ether10);
RB1100AHx4	RTL8367 (ether1-ether5); RTL8367 (ether6-ether10); RTL8367 (ether11-ether13)
L41G-2axD (hAP ax lite)	MT7531 (ether1-ether4)
RB750Gr3 (hEX), RB760iGS (hEX S)	MT7621 (ether1-ether5)
RBM33G	MT7621 (ether1-ether3)
RB3011 series	QCA8337 (ether1-ether5); QCA8337 (ether6-ether10)
RB OmniTik ac series	QCA8337 (ether1-ether5)
RBwsAP-5Hac2nD (wsAP ac lite)	Atheros8227 (ether1-ether3)
RB941-2nD (hAP lite)	Atheros8227 (ether1-ether4)
RB951Ui-2nD (hAP); RB952Ui-5ac2nD (hAP ac lite); RB750r2 (hEX lite); RB750UPr2 (hEX PoE lite); RB750P-PBr2 (PowerBox); RB750P r2; RBOmniTikU-5HnDr2 (OmniTIK 5); RBOmniTikUPA-5HnDr2 (OmniTIK 5 PoE)	Atheros8227 (ether1-ether5)
RB750Gr2 (hEX); RB962UiGS-5HacT2HnT (hAP ac); RB960PGS (hEX PoE); RB960PGS-PB (PowerBox Pro)	QCA8337 (ether1-ether5)
RB953GS	Atheros8327 (ether1-ether3+sfp1)
RB850Gx2	Atheros8327 (ether1-ether5) with ether1 optional
RB2011 series	Atheros8327 (ether1-ether5+sfp1); Atheros8227 (ether6-ether10)
RB750GL; RB751G-2HnD; RB951G-2HnD; RBD52G-5HacD2HnD (hAP ac²), RBD53iG-5HacD2HnD (hAP ac³), RBD53GR-5HacD2HnD&R11e-LTE6 (hAP ac³ LTE6 kit), RBD53G-5HacD2HnD-TC&EG12-EA (Chateau LTE12)	Atheros8327 (ether1-ether5)
RBcAPGi-5acD2nD (cAP ac), RBwAPGR-5HacD2HnD (wAP R ac and wAP ac LTE series), RBwAPG-5HacD2HnD (wAP ac), RBD25G-5HPacQD2HPnD (Audience), RBD25GR-5HPacQD2HPnD&R11e-LTE6 (Audience LTE6 kit),	Atheros8327 (ether1-ether2)
RBD22UGS-5HPacD2HnD (mANTBox 52 15s)	Atheros8327 (ether1-sfp1)
RB1100AH	Atheros8327 (ether1-ether5); Atheros8327 (ether6-ether10)
RB1100AHx2	Atheros8327 (ether1-ether5); Atheros8327 (ether6-ether10)
CCR1009-8G-1S-1S+; CCR1009-8G-1S	Atheros8327 (ether1-ether4)
RB493G	Atheros8316 (ether1+ether6-ether9); Atheros8316 (ether2-ether5)
RB435G	Atheros8316 (ether1-ether3) with ether1 optional

RB450G	Atheros8316 (ether1-ether5) with ether1 optional
RB450Gx4	Atheros8327 (ether1-ether5)
RB433GL	Atheros8327 (ether1-ether3)
RB750G	Atheros8316 (ether1-ether5)
RB1200	Atheros8316 (ether1-ether5)
RB1100	Atheros8316 (ether1-ether5); Atheros8316 (ether6-ether10)
DISC Lite5	Atheros8227 (ether1)
RBmAP2nD	Atheros8227 (ether1-ether2)
RBmAP2n	Atheros7240 (ether1-ether2)
RB750	Atheros7240 (ether2-ether5)
RB750UP	Atheros7240 (ether2-ether5)
RB751U-2HnD	Atheros7240 (ether2-ether5)
RB951-2n	Atheros7240 (ether2-ether5)
RB951Ui-2HnD	Atheros8227 (ether1-ether5)
RB433 series	ICPlus175D (ether2-ether3); older models had ICPlus175C
RB450	ICPlus175D (ether2-ether5); older models had ICPlus175C
RB493 series	ICPlus178C (ether2-ether9)
RB816	ICPlus178C (ether1-ether16)

The command-line configuration is under the switch menu. This menu contains a list of all switch chips present in the system and some sub-menus as well.

```
[admin@MikroTik] > /interface ethernet switch print
Flags: I - invalid
 #   NAME         TYPE            MIRROR-SOURCE       MIRROR-TARGET       SWITCH-ALL-PORTS
 0   switch1      Atheros-8327    none                none
 1   switch2      Atheros-8227    none                none
```

Depending on the switch type there can be different configuration capabilities available.

Features

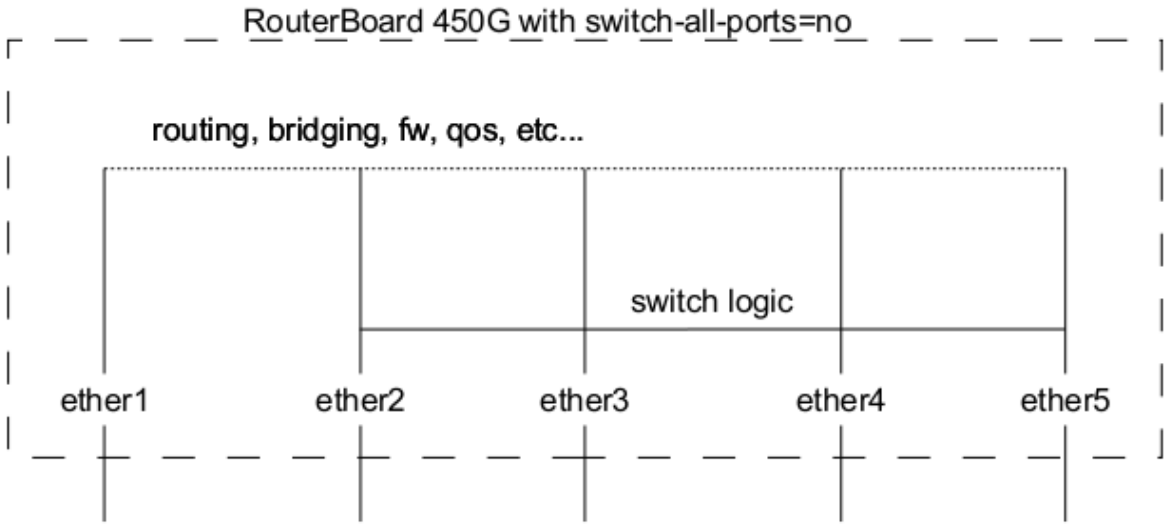
Port Switching

To set up port switching on non-CRS series devices, check the Bridge Hardware Offloading page.

> ⚠ Port switching in RouterOS v6.41 and newer is done using the bridge configuration. Before RouterOS v6.41 port switching was done using the master-port property.

Switch All Ports Feature

Ether1 port on RB450G/RB435G/RB850Gx2 devices has a feature that allows it to be removed/added to the default switch group, this setting is available on the `/interface ethernet switch` menu. By default ether1 port will be included in the switch group.



Property	Description
switch-all-ports (no / *yes*; Default: **yes**)	Changes ether1 switch group only on RB450G/RB435G/RB850Gx2 devices. • `yes` - ether1 is part of the switch and supports switch grouping and all other advanced Atheros8316/Atheros8327 features including extended statistics (`/interface ethernet print stats`). • `no` - ether1 is not part of the switch, effectively making it a stand-alone ethernet port, this way increasing its throughput to other ports in bridged and routed mode, but removing the switching possibility on this port.

Port Mirroring

Port mirroring lets the switch to copy all traffic that is going in and out of one port (`mirror-source`) and send out these copied frames to some other port (`mirror-target`). This feature can be used to easily set up a 'tap' device that receives all traffic that goes in/out of some specific port. Note that `mirror-source` and `mirror-target` ports have to belong to the same switch (see which port belongs to which switch in `/interface ethernet` menu). Also, mirror-target can have a special 'cpu' value, which means that mirrored packets should be sent out to the switch chips CPU port. Port mirroring happens independently of switching groups that have or have not been set up.

Sub-menu: `/interface ethernet switch`

Property	Description
mirror-source (*name* / *none*; Default: **none**)	Selects a single mirroring source port. Ingress and egress traffic will be sent to the `mirror-target` port. Note that `mirror-target` port has to belong to the same switch (see which port belongs to which switch in `/interface ethernet` menu).
mirror-target (*name* / *none* / *cpu*; Default: **none**)	Selects a single mirroring target port. Mirrored packets from `mirror-source` and `mirror` (see the property in rule and host table) will be sent to the selected port.
mirror-egress-target (*name* / *none*; Default: **none**)	Selects a single mirroring egress target port, only available on **88E6393X**, **88E6191X** and **88E6190** switch chips. Mirrored packets from `mirror-egress` (see the property in port menu) will be sent to the selected port.

Sub-menu: `/interface ethernet switch rule`

Property	Description
mirror (*no* / *yes*; Default: **no**)	Whether to send a packet copy to `mirror-target` port.
mirror-ports (*name*; Default:)	Selects multiple mirroring target ports, only available on **88E6393X** switch chip. Matched packets in the ACL rule will be copied and sent to selected ports.

Sub-menu: `/interface ethernet switch host`

Property	Description
mirror (*no* / *yes*; Default: **no**)	Whether to send a frame copy to `mirror-target` port from a frame with a matching MAC destination address (matching destination or source address for CRS3xx series switches)

Sub-menu: `/interface ethernet switch port`

Property	Description
mirror-egress (*no* / *yes*; Default: **no**)	Whether to send egress packet copy to the `mirror-egress-target` port, only available on **88E6393X**, **88E6191X** and **88E6190** switch chips.
mirror-ingress (*no* / *yes*; Default: **no**)	Whether to send ingress packet copy to the `mirror-ingress-target` port, only available on **88E6393X**, **88E6191X** and **88E6190** switch chips.
mirror-ingress-target (*name* / *none*; Default: **none**)	Selects a single mirroring ingress target port, only available on **88E6393X**, **88E6191X** and **88E6190** switch chips. Mirrored packets from `mirror-ingress` will be sent to the selected port.

Port mirroring configuration example:

```
/interface ethernet switch
set switch1 mirror-source=ether2 mirror-target=ether3
```

If you set mirror-source as an Ethernet port for a device with at least two switch chips and these mirror-source ports are in a single bridge while mirror-target for both switch chips are set to send the packets to the CPU, then this will result in a loop, which can make your device inaccessible.

Port Settings

Properties under this menu are used to configure VLAN switching and filtering options for switch chips that support a VLAN Table. These properties are only available to switch chips that have VLAN Table support, check the Switch Chip Features table to make sure your device supports such a feature.

Ingress traffic is considered as traffic that is being sent **IN** a certain port, this port is sometimes called **ingress port**. Egress traffic is considered as traffic that is being sent **OUT** of a certain port, this port is sometimes called **egress port**. Distinguishing them is very important to properly set up VLAN filtering since some properties apply only to either ingress or egress traffic.

Property	Description
vlan-mode (*check* / *disabled* / *fallback* / *secure*; Default: **disabled**)	Changes the VLAN lookup mechanism against the VLAN Table for ingress traffic. • `disabled` - disables checking against the VLAN Table completely for ingress traffic. No traffic is dropped when set on the ingress port. • `fallback` - checks tagged traffic against the VLAN Table for ingress traffic and forwards all untagged traffic. If ingress traffic is tagged and the egress port is not found in the VLAN table for the appropriate VLAN ID, then traffic is dropped. If a VLAN ID is not found in the VLAN Table, then traffic is forwarded. Used to allow known VLANs only in specific ports. • `check` - checks tagged traffic against the VLAN Table for ingress traffic and drops all untagged traffic. If ingress traffic is tagged and the egress port is not found in the VLAN table for the appropriate VLAN ID, then traffic is dropped. • `secure` - checks tagged traffic against the VLAN Table for ingress traffic and drops all untagged traffic. Both ingress and egress port must be found in the VLAN Table for the appropriate VLAN ID, otherwise, traffic is dropped.
vlan-header (*add-if-missing* / *always-strip* / *leave-as-is*; Default: **leave-as-is**)	Sets action which is performed on the port for egress traffic. • `add-if-missing` - adds a VLAN tag on egress traffic and uses default-vlan-id from the ingress port. Should be used for trunk ports. • `always-strip` - removes a VLAN tag on egress traffic. Should be used for access ports. • `leave-as-is` - does not add nor remove a VLAN tag on egress traffic. Should be used for hybrid ports.

default-vlan-id (*auto* / *integer: 0..4095*; Default: **auto**)	Adds a VLAN tag with the specified VLAN ID on all untagged ingress traffic on a port, should be used with vlan-header set to `always-strip` on a port to configure the port to be the access port. For hybrid ports default-vlan-id is used to tag untagged traffic. If two ports have the same default-vlan-id, then VLAN tag is not added since the switch chip assumes that traffic is being forwarded between access ports.

On **QCA8337** and **Atheros8327** switch chips, a default `vlan-header=leave-as-is` property should be used. The switch chip will determine which ports are access ports by using the `default-vlan-id` property. The `default-vlan-id` should only be used on access/hybrid ports to specify which VLAN the untagged ingress traffic is assigned to.

VLAN Table

VLAN table specifies certain forwarding rules for packets that have a specific 802.1Q tag. Those rules are of higher priority than switch groups configured using the Bridge Hardware Offloading feature. Basically, the table contains entries that map specific VLAN tag IDs to a group of one or more ports. Packets with VLAN tags leave the switch chip through one or more ports that are set in the corresponding table entry. The exact logic that controls how packets with VLAN tags are treated is controlled by a `vlan-mode` parameter that is changeable per switch port.

VLAN ID based forwarding takes into account the MAC addresses dynamically learned or manually added in the host table. QCA8337 and Atheros8327 switch-chips also support Independent VLAN Learning (IVL) which does the learning based on both - MAC addresses and VLAN IDs, thus allowing the same MAC to be used in multiple VLANs.

Packets without VLAN tag are treated just as if they had a VLAN tag with port `default-vlan-id`. If `vlan-mode=check` or `vlan=mode=secure` is configured, to forward packets without VLAN tags you have to add an entry to the VLAN table with the same VLAN ID according to `default-vlan-id`.

Property	Description
disabled (*no* / *yes*; Default: **no**)	Enables or disables switch VLAN entry.
independent-learning (no / *yes*; Default: **yes**)	Whether to use shared-VLAN-learning (SVL) or independent-VLAN-learning (IVL).
ports (*name*; Default: **none**)	Interface member list for the respective VLAN. This setting accepts comma-separated values. e.g. `ports=ether1,ether2`.
switch (*name*; Default: **none**)	Name of the switch for which the respective VLAN entry is intended for.
vlan-id (*integer: 0..4095*; Default:)	The VLAN ID for certain switch port configurations.

Devices with **MT7621**, **MT7531**, **RTL8367**, **88E6393X**, **88E6191X**, **88E6190** switch chips support HW offloaded vlan-filtering in RouterOS v7. VLAN-related configuration on the "/interface ethernet switch" menu is not available.

VLAN Forwarding

Both `vlan-mode` and `vlan-header` along with the VLAN Table can be used to configure VLAN tagging, untagging and filtering, multiple combinations are possible, each achieving a different result. Below you can find a table of what kind of traffic is going to be sent out through an egress port when a certain traffic is received on an ingress port for each VLAN Mode.

NOTES:

- **L** - `vlan-header` is set to `leave-as-is`
- **S** - `vlan-header` set to `always-strip`
- **A** - `vlan-header` set to `add-if-missing`
- **U** - Untagged traffic is sent out
- **T** - Tagged traffic is sent out, a tag is already present on the ingress port
- **TA** - Tagged traffic is sent out, a tag was added on the ingress port
- **DI** - Traffic is dropped on ingress port because of mode selected in vlan-mode
- **DE** - Traffic is dropped on egress port because egress port was not found in the VLAN Table
- **VID match** - VLAN ID from the VLAN tag for ingress traffic is present in the VLAN Table
- **Port match** - Ingress port is present in the VLAN Table for the appropriate VLAN ID

VLAN Mode = disabled	**Egress port not present in VLAN Table**	**Egress port is present in VLAN Table**

	L	S	A	L	S	A
Untagged traffic	U	U	TA	U	U	TA
Tagged traffic; no VID match	T	U	T			
Tagged traffic; VID match; no Port match	T	U	T	T	U	T
Tagged traffic; VID match; Port match	T	U	T	T	U	T

VLAN Mode = fallback	**Egress port not present in VLAN Table**			**Egress port is present in VLAN Table**		
	L	**S**	**A**	**L**	**S**	**A**
Untagged traffic	U	U	TA	U	U	TA
Tagged traffic; no VID match	T	U	T			
Tagged traffic; VID match; no Port match	DE	DE	DE	T	U	T
Tagged traffic; VID match; Port match	DE	DE	DE	T	U	T

VLAN Mode = check	**Egress port not present in VLAN Table**			**Egress port is present in VLAN Table**		
	L	**S**	**A**	**L**	**S**	**A**
Untagged traffic						
Tagged traffic; no VID match	DI	DI	DI			
Tagged traffic; VID match; no Port match	DE	DE	DE	T	U	T
Tagged traffic; VID match; Port match	DE	DE	DE	T	U	T

VLAN Mode = secure	**Egress port not present in VLAN Table**			**Egress port is present in VLAN Table**		
	L	**S**	**A**	**L**	**S**	**A**
Untagged traffic						
Tagged traffic; no VID match	DI	DI	DI			
Tagged traffic; VID match; no Port match	DI	DI	DI	DI	DI	DI
Tagged traffic; VID match; Port match	DE	DE	DE	T	U	T

⚠ The tables above are meant for more advanced configurations and to double-check your understanding of how packets will be processed with each VLAN related property.

Host Table

The host table represents switch chip's internal MAC address to port mapping. It can contain two kinds of entries: dynamic and static. Dynamic entries get added automatically, this is also called a learning process: when switch chip receives a packet from a certain port, it adds the packet's source MAC address and port it received the packet from to the host table, so when a packet comes in with the same destination MAC address, it knows to which port it should forward the packet. If the destination MAC address is not present in the host table (so-called unknown-unicast traffic) then it forwards the packet to all ports in the group. Dynamic entries take about 5 minutes to time out. Learning is enabled only on ports that are configured as part of the switch group, so you won't see dynamic entries if you have not set up port switching. Also, you can add static entries that take over dynamic if a dynamic entry with the same MAC address already exists. Since port switching is configured using a bridge with hardware offloading, any static entries created on one table (either bridge host or switch host) will appear on the opposite table as a dynamic entry. Adding a static entry on the switch host table will provide access to some more functionality that is controlled via the following params:

Property	Description
copy-to-cpu (*no* / *yes*; Default: **no**)	Whether to send a frame copy to switch CPU port from a frame with a matching MAC destination address (matching destination or source address for CRS3xx series switches)
drop (*no* / *yes*; Default: **no**)	Whether to drop a frame with a matching MAC source address received on a certain port (matching destination or source address for CRS3xx series switches)
mac-address (*MAC;* Default: **00:00:00:00:00:00**)	Host's MAC address
mirror (*no* / *yes*; Default: **no**)	Whether to send a frame copy to `mirror-target` port from a frame with a matching MAC destination address (matching destination or source address for CRS3xx series switches)
ports (*name*; Default: **none**)	Name of the interface, static MAC address can be mapped to more than one port, including switch CPU port
redirect-to-cpu (*no* / *yes*; Default: **no**)	Whether to redirect a frame to switch CPU port from a frame with a matching MAC destination address (matching destination or source address for CRS3xx series switches)
share-vlan-learned (*no* / *yes*; Default: **no**)	Whether the static host MAC address lookup is used with shared-VLAN-learning (SVL) or independent-VLAN-learning (IVL). The SVL mode is used for those VLAN entries that do not support IVL or IVL is disabled (independent-learning=no)
switch (*name*; Default: **none**)	Name of the switch to which the MAC address is going to be assigned to
vlan-id (*integer: 0..4095*; Default:)	VLAN ID for the statically added MAC address entry

Every switch chip has a finite number of MAC addresses it can store on the chip, see the Introduction table for a specific host table size. Once a host table is full, different techniques can be utilized to cope with the situation, for example, the switch can remove older entries to free space for more recent MAC addresses (used on QCA-8337 and Atheros-8327 switch chips), another option is to simply ignore the new MAC addresses and only remove entries after a timeout has passed (used on Atheros8316, Atheros8227, Atheros-7240, ICPlus175D and Realtek-RTL8367 switch chips), the last option is a combination of the previous two - only allow a certain amount of entries to be renewed and keep the other host portion intact till the timeout (used on MediaTek-MT7621, MT7531 switch chip). These techniques cannot be changed with configuration.

For Atheros8316, Atheros8227 and Atheros-7240 switch chips, the switch-cpu port will always participate in the host learning process when at least one hardware offloaded bridge port is active on the switching group. It will cause the switch-cpu port to learn MAC addresses from non-HW offloaded interfaces. This might cause packet loss when a single bridge contains HW and non-HW offloaded interfaces. Also, packet loss might appear when a duplicate MAC address is used on the same switching group regardless if hosts are located on different logical networks. It is recommended to use HW offloading only when all bridge ports can use HW offloaded or keep it disabled on all switch ports when one or more bridge ports cannot be configured with HW offloading.

Rule Table

Rule table is a very powerful tool allowing wire-speed packet filtering, forwarding and VLAN tagging based on L2, L3 and L4 protocol header field conditions. The menu contains an ordered list of rules just like in `/ip firewall filter`, so ACL rules are checked for each packet until a match has been found. If multiple rules can match, then only the first rule will be triggered. A rule without any action parameters is a rule to accept the packet.

Each rule contains a conditions part and an action part. The action part is controlled by the following parameters:

Property	Description
copy-to-cpu (*no* / *yes*; Default: **no**)	Whether to send a packet copy to switch CPU port
mirror (*no* / *yes*; Default: **no**)	Whether to send a packet copy to `mirror-target` port
new-dst-ports (*name*; Default: **none**)	Changes the destination port as specified, multiple ports allowed, including a switch CPU port. An empty setting will drop the packet. When the parameter is not used, the packet will be accepted
new-vlan-id (*integer: 0..4095*)	Changes the VLAN ID to the specified value or adds a new VLAN tag if one was not already present (the property only applies to the **Atheros8316**, and **88E6393X** switch chips)
new-vlan-priority (*integer: 0..7*)	Changes the VLAN priority field (priority code point, the property only applies to **Atheros8327**, **QCA8337** and **Atheros8316** switch chips)
rate (*integer: 0..4294967295*)	Sets ingress traffic limitation (bits per second) for matched traffic, can only be applied to the first 32 rule slots (the property only applies to **Atheros8327/QCA8337** switch chips)
redirect-to-cpu (*no* / *yes*; Default: **no**)	Changes the destination port of a matching packet to the switch CPU

The conditions part is controlled by the rest of the parameters:

Property	Description
disabled (*no* / *yes*; Default: **no**)	Enables or disables switch rule
dscp (*integer: 0..63*)	Matching DSCP field of the packet
dst-address (*IP address/Mask*)	Matching destination IP address and mask
dst-address6 (*IPv6 address/Mask*)	Matching destination IPv6 address and mask
dst-mac-address (*MAC address/Mask*)	Matching destination MAC address and mask
dst-port (*integer: 0..65535*)	Matching destination protocol port number or range
flow-label (*integer: 0..1048575*)	Matching IPv6 flow label
mac-protocol (*802.2 / arp / homeplug-av / ip / ipv6 / ipx / lldp / loop-protect / mpls-multicast / mpls-unicast / packing-compr / packing-simple / pppoe / pppoe-discovery / rarp / service-vlan / vlan / or 0..65535 / or 0x0000-0xffff*)	Matching particular MAC protocol specified by protocol name or number (skips VLAN tags if any)
ports (*name*)	Name of the interface on which the rule will apply on the received traffic, multiple ports are allowed
protocol (*dccp / ddp / egp / encap / etherip / ggp / gre / hmp / icmp / icmpv6 / idpr-cmtp / igmp / ipencap / ipip / ipsec-ah / ipsec-esp / ipv6 / ipv6-frag / ipv6-nonxt / ipv6-opts / ipv6-route / iso-tp4 / l2tp / ospf / pim / pup / rdp / rspf / rsvp / sctp / st / tcp / udp / udp-lite / vmtp / vrrp / xns-idp / xtp / or 0..255*)	Matching particular IP protocol specified by protocol name or number
src-address (*IP address/Mask*)	Matching source IP address and mask
src-address6 (*IPv6 address/Mask*)	Matching source IPv6 address and mask
src-mac-address (*MAC address/Mask*)	Matching source MAC address and mask
src-port (*0..65535*)	Matching source protocol port number or range
switch (*switch group*)	Matching switch group on which will the rule apply
traffic-class (*0..255*)	Matching IPv6 traffic class
vlan-id (*0..4095*)	Matching VLAN ID (the property only applies to the Atheros8316, Atheros8327, QCA8337, 88E6393X switch chips)

vlan-header (*not-present* | *present*)	Matching VLAN header, whether the VLAN header is present or not (the property only applies to the Atheros8316, Atheros8327, QCA8337, 88E6393X switch chips)
vlan-priority (*0..7*)	Matching VLAN priority (priority code point)

⚠ IPv4 and IPv6 specific conditions cannot be present in the same rule.

⚠ Because the rule table is processed entirely in switch chips hardware, there is a limitation to how many rules you may have. Depending on the number of conditions (MAC layer, IP layer, IPv6, L4 layer) you use in your rules, the number of active rules may vary from 8 to 32 for Atheros8316 switch chip, from 24 to 96 for Atheros8327/QCA8337 switch chip and from 42 to 256 for 88E6393X switch chip. You can always do `/interface ethernet switch rule print` after modifying your rule set to see that no rules at the end of the list are 'invalid' which means those rules did not fit into the switch chip.

Port isolation

Port isolation provides the possibility to divide (isolate) certain parts of your network, this might be useful when you need to make sure that certain devices cannot access other devices, this can be done by isolating switch ports. Port isolation only works between ports that are members of the same switch. Switch port isolation is available on all switch chips since RouterOS v6.43.

Property	Description
forwarding-override (*interface*; Default:)	Forces ingress traffic to be forwarded to a specific interface. Multiple interfaces can be specified by separating them with a comma.

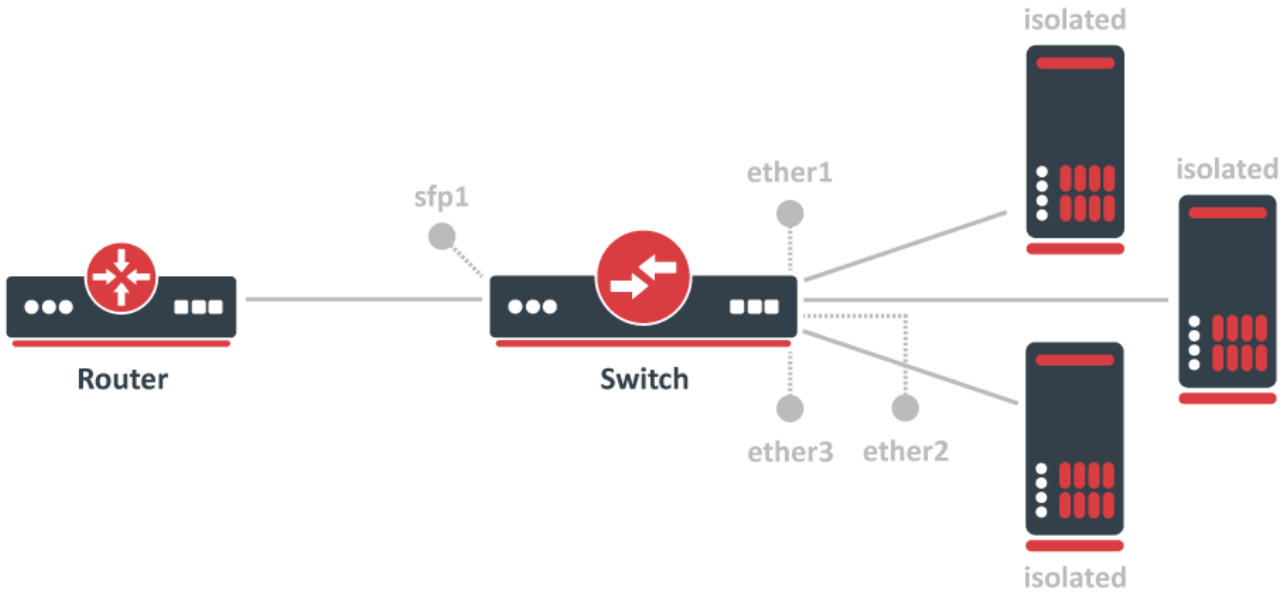
⚠ (R/M)STP will only work properly in PVLAN setups, (R/M)STP will not work properly in setups, where there are multiple isolated switch groups, because switch groups might not properly receive BPDUs and therefore fail to detect network loops.

The `forwarding-override` property affects ingress traffic only. Switch ports that do not have the `forwarding-override` specified can send packets through all switch ports.

Switch chips with a VLAN table support (**QCA8337**, **Atheros8327**, **Atheros8316**, **Atheros8227** and **Atheros7240**) can override the port isolation configuration when enabling a VLAN lookup on the switch port (the `vlan-mode` is set to `fallback`, `check` or `secure`). If additional port isolation is needed between ports on the same VLAN, a switch rule with a new-dst-ports property can be implemented. Other devices without switch rule support cannot overcome this limitation.

Private VLAN

In some scenarios, you might need to forward all traffic to an uplink port while all other ports are isolated from each other. This kind of setup is called **Private VLAN** configuration, the **Switch** will forward all Ethernet frames directly to the uplink port allowing the **Router** to filter unwanted packets and limit access between devices that are behind switch ports.



To configure switch port isolation, you need to switch all required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add interface=sfp1 bridge=bridge1 hw=yes
add interface=ether1 bridge=bridge1 hw=yes
add interface=ether2 bridge=bridge1 hw=yes
add interface=ether3 bridge=bridge1 hw=yes
```

⚠ By default, the bridge interface is configured with `protocol-mode` set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

Override the egress port for each switch port that needs to be isolated (excluding the uplink port):

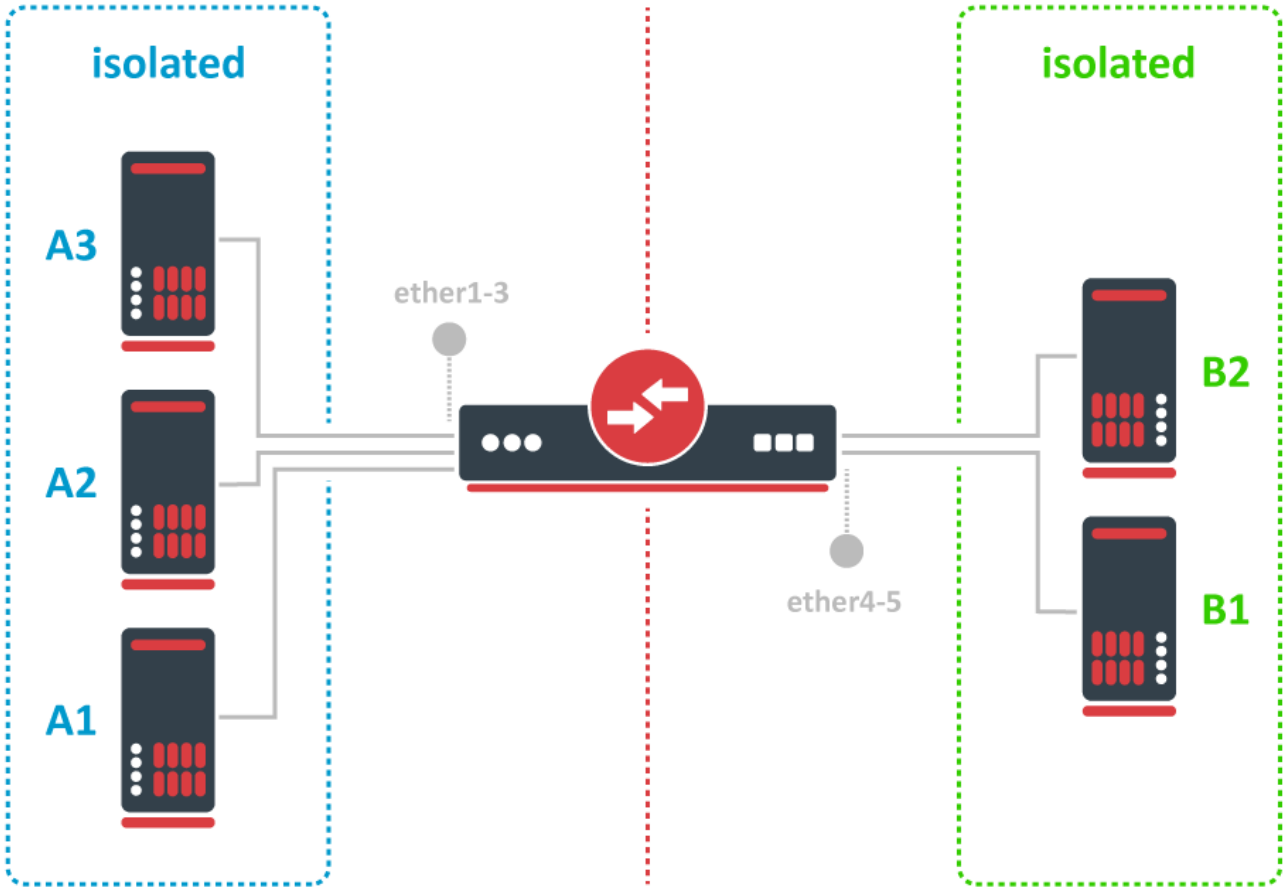
```
/interface ethernet switch port-isolation
set ether1 forwarding-override=sfp1
set ether2 forwarding-override=sfp1
set ether3 forwarding-override=sfp1
```

⚠ It is possible to set multiple uplink ports for a single switch chip, this can be done by specifying multiple interfaces and separating them with a comma.

Isolated switch groups

In some scenarios you might need to isolate a group of devices from other groups, this can be done using the switch port isolation feature. This is useful when you have multiple networks but you want to use a single switch, with port isolation you can allow certain switch ports to be able to communicate through only a set of switch ports. In this example, devices on **ether1-3** will only be able to communicate with devices that are on **ether1-3**, while devices on **ether4-5** will only be able to communicate with devices on **ether4-5** (**ether1-3** is not able to communicate with **ether4-5**)

⚠ Port isolation is only available between ports that are members of the same switch.



To configure isolated switch groups you must first switch all ports:

```
/interface bridge
add name=bridge
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

⚠ By default, the bridge interface is configured with `protocol-mode` set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

Then specify in the `forwarding-override` property all ports that you want to be in the same isolated switch group (except the port on which you are applying the property), for example, to create an isolated switch group for **A** devices:

```
/interface ethernet switch port-isolation
set ether1 forwarding-override=ether2,ether3
set ether2 forwarding-override=ether1,ether3
set ether3 forwarding-override=ether1,ether2
```

To create an isolated switch group for **B** devices:

```
/interface ethernet switch port-isolation
set ether4 forwarding-override=ether5
set ether5 forwarding-override=ether4
```

CPU Flow Control

All switch chips have a special port that is called **switchX-cpu**, this is the CPU port for a switch chip, it is meant to forward traffic from a switch chip to the CPU, such a port is required for management traffic and routing features. By default the switch chip ensures that this special CPU port is not congested and sends out Pause Frames when link capacity is exceeded to make sure the port is not oversaturated, this feature is called **CPU Flow Control**. Without this feature packets that might be crucial for routing or management purposes might get dropped.

Since RouterOS v6.43 it is possible to disable the CPU Flow Control feature on some devices that are using one of the following switch chips: Atheros8227, QCA8337, Atheros8327, Atheros7240 or Atheros8316. Other switch chips have this feature enabled by default and cannot be changed. To disable CPU Flow Control use the following command:

```
/interface ethernet switch set switch1 cpu-flow-control=no
```

Statistics

Some switch chips are capable of reporting statistics, this can be useful to monitor how many packets are sent to the CPU from the built-in switch chip. These statistics can also be used to monitor CPU Flow Control. You can find an example of the switch chip's statistics below:

```
[admin@MikroTik] > /interface ethernet switch print stats

                     name:      switch1
           driver-rx-byte:  221 369 701
         driver-rx-packet:    1 802 975
           driver-tx-byte:   42 621 969
         driver-tx-packet:      310 485
                  rx-bytes:  414 588 529
                rx-packet:    2 851 236
             rx-too-short:            0
              rx-too-long:            0
             rx-broadcast:    1 040 309
                 rx-pause:            0
             rx-multicast:      486 321
             rx-fcs-error:            0
           rx-align-error:            0
              rx-fragment:            0
               rx-control:            0
            rx-unknown-op:            0
          rx-length-error:            0
            rx-code-error:            0
         rx-carrier-error:            0
                rx-jabber:            0
                  rx-drop:            0
                tx-bytes:   44 071 621
                tx-packet:      312 597
             tx-too-short:            0
              tx-too-long:        8 397
             tx-broadcast:        2 518
                 tx-pause:        2 112
             tx-multicast:        7 142
    tx-excessive-collision:            0
     tx-multiple-collision:            0
       tx-single-collision:            0
     tx-excessive-deferred:            0
               tx-deferred:            0
         tx-late-collision:            0
        tx-total-collision:            0
                   tx-drop:            0
                tx-jabber:            0
             tx-fcs-error:            0
               tx-control:        2 112
              tx-fragment:            0
                 tx-rx-64:        6 646
             tx-rx-65-127:    1 509 891
            tx-rx-128-255:    1 458 299
            tx-rx-256-511:      178 975
           tx-rx-512-1023:          953
          tx-rx-1024-1518:          672
           tx-rx-1519-max:            0
```

Some devices have multiple CPU cores that are directly connected to a built-in switch chip using separate data lanes. These devices can report which data lane was used to forward the packet from or to the CPU port from the switch chip. For such devices an extra line is added for each row, the first line represents data that was sent using the first data lane, the second line represents data that was sent using the second data line, and so on. You can find an example of the switch chip's statistics for a device with multiple data lanes connecting the CPU and the built-in switch chip:

```
[admin@MikroTik] > /interface ethernet switch print stats
                  name:      switch1
        driver-rx-byte:  226 411 248
                                   0
      driver-rx-packet:    1 854 971
                                   0
        driver-tx-byte:   45 988 067
                                   0
      driver-tx-packet:      345 282
                                   0
              rx-bytes:  233 636 763
                                   0
             rx-packet:    1 855 018
                                   0
          rx-too-short:            0
                                   0
           rx-too-long:            0
                                   0
              rx-pause:            0
                                   0
          rx-fcs-error:            0
                                   0
           rx-overflow:            0
                                   0
              tx-bytes:   47 433 203
                                   0
             tx-packet:      345 282
                                   0
    tx-total-collision:            0
                                   0
```

Setup Examples

Make sure you have added all needed interfaces to the VLAN table when using secure `vlan-mode`. For routing functions to work properly on the same device through ports that use secure `vlan-mode`, you will need to allow access to the CPU from those ports, this can be done by adding the switchX-cpu interface itself to the VLAN table. Examples can be found in the Management port section.

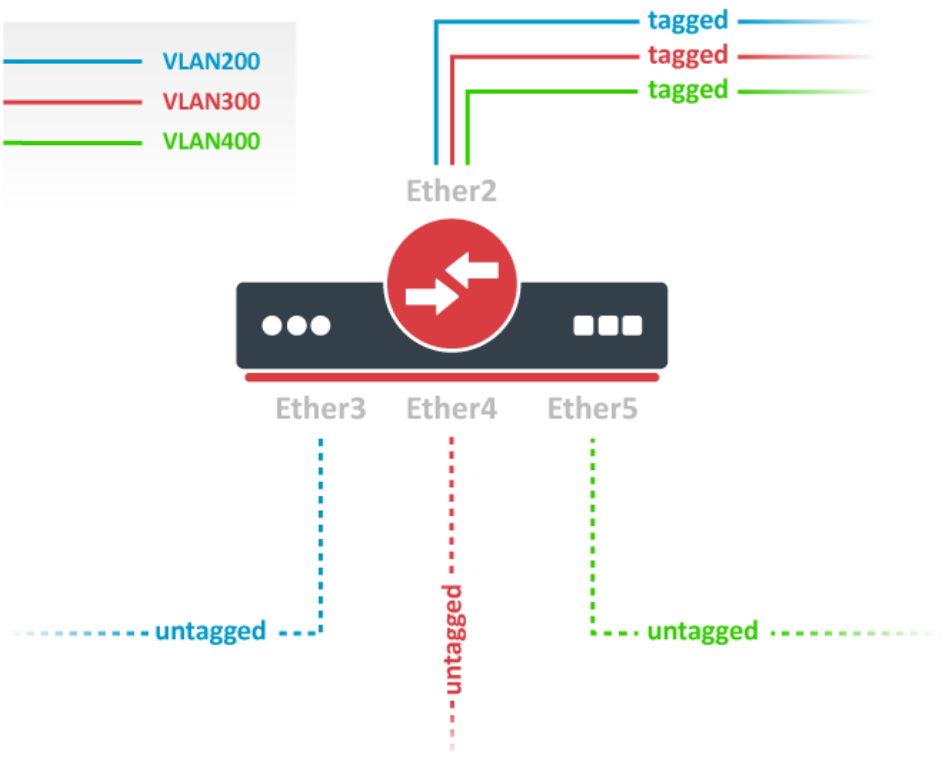
It is possible to use the built-in switch chip and the CPU at the same time to create a Switch-Router setup, where a device acts as a switch and as a router at the same time. You can find a configuration example in the Switch-Router guide.

When allowing access to the CPU, you are allowing access from a certain port to the actual router/switch, this is not always desirable. Make sure you implement proper firewall filter rules to secure your device when access to the CPU is allowed from a certain VLAN ID and port, use firewall filter rules to allow access to only certain services.

Devices with **MT7621**, **MT7531**, **RTL8367**, **88E6393X**, **88E6191X, 88E6190** switch chips support HW offloaded vlan-filtering in RouterOS v7. VLAN-related configuration on the "/interface ethernet switch" menu is not available.

VLAN Example 1 (Trunk and Access Ports)

RouterBOARDs with Atheros switch chips can be used for 802.1Q Trunking. This feature in RouterOS v6 is supported by **QCA8337, Atheros8316, Atheros8327, Atheros8227** and **Atheros7240** switch chips. In this example, **ether3**, **ether4,** and **ether5** interfaces are access ports, while **ether2** is a trunk port. VLAN IDs for each access port: ether3 - 400, ether4 - 300, ether5 - 200.



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

⚠ By default, the bridge interface is configured with `protocol-mode` set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

Add VLAN table entries to allow frames with specific VLAN IDs between ports:

```
/interface ethernet switch vlan
add ports=ether2,ether3 switch=switch1 vlan-id=200
add ports=ether2,ether4 switch=switch1 vlan-id=300
add ports=ether2,ether5 switch=switch1 vlan-id=400
```

Assign `vlan-mode` and `vlan-header` mode for each port and also `default-vlan-id` on ingress for each access port:

```
/interface ethernet switch port
set ether2 vlan-mode=secure vlan-header=add-if-missing
set ether3 vlan-mode=secure vlan-header=always-strip default-vlan-id=200
set ether4 vlan-mode=secure vlan-header=always-strip default-vlan-id=300
set ether5 vlan-mode=secure vlan-header=always-strip default-vlan-id=400
```

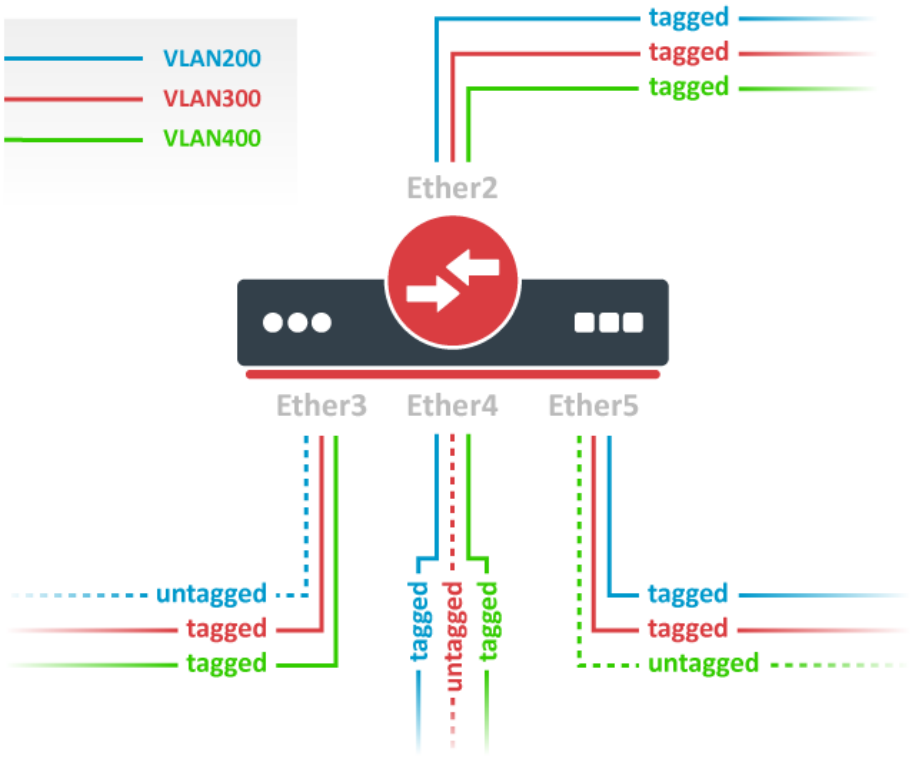
- Setting `vlan-mode=secure` ensures strict use of the VLAN table.
- Setting `vlan-header=always-strip` for access ports removes the VLAN header from the frame when it leaves the switch chip.

- Setting `vlan-header=add-if-missing` for trunk port adds VLAN header to untagged frames.
- `default-vlan-id` specifies what VLAN ID is added for untagged ingress traffic of the access port.

> ⚠ On **QCA8337** and **Atheros8327** switch chips, a default `vlan-header=leave-as-is` property should be used. The switch chip will determine which ports are access ports by using the `default-vlan-id` property. The `default-vlan-id` should only be used on access/hybrid ports to specify which VLAN the untagged ingress traffic is assigned to.

VLAN Example 2 (Trunk and Hybrid Ports)

VLAN Hybrid ports can forward both tagged and untagged traffic. This configuration is supported only by some Gigabit switch chips (**QCA8337, Atheros8327**).



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

> ⚠ By default, the bridge interface is configured with `protocol-mode` set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

Add VLAN table entries to allow frames with specific VLAN IDs between ports.

```
/interface ethernet switch vlan
add ports=ether2,ether3,ether4,ether5 switch=switch1 vlan-id=200
add ports=ether2,ether3,ether4,ether5 switch=switch1 vlan-id=300
add ports=ether2,ether3,ether4,ether5 switch=switch1 vlan-id=400
```

In the switch port menu set `vlan-mode` on all ports and also `default-vlan-id` on planned hybrid ports:

```
/interface ethernet switch port
set ether2 vlan-mode=secure vlan-header=leave-as-is
set ether3 vlan-mode=secure vlan-header=leave-as-is default-vlan-id=200
set ether4 vlan-mode=secure vlan-header=leave-as-is default-vlan-id=300
set ether5 vlan-mode=secure vlan-header=leave-as-is default-vlan-id=400
```

- `vlan-mode=secure` will ensure strict use of the VLAN table.
- `default-vlan-id` will define VLAN for untagged ingress traffic on the port.
- In QCA8337 and Atheros8327 chips when `vlan-mode=secur`e is used, it ignores switch port `vlan-header` options. VLAN table entries handle all the egress tagging/untagging and works as `vlan-header=leave-as-is` on all ports. It means what comes in tagged, goes out tagged as well, only `default-vlan-id` frames are untagged at the egress port.

Management access configuration

In these examples, there will be shown examples for multiple scenarios, but each of these scenarios requires you to have switched ports. Below you can find how to switch multiple ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add interface=ether1 bridge=bridge1 hw=yes
add interface=ether2 bridge=bridge1 hw=yes
```


By default, the bridge interface is configured with `protocol-mode` set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

In these examples, it will be assumed that **ether1** is the trunk port and **ether2** is the access port, for configuration as the following:

```
/interface ethernet switch port
set ether1 vlan-header=add-if-missing
set ether2 default-vlan-id=100 vlan-header=always-strip
/interface ethernet switch vlan
add ports=ether1,ether2,switch1-cpu switch=switch1 vlan-id=100
```

Tagged

To make the device accessible only from a certain VLAN, you need to create a new VLAN interface on the bridge interface and assign an IP address to it:

```
/interface vlan
add name=MGMT vlan-id=99 interface=bridge1
/ip address
add address=192.168.99.1/24 interface=MGMT
```

Specify from which interfaces it is allowed to access the device:

```
/interface ethernet switch vlan
add ports=ether1,switch1-cpu switch=switch1 vlan-id=99
```

⚠ Only specify trunk ports in this VLAN table entry, it is not possible to allow access to the CPU with tagged traffic through an access port since the access port will tag all ingress traffic with the specified `default-vlan-id` value.

When the VLAN table is configured, you can enable `vlan-mode=secure` to limit access to the CPU:

```
/interface ethernet switch port
set ether1 vlan-header=add-if-missing vlan-mode=secure
set ether2 default-vlan-id=100 vlan-header=always-strip vlan-mode=secure
set switch1-cpu vlan-header=leave-as-is vlan-mode=secure
```

Untagged

To make the device accessible from the access port, create a VLAN interface with the same VLAN ID as set in `default-vlan-id`, for example, VLAN 100, and add an IP address to it:

```
/interface vlan
add name=VLAN100 vlan-id=100 interface=bridge1
/ip address
add address=192.168.100.1/24 interface=VLAN100
```

Specify which access (untagged) ports are allowed to access the CPU:

```
/interface ethernet switch vlan
add ports=ether1,ether2,switch1-cpu switch=switch1 vlan-id=100
```

⛔ Most commonly an access (untagged) port is accompanied by a trunk (tagged) port. In case of untagged access to the CPU, you are forced to specify both the access port and the trunk port, this gives access to the CPU from the trunk port as well. Not always this is desired and a Firewall might be required on top of VLAN filtering.

When the VLAN table is configured, you can enable `vlan-mode=secure` to limit access to the CPU:

```
/interface ethernet switch port
set ether1 vlan-header=add-if-missing vlan-mode=secure
set ether2 default-vlan-id=100 vlan-header=always-strip vlan-mode=secure
set switch1-cpu vlan-header=leave-as-is vlan-mode=secure
```

⚠ To setup the management port using untagged traffic on a device with the **Atheros7240** switch chip, you will need to set `vlan-header=add-if-missing` for the CPU port.

Untagged from tagged port

It is possible to allow access to the device from the trunk (tagged) port with untagged traffic. To do so, assign an IP address on the bridge interface:

```
/ip address
add address=10.0.0.1/24 interface=bridge1
```

Specify which ports are allowed to access the CPU. Use `vlan-id` that is used in `default-vlan-id` for switch-cpu and trunk ports, by default it is set to 0 or 1.

```
/interface ethernet switch vlan
add ports=ether1,switch1-cpu switch=switch1 vlan-id=1
```

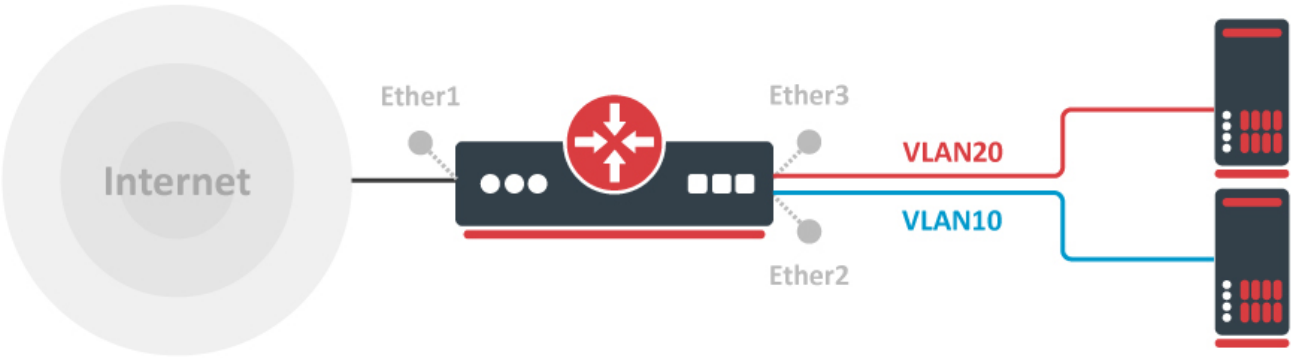
When the VLAN table is configured, you can enable `vlan-mode=secure` to limit access to the CPU:

```
/interface ethernet switch port
set ether1 default-vlan-id=1 vlan-header=add-if-missing vlan-mode=secure
set switch1-cpu default-vlan-id=1 vlan-header=leave-as-is vlan-mode=secure
```

⚠ This configuration example is not possible for devices with the **Atheros8316** and **Atheros7240** switch chips. For devices with **QCA8337** and **Atheros8327** switch chips, it is possible to use any other `default-vlan-id` as long as it stays the same on switch-cpu and trunk ports. For devices with **Atheros8227** switch chip only `default-vlan-id=0` can be used and the trunk port must use `vlan-header=leave-as-is`.

Inter-VLAN routing

Many MikroTik's devices come with a built-in switch chip that can be used to greatly improve overall throughput when configured properly. Devices with a switch chip can be used as a router and a switch at the same time, this gives you the possibility to use a single device instead of multiple devices for your network.



For this type of setup to work, you must switch all required ports together

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
```

Create a VLAN interface for each VLAN ID and assign an IP address to it:

```
/interface vlan
add interface=bridge1 name=VLAN10 vlan-id=10
add interface=bridge1 name=VLAN20 vlan-id=20
/ip address
add address=192.168.10.1/24 interface=VLAN10
add address=192.168.20.1/24 interface=VLAN20
```

Setup a DHCP Server for each VLAN:

```
/ip pool
add name=POOL10 ranges=192.168.10.100-192.168.10.200
add name=POOL20 ranges=192.168.20.100-192.168.20.200
/ip dhcp-server
add address-pool=POOL10 disabled=no interface=VLAN10 name=DHCP10
add address-pool=POOL20 disabled=no interface=VLAN20 name=DHCP20
/ip dhcp-server network
add address=192.168.10.0/24 dns-server=8.8.8.8 gateway=192.168.10.1
add address=192.168.20.0/24 dns-server=8.8.8.8 gateway=192.168.20.1
```

Enable NAT on the device:

```
/ip firewall nat
add action=masquerade chain=srcnat out-interface=ether1
```

Add each port to the VLAN table and allow these ports to access the CPU to make DHCP and routing work:

```
/interface ethernet switch vlan
add independent-learning=yes ports=ether2,switch1-cpu switch=switch1 vlan-id=10
add independent-learning=yes ports=ether3,switch1-cpu switch=switch1 vlan-id=20
```

Specify each port to be an access port, and enable secure VLAN mode on each port and on the switch1-cpu port:

```
/interface ethernet switch port
set ether2 default-vlan-id=10 vlan-header=always-strip vlan-mode=secure
set ether3 default-vlan-id=20 vlan-header=always-strip vlan-mode=secure
set switch1-cpu vlan-mode=secure
```


On **QCA8337** and **Atheros8327** switch chips, a default `vlan-header=leave-as-is` property should be used. The switch chip will determine which ports are access ports by using the `default-vlan-id` property. The `default-vlan-id` should only be used on access/hybrid ports to specify which VLAN the untagged ingress traffic is assigned to.

If your device has a switch rule table, then you can limit access between VLANs on a hardware level. As soon as you add an IP address on the VLAN interface you enable inter-VLAN routing, but this can be limited on a hardware level while preserving DHCP Server and other router-related services. To do so, use these ACL rules. With this type of configuration, you can achieve isolated port groups using VLANs.

```
/interface ethernet switch rule
add dst-address=192.168.20.0/24 new-dst-ports="" ports=ether2 switch=switch1
add dst-address=192.168.10.0/24 new-dst-ports="" ports=ether3 switch=switch1
```

See also

- Switch Router
- Basic VLAN Switching
- Bridge Hardware Offloading
- Spanning Tree Protocol
- DHCP Snooping and Option 82
- MTU on RouterBOARD
- Layer2 misconfiguration
- Master-port

VLAN

Summary

Standards: `IEEE 802.1Q, IEEE 802.1ad`

Virtual Local Area Network (VLAN) is a Layer 2 method that allows multiple Virtual LANs on a single physical interface (ethernet, wireless, etc.), giving the ability to segregate LANs efficiently.

You can use MikroTik RouterOS (as well as Cisco IOS, Linux, and other router systems) to mark these packets as well as to accept and route marked ones.

As VLAN works on OSI Layer 2, it can be used just like any other network interface without any restrictions. VLAN successfully passes through regular Ethernet bridges.

You can also transport VLANs over wireless links and put multiple VLAN interfaces on a single wireless interface. Note that as VLAN is not a full tunnel protocol (i.e., it does not have additional fields to transport MAC addresses of sender and recipient), the same limitation applies to bridging over VLAN as to bridging plain wireless interfaces. In other words, while wireless clients may participate in VLANs put on wireless interfaces, it is not possible to have VLAN put on a wireless interface in station mode bridged with any other interface.

802.1Q

The most commonly used protocol for Virtual LANs (VLANs) is IEEE 802.1Q. It is a standardized encapsulation protocol that defines how to insert a four-byte VLAN identifier into the Ethernet header.

Each VLAN is treated as a separate subnet. It means that by default, a host in a specific VLAN cannot communicate with a host that is a member of another VLAN, although they are connected to the same switch. So if you want inter-VLAN communication you need a router. RouterOS supports up to 4095 VLAN interfaces, each with a unique VLAN ID, per interface. VLAN priorities may also be used and manipulated.

When the VLAN extends over more than one switch, the inter-switch link has to become a 'trunk', where packets are tagged to indicate which VLAN they belong to. A trunk carries the traffic of multiple VLANs; it is like a point-to-point link that carries tagged packets between switches or between a switch and router.

> The IEEE 802.1Q standard has reserved VLAN IDs with special use cases, the following VLAN IDs should not be used in generic VLAN setups: 0, 1, 4095

Q-in-Q

Original 802.1Q allows only one VLAN header, Q-in-Q on the other hand allows two or more VLAN headers. In RouterOS, Q-in-Q can be configured by adding one VLAN interface over another. Example:

```
/interface vlan
add name=vlan1 vlan-id=11 interface=ether1
add name=vlan2 vlan-id=12 interface=vlan1
```

If any packet is sent over the 'vlan2' interface, two VLAN tags will be added to the Ethernet header - '11' and '12'.

Properties

Property	Description
arp (*disabled* / *enabled* / *local-proxy-arp* / *proxy-arp* / *reply-only*; Default: **enabled**)	Address Resolution Protocol setting • `disabled` - the interface will not use ARP • `enabled` - the interface will use ARP • `local-proxy-arp` - the router performs proxy ARP on the interface and sends replies to the same interface • `proxy-arp` - the router performs proxy ARP on the interface and sends replies to other interfaces • `reply-only` - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the IP/ARP table. No dynamic entries will be automatically stored in the IP/ARP table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*auto* / *integer*; Default: **auto**)	How long the ARP record is kept in the ARP table after no packets are received from IP. Value `auto` equals to the value of `arp-timeout` in IP/Settings, default is 30s.
disabled (*yes* / *no*; Default: **no**)	Changes whether the bridge is disabled.
interface (*name*; Default:)	Name of the interface on top of which VLAN will work
mvrp (*yes* / *no*; Default: **no**)	Specifies whether this VLAN should declare its attributes through Multiple VLAN Registration Protocol (MVRP) as an applicant. It can be used to register the VLAN with connected bridges that support MVRP. This property only has an effect when `use-service-tag` is disabled.
mtu (*integer*; Default: **1500**)	Layer3 Maximum transmission unit
name (*string*; Default:)	Interface name
use-service-tag (*yes* / *no*; Default:)	IEEE 802.1ad compatible Service Tag
vlan-id (*integer: 4095*; Default: **1**)	Virtual LAN identifier or tag that is used to distinguish VLANs. Must be equal for all computers that belong to the same VLAN.

MTU should be set to 1500 bytes same as on Ethernet interfaces. But this may not work with some Ethernet cards that do not support receiving /transmitting of full-size Ethernet packets with VLAN header added (1500 bytes data + 4 bytes VLAN header + 14 bytes Ethernet header). In this situation, MTU 1496 can be used, but note that this will cause packet fragmentation if larger packets have to be sent over the interface. At the same time remember that MTU 1496 may cause problems if path MTU discovery is not working properly between source and destination.

Setup examples

Layer2 VLAN examples

There are multiple possible configurations that you can use, but each configuration type is designed for a special set of devices since some configuration methods will give you the benefits of the built-in switch chip and gain larger throughput. Check the Basic VLAN switching guide to see which configuration to use for each type of device to gain maximum possible throughput and compatibility, the guide shows how to set up a very basic VLAN trunk/access port configuration.

There are some other ways to set up VLAN tagging or VLAN switching, but the recommended way is to use Bridge VLAN Filtering. Make sure you have not used any known Layer2 misconfigurations.

Layer3 VLAN examples

Simple VLAN routing

Let us assume that we have several MikroTik routers connected to a hub. Remember that a hub is an OSI physical layer device (if there is a hub between routers, then from the L3 point of view it is the same as an Ethernet cable connection between them). For simplification assume that all routers are connected to the hub using the ether1 interface and have assigned IP addresses as illustrated in the figure below. Then on each of them the VLAN interface is created.

Configuration for R2 and R4 is shown below:

R2:

```
[admin@MikroTik] /interface vlan> add name=VLAN2 vlan-id=2 interface=ether1 disabled=no

[admin@MikroTik] /interface vlan> print
Flags: X - disabled, R - running, S - slave
 #    NAME                 MTU   ARP        VLAN-ID INTERFACE
0 R  VLAN2                1500  enabled    2       ether1
```

R4:

```
[admin@MikroTik] /interface vlan> add name=VLAN2 vlan-id=2 interface=ether1 disabled=no

[admin@MikroTik] /interface vlan> print
Flags: X - disabled, R - running, S - slave
 #    NAME                 MTU   ARP        VLAN-ID INTERFACE
0 R  VLAN2                1500  enabled    2       ether1
```

The next step is to assign IP addresses to the VLAN interfaces.

R2:

```
 [admin@MikroTik] ip address> add address=10.10.10.3/24 interface=VLAN2
 [admin@MikroTik] ip address> print
 Flags: X - disabled, I - invalid, D - dynamic
   #   ADDRESS            NETWORK         BROADCAST       INTERFACE
   0   10.0.1.4/24        10.0.1.0        10.0.1.255      ether1
   1   10.20.0.1/24       10.20.0.0       10.20.0.255     pc1
   2   10.10.10.3/24      10.10.10.0      10.10.10.255    vlan2

 [admin@MikroTik] ip address>
```

R4:

```
 [admin@MikroTik] ip address> add address=10.10.10.5/24 interface=VLAN2
 [admin@MikroTik] ip address> print
 Flags: X - disabled, I - invalid, D - dynamic
   #   ADDRESS            NETWORK         BROADCAST       INTERFACE
   0   10.0.1.5/24        10.0.1.0        10.0.1.255      ether1
   1   10.30.0.1/24       10.30.0.0       10.30.0.255     pc2
   2   10.10.10.5/24      10.10.10.0      10.10.10.255    vlan2

[admin@MikroTik] ip address>
```

At this point, it should be possible to ping router R4 from router R2 and vice versa:

```
"Ping from R2 to R4:"

[admin@MikroTik] ip address> /ping 10.10.10.5

10.10.10.5 64 byte ping: ttl=255 time=4 ms

10.10.10.5 64 byte ping: ttl=255 time=1 ms

2 packets transmitted, 2 packets received, 0% packet loss

round-trip min/avg/max = 1/2.5/4 ms


"From R4 to R2:"

[admin@MikroTik] ip address> /ping 10.10.10.3
10.10.10.3 64 byte ping: ttl=255 time=6 ms
10.10.10.3 64 byte ping: ttl=255 time=1 ms
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1/3.5/6 ms
```

To make sure if the VLAN setup is working properly, try to ping R1 from R2. If pings are timing out then VLANs are successfully isolated.

```
"From R2 to R1:"

[admin@MikroTik] ip address> /ping 10.10.10.2
10.10.10.2 ping timeout
10.10.10.2 ping timeout
3 packets transmitted, 0 packets received, 100% packet loss
```

InterVLAN routing

If separate VLANs are implemented on a switch, then a router is required to provide communication between VLANs. A switch works at OSI layer 2 so it uses only the Ethernet header to forward and does not check the IP header. For this reason, we must use the router that is working as a gateway for each VLAN. Without a router, a host is unable to communicate outside of its own VLAN. The routing process between VLANs described above is called inter-VLAN communication.

To illustrate inter-VLAN communication, we will create a trunk that will carry traffic from three VLANs (VLAN2 and VLAN3, VLAN4) across a single link between a Mikrotik router and a manageable switch that supports VLAN trunking.

Each VLAN has its own separate subnet (broadcast domain) as we see in the figure above:

- VLAN 2 – 10.10.20.0/24;
- VLAN 3 – 10.10.30.0/24;
- VLAN 4 – 10.10.40.0./24.

VLAN configuration on most switches is straightforward, we need to define which ports are members of the VLANs and define a 'trunk' port that can carry tagged frames between the switch and the router.

Create VLAN interfaces:

```
/interface vlan
add name=VLAN2 vlan-id=2 interface=ether1 disabled=no
add name=VLAN3 vlan-id=3 interface=ether1 disabled=no
add name=VLAN4 vlan-id=4 interface=ether1 disabled=no
```

Add IP addresses to VLANs:

```
/ip address
add address=10.10.20.1/24 interface=VLAN2
add address=10.10.30.1/24 interface=VLAN3
add address=10.10.40.1/24 interface=VLAN4
```

RouterOS /32 and IP unnumbered addresses

In RouterOS, to create a point-to-point tunnel with addresses you have to use the address with a network mask of '/32' that effectively brings you the same features as some vendors unnumbered IP address.

There are 2 routers RouterA and RouterB where each is part of networks 10.22.0.0/24 and 10.23.0.0/24 respectively and to connect these routers using VLANs as a carrier with the following configuration:

RouterA:

```
/ip address add address=10.22.0.1/24 interface=ether1
/interface vlan add interface=ether2 vlan-id=1 name=vlan1
/ip address add address=10.22.0.1/32 interface=vlan1 network=10.23.0.1
/ip route add gateway=10.23.0.1 dst-address=10.23.0.0/24
```

RouterB:

```
/ip address add address=10.23.0.1/24 interface=ether1
/interface vlan add interface=ether2 vlan-id=1 name=vlan1
/ip address add address=10.23.0.1/32 interface=vlan1 network=10.22.0.1
/ip route add gateway=10.22.0.1 dst-address=10.22.0.0/24
```

VXLAN

Introduction

Virtual eXtensible Local Area Network (VXLAN) is a tunneling protocol designed to solve the problem of limited VLAN IDs (4096) in IEEE 802.1Q, and it is described by IETF RFC 7348. With VXLAN the size of the identifier is expanded to 24 bits (16777216). It creates a Layer 2 overlay scheme on a Layer 3 network and the protocol runs over UDP. RouterOS VXLAN interface supports IPv4 or IPv6 (since version 7.6), but dual-stack is not supported.

> VXLAN creates a 50-byte overhead for IPv4 and a 70-byte overhead for IPv6. When configuring VXLAN, it is recommended to ensure that the size of the encapsulated Ethernet frame does not exceed the MTU of the underlying network, by configuring the MTU accordingly or by limiting the size of the Ethernet frames.

Only devices within the same VXLAN segment can communicate with each other. Each VXLAN segment is identified through a 24-bit segment ID, termed the VXLAN Network Identifier (VNI). Unlike most tunnels, a VXLAN is a 1-to-N network, not just point-to-point. VXLAN endpoints, which terminate VXLAN tunnels are known as VXLAN tunnel endpoints (VTEPs). RouterOS only supports statically configured remote VTEPs. When unicast traffic needs to be sent over VXLAN, a device can learn the IP address of the other endpoint dynamically in a manner similar to a learning bridge, and forward traffic only to the necessary VTEP. For traffic that needs to be flooded (broadcast, unknown-unicast, and multicast) to all VTEPs on the same segment, VXLAN can use multicast or unicast with head-end replication to send one replica for every remote VTEP.

Configuration options

This section describes the VXLAN interface and VTEP configuration options.

Sub-menu: `/interface vxlan`

Property	Description
allow-fast-path (*yes* | *no*; Default: **yes**)	Whether to allow Fast Path processing. Fragmented and flooded packets over VXLAN are redirected via a slow path. Fast Path is disabled for VXLAN interface that uses IPv6 VTEP version or VRF. The setting is available since RouterOS version 7.8.
arp (*disabled* | *enabled* | *local-proxy-arp* | *proxy-arp* | *reply-only*; Default: **enabled**)	Address Resolution Protocol setting • `disabled` - the interface will not use ARP • `enabled` - the interface will use ARP • `local-proxy-arp` - the router performs proxy ARP on the interface and sends replies to the same interface • `proxy-arp` - the router performs proxy ARP on the interface and sends replies to other interfaces • `reply-only` - the interface will only reply to requests originating from matching IP address/MAC address combinations which are entered as static entries in the IP/ARP table. No dynamic entries will be automatically stored in the IP/ARP table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*auto* | *integer*; Default: **auto**)	How long the ARP record is kept in the ARP table after no packets are received from IP. Value `auto` equals to the value of `arp-timeout` in IP/Settings, default is the 30s.
comment (*string* ; Default:)	Short description of the interface.
disabled (*yes* | *no*; Default: **no**)	Changes whether the interface is disabled.

dont-fragment (*disabled* | *enabled* | *inherit*; Default: **disabled**)	The Don't Fragment (DF) flag controls whether a packet can be broken into smaller packets, called fragments, before being sent over a network. When configuring VXLAN, this setting determines the presence of the DF flag on the outer IPv4 header and can control packet fragmentation if the encapsulated packet exceeds the outgoing interface MTU. This setting has three options: • `disabled` - the DF flag is not set on the outer IPv4 header, which means that packets can be fragmented if they are too large to be sent over the outgoing interface. This also allows packet fragmentation when VXLAN uses IPv6 underlay. • `enabled` - the DF flag is always set on the outer IPv4 header, which means that packets will not be fragmented and will be dropped if they exceed the outgoing interface's MTU. This also avoids packet fragmentation when VXLAN uses IPv6 underlay. • `inherit` - The DF flag on the outer IPv4 header is based on the inner IPv4 DF flag. If the inner IPv4 header has the DF flag set, the outer IPv4 header will also have it set. If the packet exceeds the outgoing interface's MTU and DF is set, it will be dropped. If the inner packet is non-IP, the outer IPv4 header will not have the DF flag set and packets can be fragmented. If the inner packet is IPv6, the outer IPv4 header will always set the DF flag and packets cannot be fragmented. Note that when VXLAN uses IPv6 underlay, this setting does not have any effect and is treated the same as `disabled`. The setting is available since RouterOS version 7.8.
group (*IPv4* | *IPv6*; Default:)	When specified, a multicast group address can be used to forward broadcast, unknown-unicast, and multicast traffic between VTEPs. This property requires specifying the `interface` setting. The interface will use IGMP or MLD to join the specified multicast group, make sure to add the necessary PIM and IGMP/MDL configuration. When this property is set, the `vteps-ip-version` automatically gets updated to the used multicast IP version.
interface (*name*; Default:)	Interface name used for multicast forwarding. This property requires specifying the `group` setting.
local-address (*IPv4* | *IPv6*; Default:)	Specifies the local source address for the VXLAN interface. If not set, one IP address of the egress interface will be selected as a source address for VXLAN packets. When the property is set, the `vteps-ip-version` automatically gets updated to the used local IP version. The setting is available since RouterOS version 7.7.
mac-address (*MAC*; Default:)	Static MAC address of the interface. A randomly generated MAC address will be assigned when not specified.
max-fdb-size (*integer: 1..65535*; Default: **4096**)	Limits the maximum number of MAC addresses that VXLAN can store in the forwarding database (FDB).
mtu (*integer*; Default: **1500**)	For the maximum transmission unit, the VXLAN interface will set MTU to 1500 by default. The `l2mtu` will be set automatically according to the associated `interface` (subtracting 50 bytes corresponding to the VXLAN header). If no interface is specified, the `l2mtu` value of 65535 is used. The `l2mtu` cannot be changed.
name (*text*; Default: **vxlan1**)	Name of the interface.
port (*integer: 1..65535*; Default: **8472**)	Used UDP port number.
vni (*integer: 1..16777216*; Default:)	VXLAN Network Identifier (VNI).
vrf (*name*; Default: **main**)	Set VRF for the VXLAN interface on which the VTEPs listen and make connections. VRF is not supported when using `interface` and multicast `group` settings. The same UDP `port` cannot be used in multiple routing tables at the same time. The setting is available since RouterOS version 7.7.
vteps-ip-version (*ipv4* | *ipv6*; Default: **ipv4**)	Used IP protocol version for statically configured VTEPs. The RouterOS VXLAN interface does not support dual-stack, any configured remote VTEPs with the opposite IP version will be ignored. When multicast `group` or `local-address` properties are set, the `vteps-ip-version` automatically gets updated to the used IP version. The setting is available since RouterOS version 7.6.

Sub-menu: `/interface vxlan vteps`

Property	Description
interface (*name*; Default:)	Name of the VXLAN interface.
port (*integer: 1..65535*; Default: **8472**)	Used UDP port number.

remote-ip (*IPv4* | *IPv6*; Default:)	The IPv4 or IPv6 destination address of remote VTEP.

Forwarding table

Since RouterOS version 7.9, it is possible to monitor the learned MAC addresses from remote VTEPs.

Sub-menu: `/interface vxlan fdb`

Property	Description
interface (*read-only: name*)	Name of the VXLAN interface.
mac-address (*read-only: MAC address*)	MAC address.
remote-ip (*read-only: IPv4* | *IPv6 address*)	The IPv4 or IPv6 destination address of remote VTEP.

```
[admin@MikroTik] > /interface vxlan fdb print
 0 remote-ip=2001::2 mac-address=56:FF:AA:1A:72:33 interface=vxlan1

 1 remote-ip=2002::2 mac-address=AE:EC:C4:12:8B:B9 interface=vxlan1

 2 remote-ip=192.168.10.20 mac-address=FE:AF:58:31:A7:B6 interface=vxlan2
```

Configuration example

This configuration example creates a single VXLAN tunnel between two statically configured VTEP endpoints.

First, create VXLAN interfaces on both routers.

```
/interface vxlan
add name=vxlan1 port=8472 vni=10
```

Then configure VTEPs on both routers with respective IPv4 destination addresses. Both devices should have an active route toward the destination address.

```
# Router1
/interface vxlan vteps
add interface=vxlan1 remote-ip=192.168.10.10

# Router2
/interface vxlan vteps
add interface=vxlan1 remote-ip=192.168.20.20
```

The configuration is complete. It is possible to include the VXLAN interface into a bridge with other Ethernet interfaces.

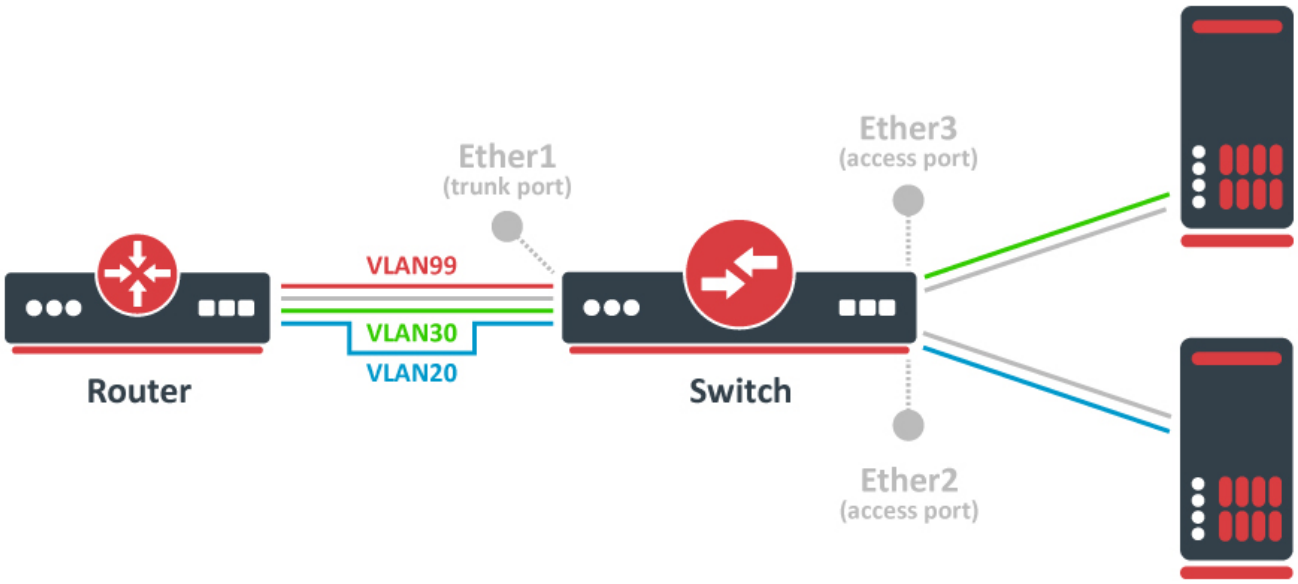
Bridging and Switching Case Studies

IN THIS SECTION

Basic VLAN switching

Introduction

Many MikroTik devices come with built-in switch chips that usually have an option to do VLAN switching on a hardware level, this means that you can achieve wire-speed performance using VLANs if a proper configuration method is used. The configuration method changes across different models, this guide will focus on setting up a basic trunk/access port with a management port from the trunk port using different devices with the right configuration to achieve the best performance and to fully utilize the available hardware components.



CRS3xx, CRS5xx series switches, CCR2116, CCR2216 and RTL8367, 88E6393X, 88E6191X, 88E6190, MT7621 and MT7531 switch chips

```
/interface bridge
add name=bridge1 frame-types=admit-only-vlan-tagged
/interface bridge port
add bridge=bridge1 interface=ether1 frame-types=admit-only-vlan-tagged
add bridge=bridge1 interface=ether2 pvid=20 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether3 pvid=30 frame-types=admit-only-untagged-and-priority-tagged
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=20
add bridge=bridge1 tagged=ether1 vlan-ids=30
add bridge=bridge1 tagged=ether1,bridge1 vlan-ids=99
/interface vlan
add interface=bridge1 vlan-id=99 name=MGMT
/ip address
add address=192.168.99.1/24 interface=MGMT
/interface bridge
set bridge1 vlan-filtering=yes
```

More detailed examples can be found here.

⚠ RTL8367, 88E6393X, 88E6191X, 88E6190, MT7621 and MT7531 switch chips can use HW offloaded vlan-filtering since RouterOS v7.

⚠ Bridge ports with `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged` will be automatically added as untagged ports for the `pvid` VLAN.

CRS1xx/CRS2xx series switches

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
/interface ethernet switch ingress-vlan-translation
add ports=ether2 customer-vid=0 new-customer-vid=20
add ports=ether3 customer-vid=0 new-customer-vid=30
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether1 vlan-id=20
add tagged-ports=ether1 vlan-id=30
add tagged-ports=ether1,switch1-cpu vlan-id=99
/interface ethernet switch vlan
add ports=ether1,ether2 vlan-id=20
add ports=ether1,ether3 vlan-id=30
add ports=ether1,switch1-cpu vlan-id=99
/interface vlan
add interface=bridge1 vlan-id=99 name=MGMT
/ip address
add address=192.168.99.1/24 interface=MGMT
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether1,ether2,ether3
```

More detailed examples can be found here.

Other devices with a built-in switch chip

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
/interface ethernet switch vlan
add ports=ether1,ether2 switch=switch1 vlan-id=20
add ports=ether1,ether3 switch=switch1 vlan-id=30
add ports=ether1,switch1-cpu switch=switch1 vlan-id=99
/interface vlan
add interface=bridge1 vlan-id=99 name=MGMT
/ip address
add address=192.168.99.1/24 interface=MGMT
/interface ethernet switch port
set ether1 vlan-mode=secure vlan-header=add-if-missing
set ether2 vlan-mode=secure vlan-header=always-strip default-vlan-id=20
set ether3 vlan-mode=secure vlan-header=always-strip default-vlan-id=30
set switch1-cpu vlan-header=leave-as-is vlan-mode=secure
```

More detailed examples can be found here.

Not all devices with a switch chip are capable of VLAN switching on a hardware level, check the supported features for each switch chip, the compatibility table can be found here. If a device has `VLAN table` support, then it is capable of VLAN switching using the built-in switch chip. You can check the device's switch chip either in the provided link or by using `/interface ethernet switch print`

On **QCA8337** and **Atheros8327** switch chips, a default `vlan-header=leave-as-is` property should be used. The switch chip will determine which ports are access ports by using the `default-vlan-id` property. The `default-vlan-id` should only be used on access/hybrid ports to specify which VLAN the untagged ingress traffic is assigned to.

This type of configuration should be used on RouterBOARD series devices, this includes RB4xx, RB9xx, RB2011, RB3011, hAP, hEX, cAP, and other devices.

By default, the bridge interface is configured with protocol-mode set to `rstp`. For some devices, this can disable hardware offloading because specific switch chips do not support this feature. See the Bridge Hardware Offloading section with supported features.

For devices that have multiple switch chips (for example, RB2011, RB3011, RB1100), each switch chip is only able to switch VLAN traffic between ports that are on the same switch chip, VLAN filtering will not work on a hardware level between ports that are on different switch chips, this means you should not add all ports to a single bridge if you are intending to use VLAN filtering using the switch chip, VLANs between switch chips will not get filtered. You can connect a single cable between both switch chips to work around this hardware limitation, another option is to use Bridge VLAN Filtering, but it disables hardware offloading (and lowers the total throughput).

Other devices without a built-in switch chip

It is possible to do VLAN filtering using the CPU, there are multiple ways to do it, but it is highly recommended to use bridge VLAN filtering.

```
/interface bridge
add name=bridge1 frame-types=admit-only-vlan-tagged
/interface bridge port
add bridge=bridge1 interface=ether1 frame-types=admit-only-vlan-tagged
add bridge=bridge1 interface=ether2 pvid=20 frame-types=admit-only-untagged-and-priority-tagged
add bridge=bridge1 interface=ether3 pvid=30 frame-types=admit-only-untagged-and-priority-tagged
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=20
add bridge=bridge1 tagged=ether1 vlan-ids=30
add bridge=bridge1 tagged=ether1,bridge1 vlan-ids=99
/interface vlan
add interface=bridge1 vlan-id=99 name=MGMT
/ip address
add address=192.168.99.1/24 interface=MGMT
/interface bridge
set bridge1 vlan-filtering=yes
```

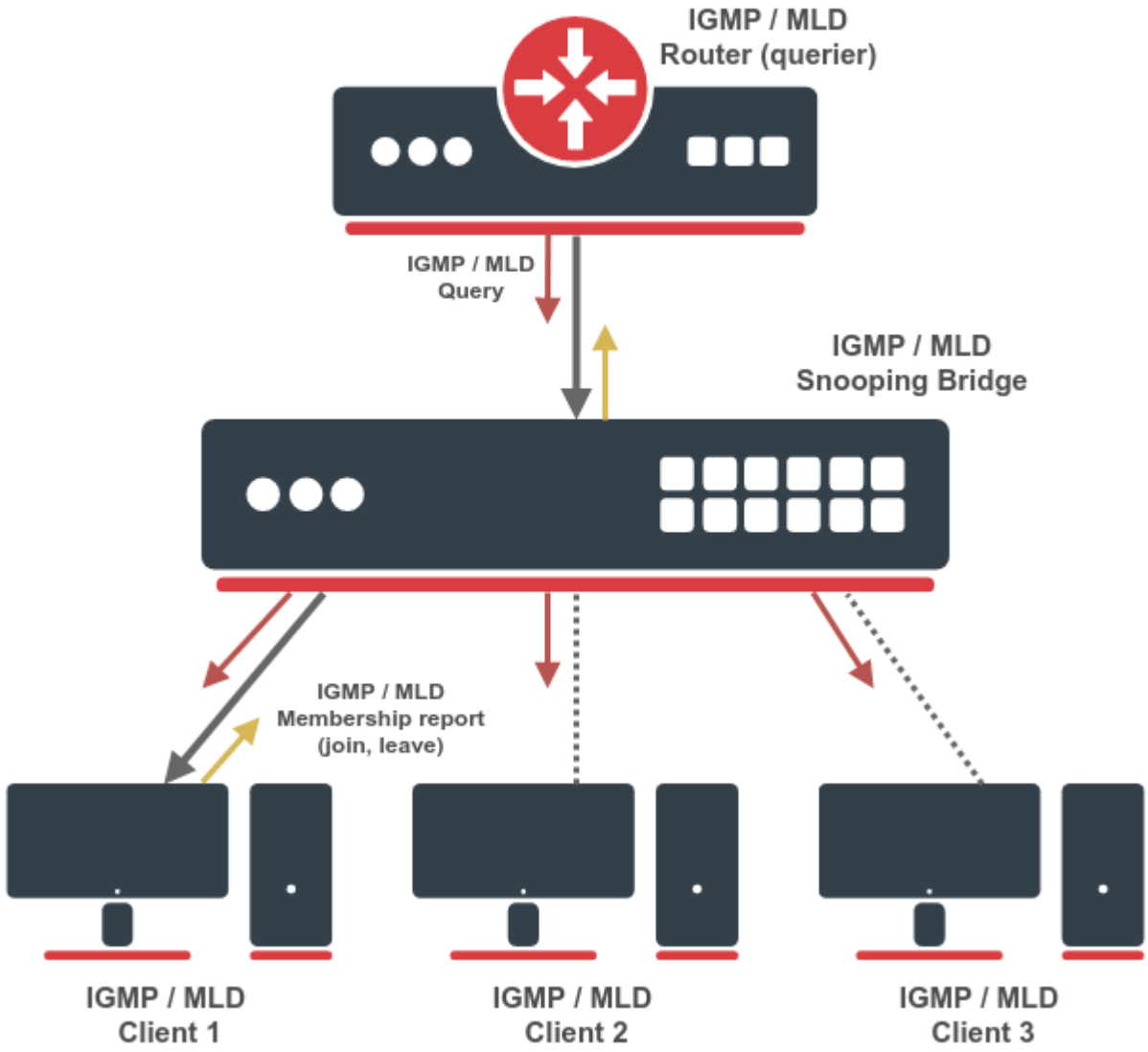
More detailed examples can be found here.

Bridge IGMP/MLD snooping

Introduction

IGMP (Internet Group Management Protocol) and MLD (Multicast Listener Discovery) snooping allow the bridge to listen to IGMP/MLD communication and make forwarding decisions for multicast traffic based on the received information. By default, bridges are flooding multicast traffic to all bridge ports just like broadcast traffic, which might not always be the best scenario (e.g. for multicast video traffic or SDVoE applications). The IGMP/MLD snooping tries to solve the problem by forwarding the multicast traffic only to ports where clients are subscribed to, see an IGMP/MLD network concept below. RouterOS bridge can process IGMP v1/v2/v3 and MLD v1/v2 packets. The implemented bridge IGMP/MLD snooping is based on RFC4541, and IGMP/MLD protocols are specified on RFC1112 (IGMPv1) RFC2236 (IGMPv2), RFC3376 (IGMPv3), RFC2710 (MLDv1), RFC3810 (MLDv2).

Source-specific multicast forwarding is not supported for IGMP v3 and MLD v2.



The bridge will process the IGMP/MLD messages only when `igmp-snooping` is enabled. Additionally, the bridge should have an active IPv6 address to process MLD packets. At first, the bridge does not restrict the multicast traffic and all multicast packets get flooded. Once IGMP/MLD querier is detected by receiving an IGMP/MLD query message (the query message can be received by an external multicast router or locally by bridge interface with enabled `multicast-querier`), only then the bridge will start to restrict unknown IP multicast traffic and forward the known multicast from the multicast database (MDB). The IGMP and MLD querier detection is independent, which means that detecting only IGMP querier will not affect IPv6 multicast forwarding and vice versa. The querier detection also does not restrict the forwarding of non-IP and link-local multicast groups, like 224.0.0.0/24 and ff02::1.

> CRS3xx series devices with Marvell-98DX3236, Marvell-98DX224S or Marvell-98DX226S switch chips are not able to distinguish non-IP/IPv4 /IPv6 multicast packets once IGMP or MLD querier is detected. It means that the switch will stop forwarding all unknown non-IP/IPv4/IPv6 multicast traffic when the querier is detected. This does not apply to certain link-local multicast address ranges, like 224.0.0.0/24 or ff02::1.

Configuration options

This section describes the IGMP and MLD snooping bridge configuration options.

Sub-menu: `/interface bridge`

Property	Description

igmp-snooping (*yes* | *no*; Default: **no**)	Enables IGMP and MLD snooping.
igmp-version (*2* | *3*; Default: **2**)	Selects the IGMP version in which IGMP membership queries will be generated when the bridge interface is acting as an IGMP querier. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
last-member-interval (*time*; Default: **1s**)	When the last client on the bridge port unsubscribes to a multicast group and the bridge is acting as an active querier, the bridge will send group-specific IGMP/MLD query, to make sure that no other client is still subscribed. The setting changes the response time for these queries. In case no membership reports are received in a certain time period (`last-member-interval` * `last-member-query-count`), the multicast group is removed from the multicast database (MDB). If the bridge port is configured with fast-leave, the multicast group is removed right away without sending any queries. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
last-member-query-count (*integer: 0.. 4294967295*; Default: **2**)	How many times should `last-member-interval` pass until the IGMP/MLD snooping bridge stops forwarding a certain multicast stream. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
membership-interval (*time*; Default: **4m20s**)	The amount of time after an entry in the Multicast Database (MDB) is removed if no IGMP/MLD membership reports are received on a bridge port. This property only has an effect when `igmp-snooping` is set to `yes`.
mld-version (*1* | *2*; Default: **1**)	Selects the MLD version in which MLD membership queries will be generated, when the bridge interface is acting as an MLD querier. This property only has an effect when the bridge has an active IPv6 address, `igmp-snooping` and `multicast-querier` is set to `yes`.
multicast-querier (*yes* | *no*; Default: **no**)	Multicast querier generates periodic IGMP/MLD general membership queries to which all IGMP/MLD capable devices respond with an IGMP/MLD membership report, usually a PIM (multicast) router or IGMP proxy generates these queries. By using this property you can make an IGMP/MLD snooping enabled bridge to generate IGMP/MLD general membership queries. This property should be used whenever there is no active querier (PIM router or IGMP proxy) in a Layer2 network. Without a multicast querier in a Layer2 network, the Multicast Database (MDB) is not being updated, the learned entries will timeout and IGMP/MLD snooping will not function properly. Only untagged IGMP/MLD general membership queries are generated, IGMP queries are sent with IPv4 0.0.0.0 source address, MLD queries are sent with IPv6 link-local address of the bridge interface. The bridge will not send queries if an external IGMP /MLD querier is detected (see the monitoring values `igmp-querier` and `mld-querier`). This property only has an effect when `igmp-snooping` is set to `yes`.
multicast-router (*disabled* | *permanent* | *temporary-query*; Default: **temporary-query**)	A multicast router port is a port where a multicast router or querier is connected. On this port, unregistered multicast streams and IGMP/MLD membership reports will be sent. This setting changes the state of the multicast router for a bridge interface itself. This property can be used to send IGMP/MLD membership reports and multicast traffic to the bridge interface for further multicast routing or proxying. This property only has an effect when `igmp-snooping` is set to `yes`. • `disabled` - disabled multicast router state on the bridge interface. Unregistered multicast streams and IGMP/MLD membership reports are not sent to the bridge interface regardless of what is configured on the bridge interface. • `permanent` - enabled multicast router state on the bridge interface. Unregistered multicast streams and IGMP/MLD membership reports are sent to the bridge interface itself regardless of what is configured on the bridge interface. • `temporary-query` - automatically detect multicast router state on the bridge interface using IGMP/MLD queries.
querier-interval (*time* ; Default: **4m15s**)	Changes the timeout period for detected querier and multicast-router ports. This property only has an effect when `igmp-snooping` is set to `yes`.
query-interval (*time*; Default: **2m5s**)	Changes the interval on how often IGMP/MLD general membership queries are sent out when the bridge interface is acting as an IGMP/MLD querier. The interval takes place when the last startup query is sent. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
query-response-interval (*time*; Default: **10s**)	The setting changes the response time for general IGMP/MLD queries when the bridge is acting as an IGMP/MLD querier. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.

startup-query-count (*integer: 0..4294967295*; Default: **2**)	Specifies how many times general IGMP/MLD queries must be sent when the bridge interface is enabled or active querier timeouts. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.
startup-query-interval (*time*; Default: **31s250ms**)	Specifies the interval between startup general IGMP/MLD queries. This property only has an effect when `igmp-snooping` and `multicast-querier` is set to `yes`.

Sub-menu: `/interface bridge` port

Property	Description
fast-leave (*yes* | *no*; Default: **no**)	Enables IGMP/MLD fast leave feature on the bridge port. The bridge will stop forwarding multicast traffic to a bridge port when an IGMP/MLD leave message is received. This property only has an effect when `igmp-snooping` is set to `yes`.
multicast-router (*disabled* | *permanent* | *temporary-query*; Default: **temporary-query**)	A multicast router port is a port where a multicast router or querier is connected. On this port, unregistered multicast streams and IGMP/MLD membership reports will be sent. This setting changes the state of the multicast router for bridge ports. This property can be used to send IGMP/MLD membership reports and multicast streams to certain bridge ports for further multicast routing or proxying. This property only has an effect when `igmp-snooping` is set to `yes`. • `disabled` - disabled multicast router state on the bridge port. Unregistered multicast streams and IGMP/MLD membership reports are not sent to the bridge port regardless of what is connected to it. • `permanent` - enabled multicast router state on the bridge port. Unregistered multicast and IGMP/MLD membership reports are sent to the bridge port regardless of what is connected to it. • `temporary-query` - automatically detect multicast router state on the bridge port using IGMP/MLD queries.
unknown-multicast-flood (*yes* | *no*; Default: **yes**)	Changes the multicast flood option on the bridge port, only controls the egress traffic. When enabled, the bridge allows flooding multicast packets to the specified bridge port, but when disabled, the bridge restricts multicast traffic from being flooded to the specified bridge port. The setting affects all multicast traffic, this includes non-IP, IPv4, IPv6, and the link-local multicast ranges (e.g. 224.0.0.0/24 and ff02::1). Note that when `igmp-snooping` is enabled and IGMP/MLD querier is detected, the bridge will automatically restrict unknown IP multicast from being flooded, so the setting is not mandatory for IGMP/MLD snooping setups. When using this setting together with `igmp-snooping`, the only multicast traffic that is allowed on the bridge port is the known multicast from the MDB table.

Sub-menu: `/interface bridge mdb`

Property	Description
bridge (*name*; Default:)	The bridge interface to which the MDB entry is going to be assigned.
disabled (*yes* | *no*; Default: **no**)	Disables or enables static MDB entry.
group (*ipv4* | *ipv6 address*; Default:)	The IPv4 or IPv6 multicast address. Static entries for link-local multicast groups 224.0.0.0/24 and ff02::1 cannot be created, as these packets are always flooded on all ports and VLANs.
ports (*name*; Default:)	The list of bridge ports to which the multicast group will be forwarded.
vid (*integer: 1..4094*; Default:)	The VLAN ID on which the MDB entry will be created, only applies when `vlan-filtering` is enabled. When the VLAN ID is not specified, the entry will work in shared-VLAN mode and dynamically apply on all defined VLAN IDs for particular ports.

Monitoring and troubleshooting

This section describes the IGMP/MLD snooping bridge monitoring and troubleshooting options.

To monitor learned multicast database (MDB) entries, use the `print` command.

Sub-menu: `/interface bridge mdb`

Property	Description
bridge (*read-only: name*)	Shows the bridge interface the entry belongs to.
group (*read-only: ipv4 | ipv6 address*)	Shows a multicast group address.
on-ports (*read-only: name*)	Shows the bridge ports that are subscribed to the certain multicast group.
vid (*read-only: integer*)	Shows the VLAN ID for the multicast group, only applies when `vlan-filtering` is enabled.

```
[admin@MikroTik] /interface bridge mdb print
Flags: D - DYNAMIC
Columns: GROUP, VID, ON-PORTS, BRIDGE
 #   GROUP              VID  ON-PORTS  BRIDGE
 0 D ff02::2              1  bridge1   bridge1
 1 D ff02::6a             1  bridge1   bridge1
 2 D ff02::1:ff00:0       1  bridge1   bridge1
 3 D ff02::1:ff01:6a43    1  bridge1   bridge1
 4 D 229.1.1.1           10  ether2    bridge1
 5 D 229.2.2.2           10  ether3    bridge1
                             ether2
 6 D ff02::2             10  ether5    bridge1
                             ether3
                             ether2
                             ether4
```

To monitor the current status of a bridge interface, use the `monitor` command.

Sub-menu: `/interface bridge`

Property	Description
igmp-querier (*none | interface & IPv4 address*)	Shows a bridge port and source IP address from the detected IGMP querier. Only shows detected external IGMP querier, local bridge IGMP querier (including IGMP proxy and PIM) will not be displayed. Monitoring value appears only when `igmp-snooping` is enabled.
mld-querier (*none | interface & IPv6 address*)	Shows a bridge port and source IPv6 address from the detected MLD querier. Only shows detected external MLD querier, local bridge MLD querier will not be displayed. Monitoring value appears only when `igmp-snooping` is enabled and the bridge has an active IPv6 address.
multicast-router (*yes | no*)	Shows if a multicast router is detected on the bridge interface. Monitoring value appears only when `igmp-snooping` is enabled.

```
[admin@MikroTik] /interface bridge monitor bridge1
                  state: enabled
    current-mac-address: 64:D1:54:C7:3A:59
            root-bridge: yes
         root-bridge-id: 0x8000.64:D1:54:C7:3A:59
         root-path-cost: 0
              root-port: none
             port-count: 3
  designated-port-count: 3
           fast-forward: no
       multicast-router: no
           igmp-querier: ether2 192.168.10.10
            mld-querier: ether2 fe80::e68d:8cff:fe39:3824
```

To monitor the current status of bridge ports, use the `monitor` command.

Sub-menu: `/interface bridge port`

Property	Description

multicast-router (*yes* / *no*)	Shows if a multicast router is detected on the port. Monitoring value appears only when `igmp-snooping` is enabled.

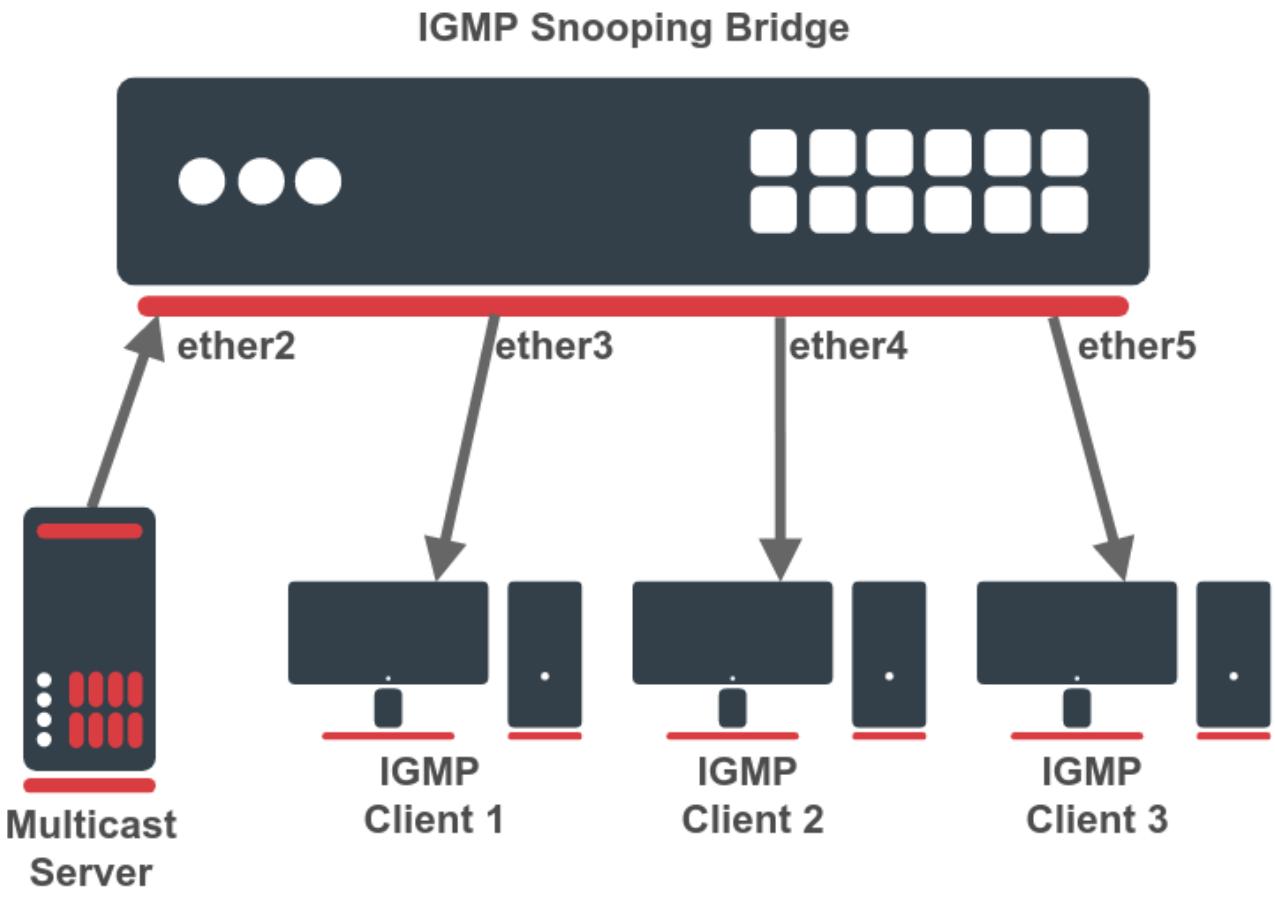
```
[admin@MikroTik] > /interface bridge port monitor [find]
              interface: ether2          ether3          ether4
                 status: in-bridge       in-bridge       in-bridge
            port-number: 1               2               3
                   role: designated-port designated-port designated-port
              edge-port: no              yes             yes
    edge-port-discovery: yes             yes             yes
    point-to-point-port: yes             yes             yes
           external-fdb: no              no              no
           sending-rstp: yes             yes             yes
               learning: yes             yes             yes
             forwarding: yes             yes             yes
       multicast-router: yes             no              no
       hw-offload-group: switch1         switch1         switch1
```

Configuration examples

Below are described the most common configuration examples. Some examples are using a bridge with VLAN filtering, so make sure to understand the filtering principles first - bridge VLAN filtering, bridge VLAN table.

Basic IGMP snooping configuration

The first example consists only of a single IGMP snooping bridge, a single multicast source device, and a couple of multicast client devices. See a network scheme below.



First, create a bridge interface with enabled IGMP snooping. In this example, there is no active IGMP querier (no multicast router or proxy), so a local IGMP querier must be enabled on the same bridge. This can be done with a `multicast-querier` setting. If there is no active IGMP querier in the LAN, the unregistered IP multicast will be flooded and multicast entries will always timeout from the multicast database.

```
/interface bridge
add igmp-snooping=yes multicast-querier=yes name=bridge1
```

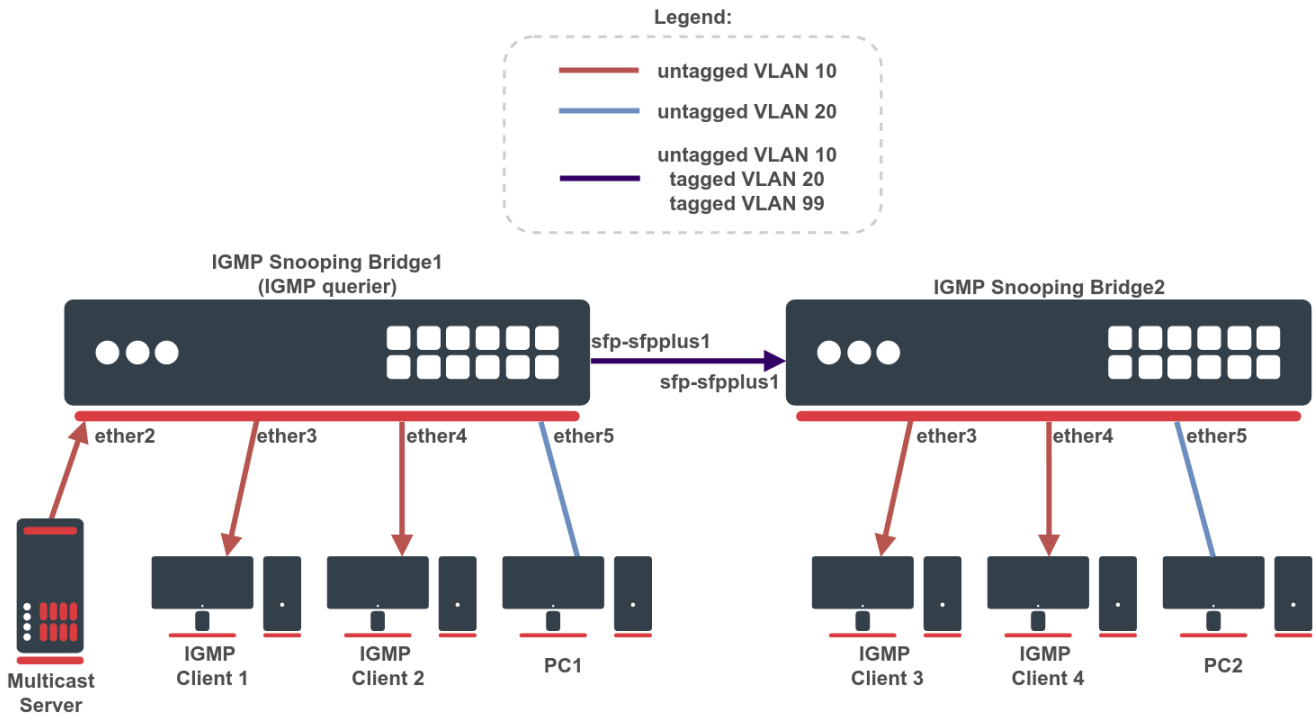
Then add the necessary interfaces as bridge ports.

```
/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
add bridge=bridge1 interface=ether4
add bridge=bridge1 interface=ether5
```

The basic IGMP snooping configuration is finished. Use "`/interface bridge mdb print`" command to monitor the active multicast groups. If necessary, you can configure an IP address and DHCP server on the same bridge interface.

IGMP snooping configuration with VLANs

The second example adds some complexity. There are two IGMP snooping bridges and we need to isolate the multicast traffic on a different VLAN. See a network scheme below.



First, create a bridge on both devices and add the needed interfaces as bridge ports. To change untagged VLAN for a bridge port, use the `pvid` setting. The Bridge1 will be acting as an IGMP querier. Below are the configuration commands for the Bridge1:

```
/interface bridge
add igmp-snooping=yes multicast-querier=yes name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 pvid=10
add bridge=bridge1 interface=ether3 pvid=10
add bridge=bridge1 interface=ether4 pvid=10
add bridge=bridge1 interface=ether5 pvid=20
add bridge=bridge1 interface=sfp-sfpplus1 pvid=10
```

And for the Bridge2:

```
/interface bridge
add igmp-snooping=yes name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether3 pvid=10
add bridge=bridge1 interface=ether4 pvid=10
add bridge=bridge1 interface=ether5 pvid=20
add bridge=bridge1 interface=sfp-sfpplus1 pvid=10
```

⚠ Bridge IGMP querier implementation can only send untagged IGMP queries. In case tagged IGMP queries should be sent or IGMP queries should be generated in multiple VLANs, it is possible to install a multicast package, add a VLAN interface and configure a PIM interface on VLAN. The PIM interface can be used as an IGMP querier.

Make sure to configure management access for devices. It is essential when configuring a bridge with VLAN filtering. In this example, a VLAN 99 interface with an IP address is added to the bridge. This VLAN will be allowed on the tagged sfp-sfpplus1 port. Below are configuration commands for the Bridge1:

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT network=192.168.99.0
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,sfp-sfpplus1 vlan-ids=99
```

And for the Bridge2:

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.2/24 interface=MGMT network=192.168.99.0
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,sfp-sfpplus1 vlan-ids=99
```

Add bridge VLAN entries and specify tagged and untagged ports. The VLAN 99 entry was already created when configuring management access, only VLAN 10 and VLAN 20 should be added now. Below are the configuration commands for the Bridge1:

```
/interface bridge vlan
add bridge=bridge1 untagged=ether2,ether3,ether4,sfp-sfpplus1 vlan-ids=10
add bridge=bridge1 tagged=sfp-sfpplus1 untagged=ether5 vlan-ids=20
```

And for the Bridge2:

```
/interface bridge vlan
add bridge=bridge1 untagged=ether3,ether4,sfp-sfpplus1 vlan-ids=10
add bridge=bridge1 tagged=sfp-sfpplus1 untagged=ether5 vlan-ids=20
```

Last, enable VLAN filtering. Below is the configuration command for Bridge1 and Bridge2:

```
/interface bridge set [find name=bridge1] vlan-filtering=yes
```

At this point, VLANs and IGMP snooping are configured and devices should be able to communicate through ports. However, it is recommended to go even a step further and apply some additional filtering options. Enable `ingress-filtering` and `frame-types` on bridge ports. Below are the configuration commands for the Bridge1:

```
/interface bridge port
set [find interface=ether2] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=ether3] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=ether4] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=ether5] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=sfp-sfpplus1] ingress-filtering=yes
```

And for the Bridge2:

```
/interface bridge port
set [find interface=ether3] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=ether4] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=ether5] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
set [find interface=sfp-sfpplus1] ingress-filtering=yes
```

Static MDB entries

Since RouterOS version 7.7, it is possible to create static MDB entries for IPv4 and IPv6 multicast groups. For example, to create a static MDB entry for multicast group 229.10.10.10 on ports ether2 and ether3 on VLAN 10, use the command below:

```
/interface bridge mdb
add bridge=bridge1 group=229.10.10.10 ports=ether2,ether3 vid=10
```

Verify the results with the `print` command:

```
[admin@MikroTik] > /interface bridge mdb print where group=229.10.10.10
Columns: GROUP, VID, ON-PORTS, BRIDGE
 # GROUP         VID  ON-PORTS  BRIDGE
12 229.10.10.10   10  ether2    bridge1
                      ether3
```

In case a certain IPv6 multicast group does not need to be snooped and it is desired to be flooded on all ports and VLANs, it is possible to create a static MDB entry on all VLANs and ports, including the bridge interface itself. Use the command below to create a static MDB entry for multicast group ff02::2 on all VLANs and ports (modify the `ports` setting for your particular setup):

```
/interface bridge mdb
add bridge=bridge1 group=ff02::2 ports=bridge1,ether2,ether3,ether4,ether5

[admin@MikroTik] > /interface bridge mdb print where group=ff02::2
Flags: D - DYNAMIC
Columns: GROUP, VID, ON-PORTS, BRIDGE
 #   GROUP    VID  ON-PORTS  BRIDGE
 0   ff02::2               bridge1
15 D ff02::2    1  bridge1   bridge1
16 D ff02::2   10  bridge1   bridge1
                   ether2
                   ether3
                   ether4
                   ether5
17 D ff02::2   20  bridge1   bridge1
                   ether2
                   ether3
18 D ff02::2   30  bridge1   bridge1
                   ether2
                   ether3
```

Bridge VLAN Table

Summary

It is possible to use a bridge to filter out VLANs in your network. To achieve this, you should use the Bridge VLAN Filtering feature. This feature should be used instead of many known VLAN misconfigurations that are most likely causing you either performance issues or connectivity issues, you can read about one of the most popular misconfigurations in the VLAN in a bridge with a physical interface section. The most important part of the bridge VLAN filtering feature is the bridge VLAN table, which specifies which VLANs are allowed on each port, but configuring it might get quite complex if you are trying to make a more advanced setup, for generic setups you should be able to configure your device using the Trunk and Access ports example, but the purpose of this guide is to provide in-depth explanation and point out some of the behavior characteristics when using bridge VLAN Filtering.

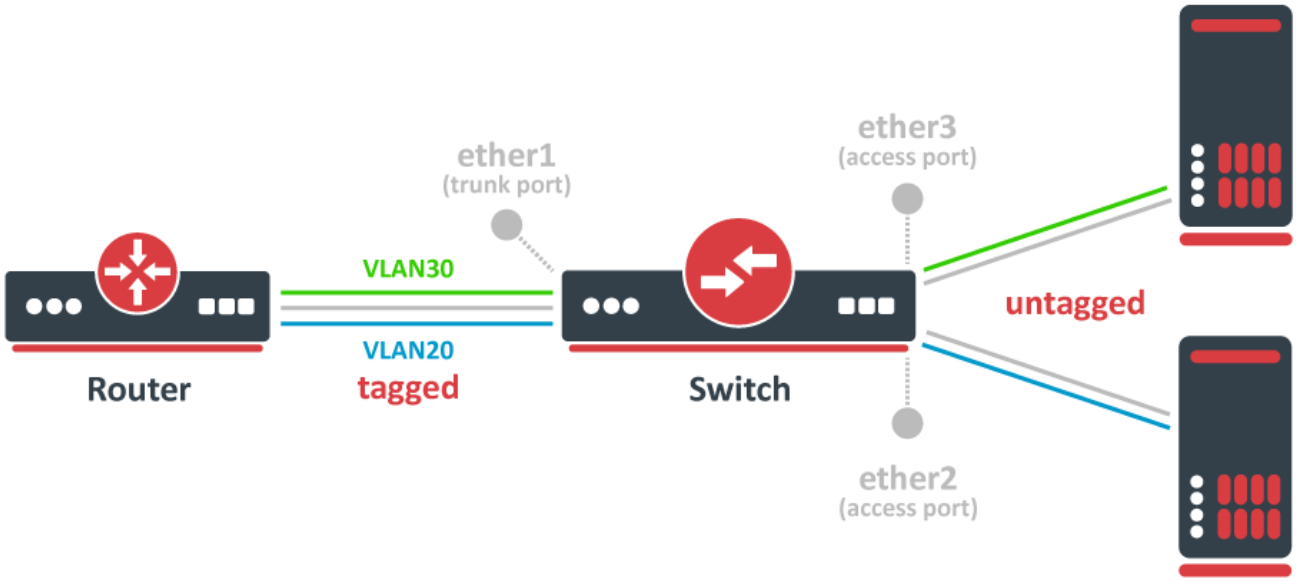
Background

Before explaining bridge VLAN filtering in-depth, you should understand a few basic concepts that are involved in bridge VLAN filtering.

- **Tagged/Untagged** - Under `/interface bridge vlan` menu, you can specify an entry that contains tagged and untagged ports. In general, tagged ports should be your trunk ports and untagged ports should be your access ports. By specifying a tagged port the bridge will always set a VLAN tag for packets that are being sent out through this port (egress). By specifying an untagged port the bridge will always remove the VLAN tag from egress packets.
- **VLAN-ids** - Under `/interface bridge vlan` menu, you can specify an entry in which certain VLANs are allowed on specific ports. The VLAN ID is checked on egress ports. If the packet contains a VLAN ID that does not exist in the bridge VLAN table for the egress port, then the packet is dropped before it gets sent out.
- **PVID** - The Port VLAN ID is used for access ports to tag all ingress traffic with a specific VLAN ID. A dynamic entry is added in the bridge VLAN table for every PVID used, the port is automatically added as an untagged port.
- **Ingress filtering** - By default, VLANs that don't exist in the bridge VLAN table are dropped before they are sent out (egress), but this property allows you to drop the packets when they are received (ingress).
- **Management access** - The bridge is supposed to simply forward packets between bridge ports and it would seem to other devices that there is simply a wire between them. With bridge VLAN filtering you can limit which packets are allowed to access the device that has the bridge configured, the most common practice is to allow access to the device only by using a very specific VLAN ID, but there are other ways you can grant access to the device. Management access is a great way to add another layer of security when accessing the device through a bridge port, this type of access is sometimes called the management port. For devices that support VLAN Filtering with hardware offloading, It is also related to the CPU port of a bridge.
- **CPU port** - Every device with a switch chip has a special purpose port called CPU port and it is used to communicate with the device's CPU. For devices that support VLAN filtering with hardware offloading, this port is the bridge interface itself. This port is mostly used to create management access but can be used for other purposes, for example, to route traffic between VLANs, to mark packets, and to apply queues.
- **frame-type** - You can filter out packets whether they have a VLAN tag or not, this is useful to add an extra layer of security for your bridge ports.
- **EtherType** - By default, a VLAN aware bridge will filter VLANs by checking the C-TAG (0x8100), all other VLAN tags are considered as untagged packets (without a VLAN tag). The selected EtherType will be used for VLAN filtering and VLAN tagging/untagging.
- **VLAN Tunnelling** - If the EtherType of the packet does not match with the EtherType configured for the bridge, then ingress packets are considered as untagged packets, this behavior gives a possibility to encapsulate VLANs into another, different VLAN. This also gives a possibility to divert specific traffic through different devices in your network.
- **Tag stacking** - If a packet has a VLAN tag that matches the EtherType, then the packet is considered as a tagged packet, but you can force another VLAN tag regardless of the packet's content. By setting `tag-stacking=yes` on a bridge port, you will add another VLAN tag with the PV ID value on top of any other tag for all ingress packets.

Trunk/Access port setup

Below you can find a very common diagram for a very typical type of setup that consists of a trunk port and multiple access ports:



This setup is very common since it gives the possibility to divide your network into multiple segments while using a single switch and maybe a single router, such a requirement is very common for companies that want to separate multiple departments. With VLANs you can use different DHCP Servers, which can give out an IP address from a different subnet based on the VLAN ID, which makes creating Firewall rules and QoS a lot easier.

In such a setup you would connect some generic devices like Desktop PCs to **ether2** and **ether3**, these can be considered as workstations and they generally only use untagged traffic (it is possible to force a VLAN tag for all traffic that is sent out a generic workstation, though it is not very common). To isolate some workstations from other workstations you must add a VLAN tag to all packets that enter **ether2** or **ether3**, but to decide what VLAN ID should the packet get, you need to use a concept called **Port-based VLANs**. In this concept, packets get a VLAN tag with a VLAN ID based on the bridge port to which the device is connected. For example, in this setup the device on **ether2** will get a VLAN tag with **VLAN20** and the device on **ether3** will get a VLAN tag with **VLAN30**, this concept is very scalable as long as you have enough bridge ports. This should give you the understanding that traffic between the bridge and devices behind **ether2/ether3** is untagged (since there is no VLAN tag, hence the name).

When we have determined our untagged ports, we can now determine our tagged ports. Tagged ports are going to be the trunk ports (the port, that carries multiple VLANs) and usually, this port is connected to a router or another switch/bridge, you can have multiple trunk ports as well. Tagged ports are always carrying packets with a VLAN tag (hence the name) and you must **ALWAYS** specify the tagged ports for each VLAN ID you want this port to forward. It is possible that a port is a tagged port for one VLAN ID and the same port is an untagged port for a different VLAN ID, but this is for a different type of setup (Hybrid port setup).

A special note must be added for the PVID property. This property should be used on access ports, but it can be used for trunk ports as well (in Hybrid port setup). By using the PVID property you are adding a new VLAN tag with a VLAN ID that is specified in the PVID to all **UNTAGGED** packets that are received on that specific bridge port. The PVID does not have any effect on tagged packets, this means that, for example, if a packet with a VLAN tag of **VLAN40** is received on **ether2** that has `PVID=20`, then the VLAN tag is **NOT** changed and forwarding will depend on the entries from the bridge VLAN table.

To configure the trunk/access port setup, you need to first create a bridge:

```
/interface bridge
add name=bridge1
```

> Don't enable VLAN filtering yet as you might get locked out from the device because of the lack of management access, which is configured at the end.

Add the bridge ports and specify PVID for each access port:

```
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2 pvid=20
add bridge=bridge1 interface=ether3 pvid=30
```

⚠ PVID has no effect until VLAN filtering is enabled.

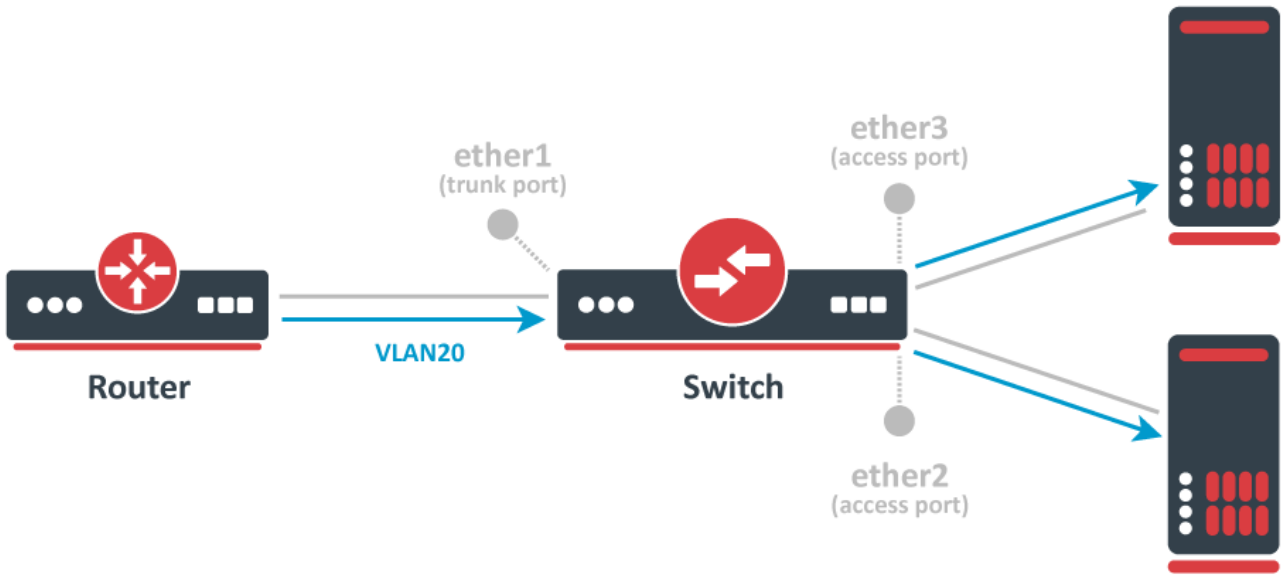
Add appropriate entries in the bridge VLAN table:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 untagged=ether2 vlan-ids=20
add bridge=bridge1 tagged=ether1 untagged=ether3 vlan-ids=30
```

You might think that you could simplify this entry with a single entry, similar to this:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 untagged=ether2,ether3 vlan-ids=20,30
```

Do **NOT** use multiple VLAN IDs on access ports. This will unintentionally allow both **VLAN20** and **VLAN30** on both access ports. In the example above, **ether3** is supposed to set a VLAN tag for all ingress packets to use **VLAN30** (since `PVID=30`), but this does not limit the allowed VLANs on this port when VLANs are being sent out through this port. The bridge VLAN table is responsible for deciding whether a VLAN is allowed to be sent through a specific port or not. The entry above specifies that both **VLAN20** and **VLAN30** are allowed to be sent out through **ether2** and **ether3** and on top of that the entry specifies that packets should be sent out without a VLAN tag (packets are sent out as untagged packets). As a result, you may create a packet leak from VLANs to ports that are not even supposed to receive such traffic, see the image below.



A misconfigured VLAN table allows VLAN20 to be sent through ether3, it will also allow VLAN30 through ether2

⚠ Don't use more than one VLAN ID specified in a bridge VLAN table entry for access ports, you should only specify multiple VLAN IDs for trunk ports.

It is not necessary to add a bridge port as an untagged port, because each bridge port is added as an untagged port dynamically with a VLAN ID that is specified in the PVID property. This is because of a feature that automatically will add an appropriate entry in the bridge VLAN table for convenience and performance reasons, this feature does have some caveats that you must be aware of. All ports that have the same PVID will be added to a single entry for the appropriate VLAN ID as untagged ports, but note that the **Bridge interface** also has a VLAN ID.

For testing purposes, we are going to enable VLAN filtering, but note that it might make you lose access to the device since it does not have management access configured yet (we will configure it later). It is always recommended to configure VLAN filtering while using a serial console, though you can also configure a device through a port, that is not added to a bridge. Make sure you are using a serial console or connected through a different port (that is not in a bridge) and enable VLAN filtering:

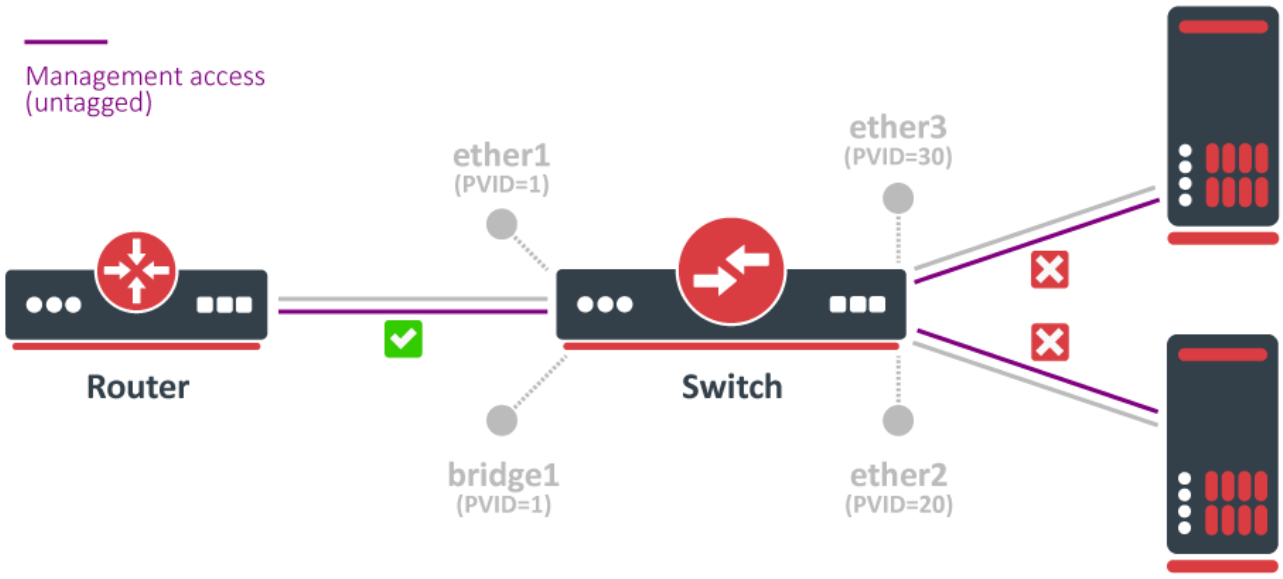
```
/interface bridge set bridge1 vlan-filtering=yes
```

You might not lose access to the device as soon as you enable VLAN filtering, but you might get disconnected since the bridge must reset itself in order for VLAN filtering to take any effect, which will force you to reconnect (this is mostly relevant when using MAC-telnet). There is a chance you might be able to access your device using untagged traffic, this scenario is described below.

If you have enabled VLAN filtering now and printed out the current VLAN table, you would see such a table:

```
[admin@MikroTik] > /interface bridge vlan print
Flags: X - disabled, D - dynamic
 #    BRIDGE                    VLAN-IDS  CURRENT-TAGGED       CURRENT-UNTAGGED
 0    bridge1                   20        ether1               ether2
 1    bridge1                   30        ether1               ether3
 2 D  bridge1                   1                              bridge1
                                                               ether1
```

There is a dynamic entry added for **VLAN1** since `PVID=1` is set by default to all bridge ports (including our trunk port, **ether1**), but you should also notice that the **bridge1** interface (the CPU port) is also added dynamically. You should be aware that **bridge1** is also a bridge port and therefore might get added to the bridge VLAN table dynamically. There is a chance that you might unintentionally allow access to the device because of this feature. For example, if you have followed this guide and left **PVID=1** set for the trunk port (**ether1**) and did not change the PVID for the CPU port (**bridge1**) as well, then access through **ether1** to the device using untagged traffic is allowed, this is also visible when you print out the bridge VLAN table. This scenario is illustrated in the image below:



Unintentionally allowed management access using untagged traffic through the trunk port

Always check the bridge VLAN table if you have not unintentionally allowed certain VLANs or untagged traffic to specific ports, especially the CPU port (bridge).

There is a simple way to prevent the bridge (CPU port) from being added as an untagged port, you can simply set the PVID on the trunk port to be different than the bridge's PVID (or change the bridge's PVID), but there is another option, which is more intuitive and recommended. Since you are expecting that the trunk port is only supposed to receive tagged traffic (in this example, it should only receive **VLAN20/VLAN30**), but no untagged traffic, then you can use ingress-filtering along with frame-type to filter out unwanted packets, but to fully understand the behavior of ingress filtering, we must first understand the details of management access.

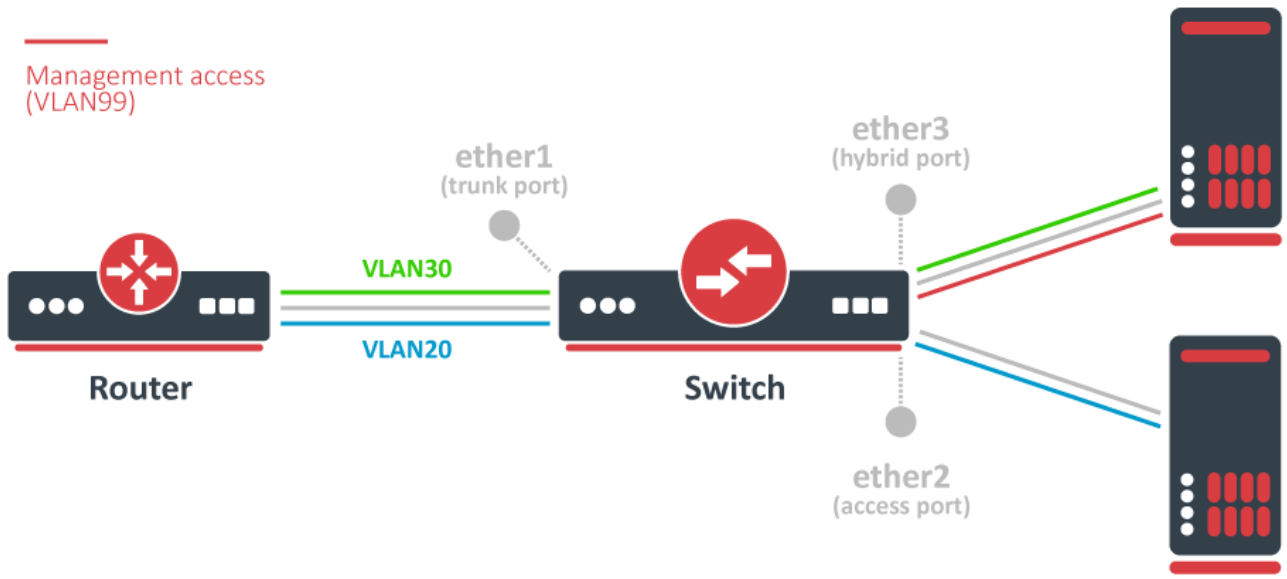
Management access is used to create a way to access a device through a bridge that has VLAN filtering enabled. You could simply allow untagged access and doing that is fairly simple. Let us say you wanted the workstation behind **ether3** to be able to access the device, we assumed before that the workstation is a generic computer that will not use tagged packets and therefore will only send out untagged packets, this means that we should add the CPU port (**bridge1**) as an untagged interface to the bridge VLAN table, to do so, simply use the same PVID value for the **bridge1** and **ether3** ports and set both ports as untagged members for the VLAN ID. In this case, you are going to connect from **ether3** that has `PVID=30`, so you change the configuration accordingly:

```
/interface bridge set [find name=bridge1] pvid=30
/interface bridge vlan set [find vlan-ids=30] untagged=bridge1,ether3
```

⚠ You can use the feature that dynamically adds untagged ports with the same PVID value, you can simply change the PVID to match between **ether3** and **bridge1**.

Allowing access to the device using untagged traffic is not considered a good security practice, a much better way is to allow access to the device using a very specific VLAN sometimes called the management VLAN, in our case, this is going to be **VLAN99**. This adds a significant layer of security since an attacker must guess the VLAN ID that is being used for management purposes and then guess the login credentials, on top of this you can even add another layer of security by allowing access to the device using only certain IP addresses. The purpose of this guide is to provide an in-depth explanation, for that reason, we are adding a level of complexity to our setup to understand some possible caveats that you must take into account. We are going to allow access from an access port using tagged traffic (illustrated in the image below). To allow access to the device using **VLAN99** from **ether3**, we must add a proper entry in the bridge VLAN table. Additionally, a network device connected to ether3 must support VLAN tagging.

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,ether3 vlan-ids=99
```



Management access using tagged traffic through an access port (which makes it a hybrid port)

⚠ If PVID for ether1 and bridge1 matches (by default, it does match with 1), then access to the device is allowed using untagged traffic from ether1 because of the feature that dynamically adds untagged ports to the bridge VLAN table.

But you might notice that access using **VLAN99** does not work at this point, this is because you need a VLAN interface that listens for tagged traffic, you can simply create this interface for the appropriate VLAN ID and you can set an IP address for the interface as well:

```
/interface vlan
add interface=bridge1 name=VLAN99 vlan-id=99
/ip address
add address=192.168.99.2/24 interface=VLAN99
```

⚠ Our access port (**ether3**) at this point expects tagged and untagged traffic at the same time, such a port is called a **hybrid port**.

At this point, we can benefit from using `ingress-filtering` and `frame-type`. First, we are going to focus on `frame-type`, which limits the allowed packet types (tagged, untagged, both), but for `frame-type` to work properly, `ingress-filtering` must be enabled, otherwise it will not have any effect. In our example, where we wanted to allow access from **ether3** using tagged traffic (**VLAN99**) and at the same time allow a generic workstation to access the network, we can conclude that this port needs to allow tagged and untagged packets, but **ether1** and **ether2** are supposed to receive only specific types of packets, for this reasons we can enhance our network's security. Since **ether1** is our trunk port, it is only supposed to carry tagged packets, but **ether2** is our access port so it should not carry any tagged packets, based on these conclusions we can drop invalid packets:

```
/interface bridge port
set [find where interface=ether1] ingress-filtering=yes frame-types=admit-only-vlan-tagged
set [find where interface=ether2] ingress-filtering=yes frame-types=admit-only-untagged-and-priority-tagged
```

Let's say that you forgot to enable ingress-filtering and change the frame-type property on **ether1**, this would unintentionally add access to the device through **ether1** using untagged traffic since PVID matches for **bridge1** and **ether1**, but you are expecting only tagged traffic to be able to access the device. It is possible to drop all untagged packets that are destined for the **CPU port**:

```
/interface bridge
set bridge1 frame-types=admit-only-vlan-tagged ingress-filtering=yes
```

This does not only drop untagged packets, but disables the feature that dynamically adds untagged ports to the bridge VLAN table. If you print out the current bridge VLAN table you will notice that **bridge1** is not dynamically added as an untagged port:

```
[admin@MikroTik] > /interface bridge vlan print
Flags: X - disabled, D - dynamic
 #   BRIDGE       VLAN-IDS  CURRENT-TAGGED        CURRENT-UNTAGGED
 0   bridge1      20        ether1
 1   bridge1      30        ether1                ether3
 2 D bridge1      1                               ether1
 3   bridge1      99        bridge1
                            ether3
```

⚠ When `frame-type=admit-only-vlan-tagged` is used on a port, then the port is not dynamically added as an untagged port for the PVID.

While `frame-type` can be used to drop a certain type of packet, the `ingress-filtering` can be used to filter out packets before they can be sent out. To fully understand the need for ingress filtering, consider the following scenario: **VLAN99** is allowed on **ether3** and **bridge1**, but you can still send **VLAN99** traffic from **ether1** to **ether3**, this is because the bridge VLAN table checks if a port is allowed to carry a certain VLAN only on egress ports. In our case, **eth er3** is allowed to carry **VLAN99** and for this reason, it is forwarded. To prevent this you **MUST** use ingress-filtering. With ingress filtering, ingress packets are also checked, in our case, the bridge VLAN table does not contain an entry that **VLAN99** is allowed on **ether1** and therefore will be dropped immediately. Of course, in our scenario without ingress filtering connection cannot be established since **VLAN99** can be forwarded only from **ether1** to **ether3** , but not from **ether3** to **ether1**, though there are still possible attacks that can be used in such a misconfiguration (for example, ARP poisoning). The packet dropping behavior is illustrated in the image below:

Trunk/access port setup with and without ingress filtering. Ingress filtering can prevent unwanted traffic from being forwarded. Note that ether1 is not allowed to carry VLAN99 in the bridge VLAN table.

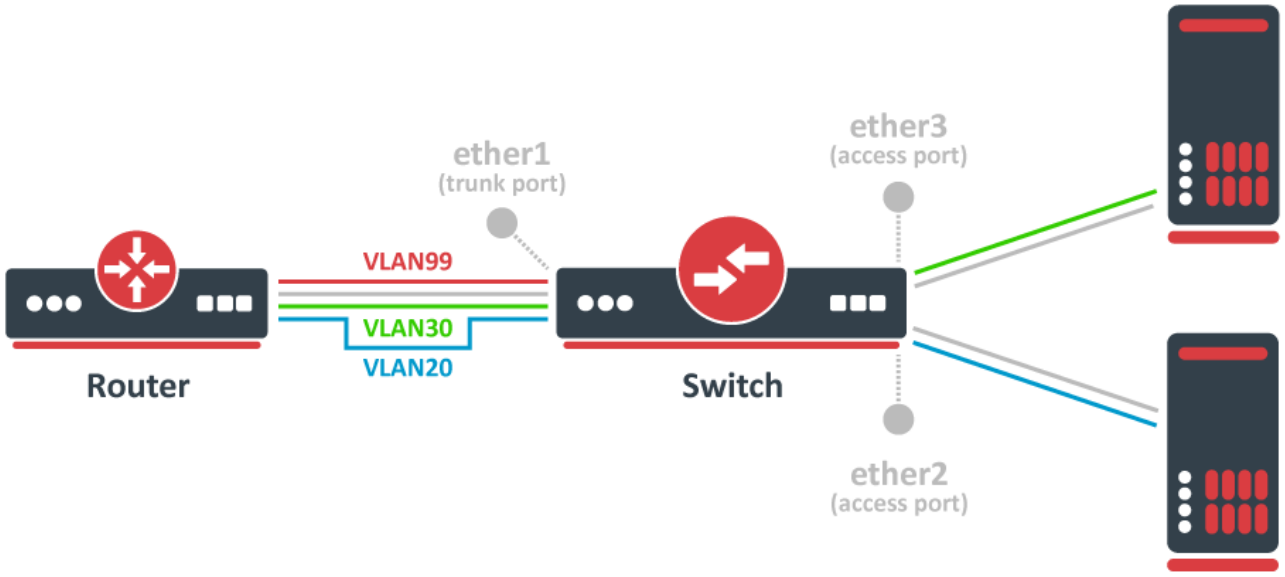
> Always try to use `ingress-filtering` wherever it is possible, it adds a significant layer of security.

The ingress-filtering can be used on the **CPU port** (bridge) as well, this can be used to prevent some possible attack vectors and limit the allowed VLANs that can access the CPU. It is better to drop a packet on an ingress port, rather than on an egress port, this reduces the CPU load, which is quite crucial when you are using hardware offloading with bridge VLAN filtering.

> The `ingress-filtering` property only affects ingress traffic, but `frame-type` affects both egress and ingress traffic.

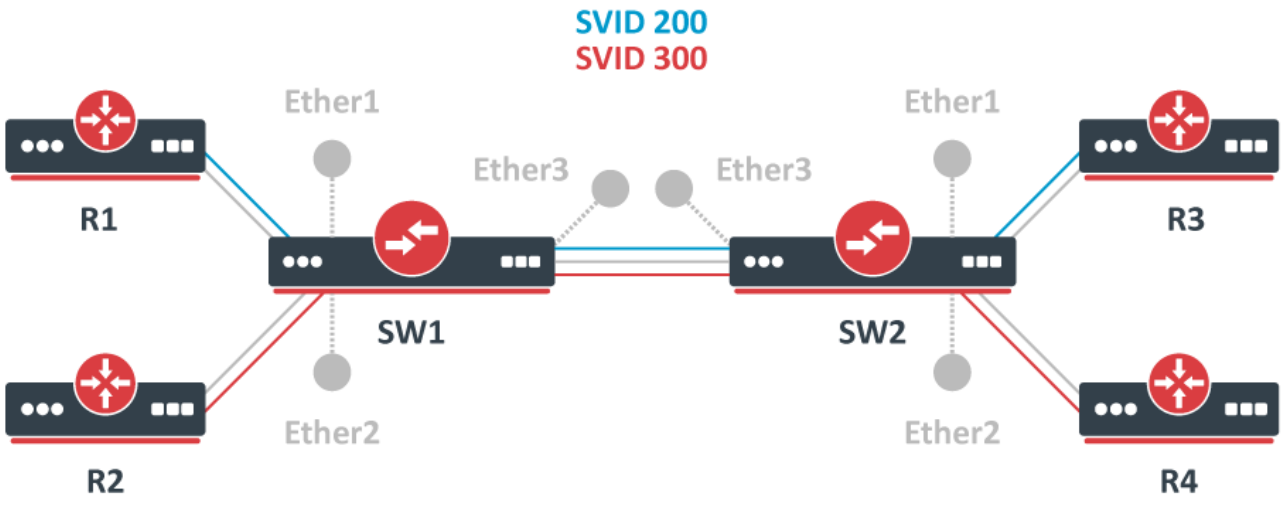
Even though you can limit the allowed VLANs and packet types on a port, it is never a good security practice to allow access to a device through access ports since an attacker could sniff packets and extract the management VLAN's ID, you should only allow access to the device from the trunk port (**ether1**) since trunk ports usually have better physical security, you should remove the previous entry and allow access to the device through the port that is connected to your router (illustrated in the image below):

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,ether1 vlan-ids=99
```



VLAN Tunnelling setup

In some cases, you might want to forward already tagged traffic through certain switches. This is a quite common setup for backbone infrastructures since it provides a possibility to encapsulate traffic from, for example, your edge routers and seamlessly forward it over your backbone to another edge router. Below you can find an example of a VLAN tunneling topology:



Provider bridge topology

SVID stands for Service VID, indicating the tag type along with the VID.

⚠ To fully understand how to configure VLAN tunneling properly, you should first read the Trunk/Access port setup section before proceeding any further.

There are two possible ways to achieve this, one is the standardized IEEE 802.1ad way, and the other way is using **Tag stacking**, we will first review the standardized way since the same principles apply to both ways and only a couple of parameters must be changed to use the other method. The way VLAN tunneling works is that the bridge checks if the outer VLAN tag is using the same VLAN tag as specified as ether-type. If the VLAN tag matches, the packet is considered as a tagged packet, otherwise, it is considered as an untagged packet.

⚠ The bridge checks only the outer tag (closest to the MAC address), any other tag is ignored anywhere in a bridge configuration. The bridge is not aware of the packet contents, even though there might be another VLAN tag, only the first VLAN tag is checked.

The ether-type property allows you to select the following EtherTypes for the VLAN tag:

- 0x88a8 - IEEE 802.1ad, Service Tag
- 0x8100 - IEEE 802.1Q, Customer VLAN (regular VLAN tag)
- 0x9100 - Unofficial tag type (rarely used)

To properly configure bridge VLAN filtering, you must understand how the bridge distinguishes between tagged and untagged packets. As mentioned before, the bridge will check if EtherType matches with the outer VLAN tag in the packet. For example, consider the following packet:

```
FFFFFFFFFFFF 6C3B6B7C413E 8100 6063 9999
---------------------------------------
DST-MAC = FFFFFFFFFFFF
SRC-MAC = 6C3B6B7C413E
Outer EtherType = 8100 (IEEE 802.1Q VLAN tag)
VLAN priority = 6
VLAN ID = 99 (HEX = 63)
Inner EtherType = 9999
```

Let us assume that we have set **`ether-type=0x88a8`**, in this case, the packet above will be considered untagged since the bridge is looking for a different VLAN tag. Lets now consider the following packet:

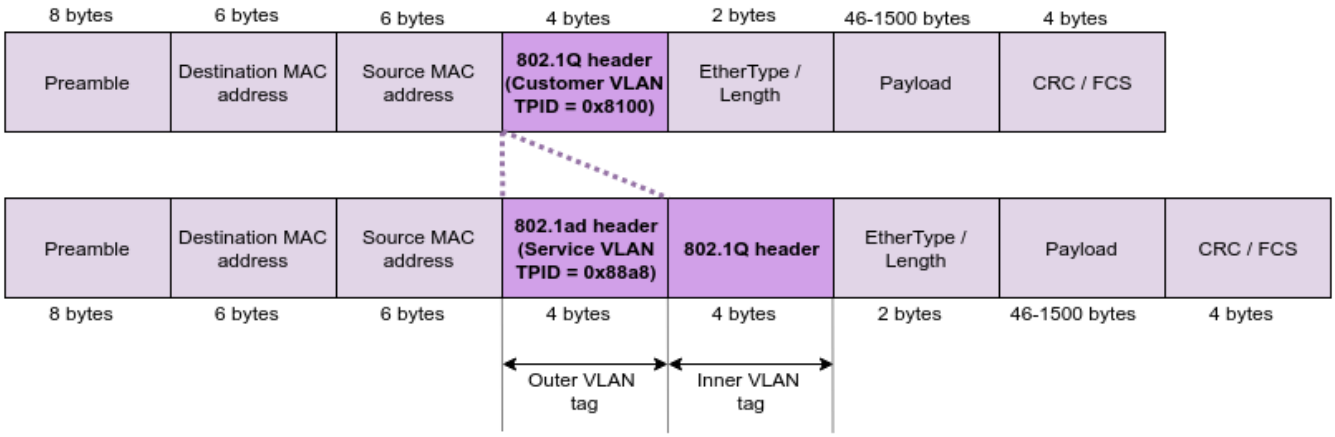
```
FFFFFFFFFFFF 6C3B6B7C413E 88A8 6063 8100 5062 9999
---------------------------------------
DST-MAC = FFFFFFFFFFFF
SRC-MAC = 6C3B6B7C413E
Outer EtherType = 88A8 (IEEE 802.1ad VLAN tag)
VLAN priority = 6
VLAN ID = 99 (HEX = 63)
Inner EtherType 1 = 8100 (IEEE 802.1Q VLAN tag)
VLAN priority = 5
VLAN ID = 98 (HEX = 62)
Innter EtherType 2 = 9999
```

This time let us assume that we have set **`ether-type=0x8100`**, in this case, the packet above is considered as untagged as well since the outer tag is using an IEEE 802.1ad VLAN tag. The same principles apply to other VLAN related functions, for example, the `PVID` property will add a new VLAN tag on access ports and the VLAN tag will be using the EtherType specified in ether-type.

Both **SW1** and **SW2** are using the same configuration:

```
/interface bridge
add name=bridge1 vlan-filtering=yes ether-type=0x88a8
/interface bridge port
add interface=ether1 bridge=bridge1 pvid=200
add interface=ether2 bridge=bridge1 pvid=300
add interface=ether3 bridge=bridge1
/interface bridge vlan
add bridge=bridge1 tagged=ether3 untagged=ether1 vlan-ids=200
add bridge=bridge1 tagged=ether3 untagged=ether2 vlan-ids=300
```

In this example, we are assuming that all routers are passing traffic that is using a regular/customer VLAN tag. Such traffic on switches will be considered as untagged traffic based on the principle described above. Switches will encapsulate this traffic using a Service VLAN tag (the outer 802.1ad tag) and traffic between **SW1** and **SW2** will be considered as tagged. Before traffic reaches its destination, the switches will decapsulate the outer tag and forward the original 802.1Q tagged frame. See a packet example below:



A packet example before and after 802.1ad VLAN encapsulation

> All principles that apply to the regular trunk/access port setup using IEEE 802.1Q also apply to VLAN tunneling setups, make sure you are limiting VLANs and packet type properly using the bridge VLAN table and ingress filtering.

In case you want to create management access from, let's say, **ether3** to the device and want to use **VLAN99**, then you would use such commands:

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1,ether3 vlan-ids=99
/interface vlan
add interface=bridge1 name=VLAN99 use-service-tag=yes vlan-id=99
/ip address
add address=192.168.99.2/24 interface=VLAN99
```

As you may notice, the only difference is that the VLAN interface is using `use-service-tag=yes`, this sets the VLAN interface to listen to IEEE 802.1ad VLAN tags. This will require you to use the IEEE 802.1ad VLAN tag to access the device using the management VLAN - you will not be able to connect to the device using a regular VLAN tag while bridge VLAN filtering is enabled. The ether-type is set globally and will affect all bridge VLAN filtering functions.

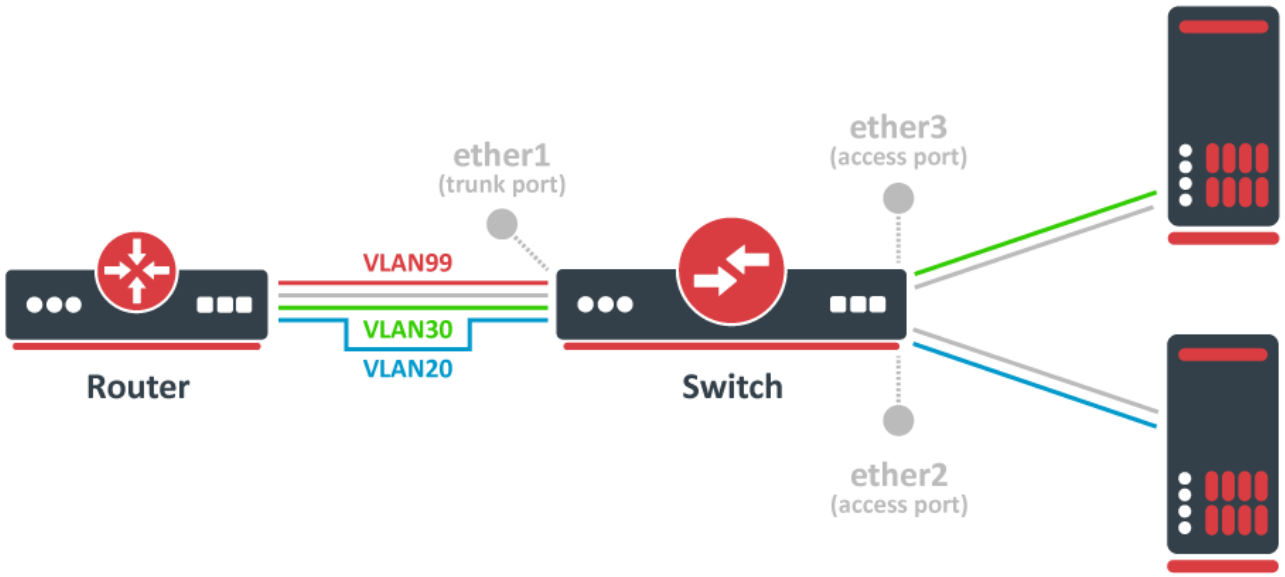
> Devices with switch chip Marvell-98DX3257 (e.g. CRS354 series) do not support VLAN filtering on 1Gbps Ethernet interfaces for other VLAN types (`0x88a8` and `0x9100`).

Tag Stacking

In the VLAN Tunnelling setup, we were adding a new VLAN tag that was different from the VLAN tag, but it is possible to add a new VLAN tag regardless of the packet contents. The difference between the regular VLAN tunneling setup is that the bridge does not check if the packet is tagged or untagged, it assumes that all packets that are received on a specific port are all untagged packets and will add a new VLAN tag regardless of whether a VLAN tag is present or not, this is called **Tag Stacking** since it "stacks" VLAN tags on top of the previous tag, regardless of the VLAN tag type. This is a very common setup for networks that do not support the IEEE 802.1ad standard, but still want to encapsulate VLAN traffic into a new VLAN.

The VLAN tag that is going to be added depends on `ether-type` and `PVID`. For example, if you have `ether-type=0x8100` and `PVID=200` on a port, then the bridge will add a new IEEE 802.1Q VLAN tag right on top of any other tag (if such are present). The same VLAN filtering principles still apply, you have to determine which ports are going to be your trunk ports and mark them as tagged ports, determine your access ports, and add them as untagged ports.

To explain how VLAN tagging and untagging works with tag stacking, let us use the same network topology as before:



What we want to achieve is that regardless of what is being received on **ether2** and **ether3**, a new VLAN tag will be added to encapsulate the traffic that is coming from those ports. `Tag-stacking` forces a new VLAN tag, so we can use this property to achieve our desired setup. We are going to be using the same configuration as in the Trunk/Access port setup, but with `tag-stacking` enabled on the access ports:

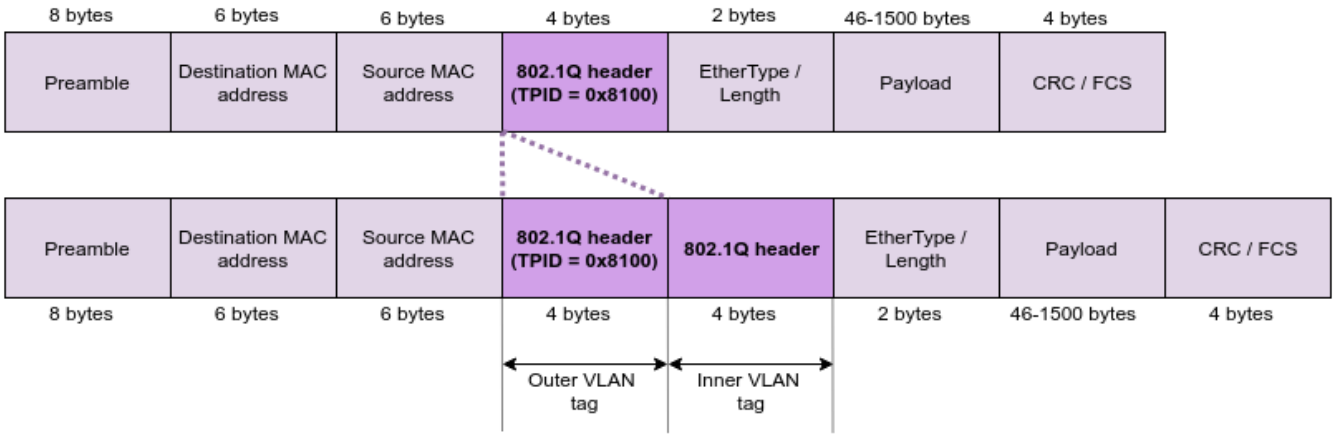
```
/interface bridge
add name=bridge1 vlan-filtering=yes ether-type=0x8100
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2 tag-stacking=yes pvid=20
add bridge=bridge1 interface=ether3 tag-stacking=yes pvid=30
/interface bridge vlan
add bridge=bridge1 tagged=ether1 untagged=ether2 vlan-ids=20
add bridge=bridge1 tagged=ether1 untagged=ether3 vlan-ids=30
```

⚠ The added VLAN tag will use the specified `ether-type`. The selected EtherType will also be used for VLAN filtering. Only the outer tag is checked, but with tag-stacking in place, the tag checking is skipped and assumes that a new tag must be added either way.

Let us assume that the devices behind **ether2** and **ether3** are sending tagged **VLAN40** traffic. With this configuration, **ALL** packets will get encapsulated with a new VLAN tag, but you must make sure that you have added the VLAN ID from the outer tag to the bridge VLAN table. The **VLAN40** is not added to the bridge VLAN table since it is the inner tag and it is not checked, we are only concerned about the outer tag, which is either **VLAN20** or **VLAN30** dependi ng on the port.

Similar to other setups, the bridge VLAN table is going to be used to determine if the VLAN tag needs to be removed or not. For example, **ether1** receives tagged **VLAN20** packets, the bridge checks that **ether2** is allowed to carry **VLAN20** so it is about to send it out through **ether2**, but it also checks the bridge VLAN table whether the VLAN tag should be removed and since **ether2** is marked as an untagged port, then the bridge will forward these packets from **ethe r1** to **ether2** without the **VLAN20** VLAN tag.

From the access port perspective, the same principles as in the Trunk/Access port setup apply. All packets that are received on **ether2** will get a new VLAN tag with the VLAN ID that is specified in PVID, in this case, a new VLAN tag will be added with **VLAN20** and this VLAN will be subjected to VLAN filtering. See a packet example below:

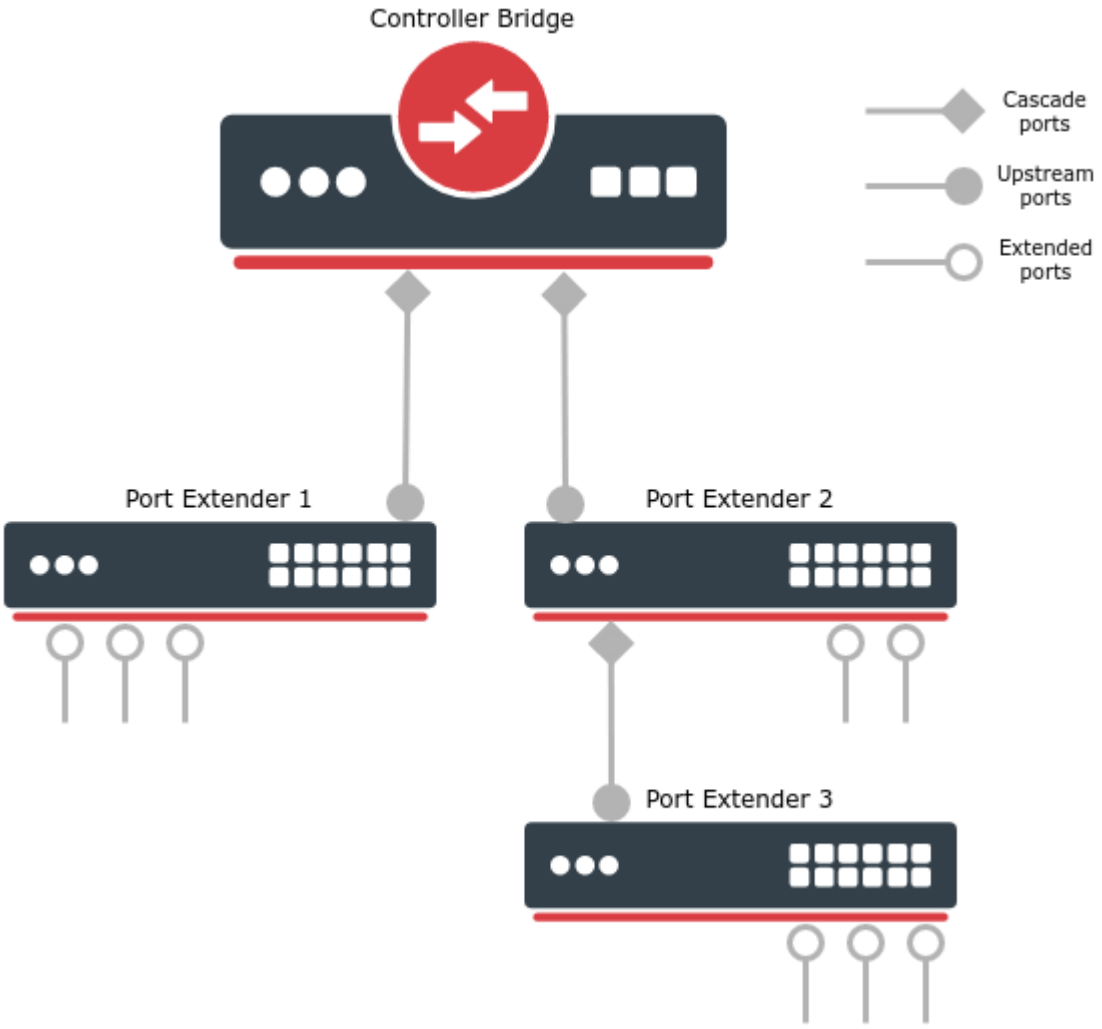


A packet example before and after tag stacking

Controller Bridge and Port Extender

Summary

Controller Bridge (CB) and Port Extender (PE) is an IEEE 802.1BR standard implementation in RouterOS for CRS3xx series switches. It allows virtually extending the CB ports with a PE device and manage these extended interfaces from a single controlling device. Such configuration provides a simplified network topology, flexibility, increased port density and ease of manageability. An example of Controller Bridge and Port Extender topology can be seen below.



The Controller Bridge establishes communication with the Port Extender through a **cascade port**. Similarly, the Port Extender will communicate with the Controller Bridge only through an **upstream port**. On a PE device, control ports must be configured and only one port (closest to the CB) will act as an upstream port, other control ports can act as a backup for upstream port or even cascade port for switches connected in series (e.g. Port Extender 2 and 3 in the image above). Cascade and upstream ports are used to transmit and receive control and network traffic. **Extended ports** are interfaces that are controlled by the CB device and they are typically connected to the end hosts. Extended ports only transmit and receive network traffic.

See supported features for each switch model below.

Model	Controller Bridge	Port Extender
netPower 15FR (CRS318-1Fi-15Fr-2S)	-	+
netPower 16P (CRS318-16P-2S+)	-	+
CRS310-1G-5S-4S+ (netFiber 9/IN)	-	+
CRS326-24G-2S+ (RM/IN)	-	+
CRS328-24P-4S+	-	+
CRS328-4C-20S-4S+	-	+
CRS305-1G-4S+	-	+
CRS309-1G-8S+	+	+
CRS317-1G-16S+	+	+
CRS312-4C+8XG	+	+
CRS326-24S+2Q+	+	+
CRS354-48G-4S+2Q+	+	+
CRS354-48P-4S+2Q+	+	+

Limitations

Although controller allows to configure port extender interfaces, some bridging and switching features cannot be used or will not work properly. Below are the most common controller and extender limitations. The list might change along upcoming RouterOS releases.

Feature	Support
Bonding for cascade and upstream ports	+
Bridge VLAN filtering	+
Bonding for extended ports	-
Dot1x authenticator (server)	-
Ingress and egress rate	-
Mirroring	-
Port ingress VLAN filtering	-
Port isolation	-
Storm control	-
Switch rules (ACL)	-
L3HW offloading	-
MLAG	-

Quick setup

In this example, we will create a Controlling Bridge (e.g. a CRS317-1G-16S+ switch) that will connect to a single Port Extender (e.g. a CRS326-24G-2S+ switch) through an SFP+1 interface.

First, configure a bridge with enabled VLAN filtering on a CB device:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
```

On the same device, configure a port that is connected to the PE device and will act as cascade port:

```
/interface bridge port-controller
set bridge=bridge1 cascade-ports=sfp-sfpplus1 switch=switch1
```

Last, on a PE device, simply configure a control port, which will be selected as an upstream port:

```
/interface bridge port-extender
set control-ports=sfp-sfpplus1 switch=switch1
```

Once PE and CB devices are connected, all interfaces that are on the same switch group (except for control ports) will be extended and can be further configured on a CB device. An automatic bridge port configuration will be applied on the CB device which adds all extended ports in a single bridge, this configuration can be modified afterward.

⚠ In order to exclude some port from being extended (e.g. for out-of-band management purposes), additionally, configure `excluded-ports` property.

⚠ Make sure not to include the `cascade-ports` and `control-ports` in any routing or bridging configurations. These ports are recommended only for a CB and PE usage.

Discovery and control protocols

Before frame forwarding on extended ports is possible, Controlling Bridge and Port Extender must discover each other and exchange with essential information.

CB and PE enabled devices are using a neighbor discovery protocol LLDP with specific Port Extension TLV. This allows CB and PE devices to advertise their support on cascade and control ports.

⚠ CB and PE configuration can override the neighbor discovery settings, for example, if a cascade port is not included in a neighbor discovery interface list, the LLDP messages will be still sent.

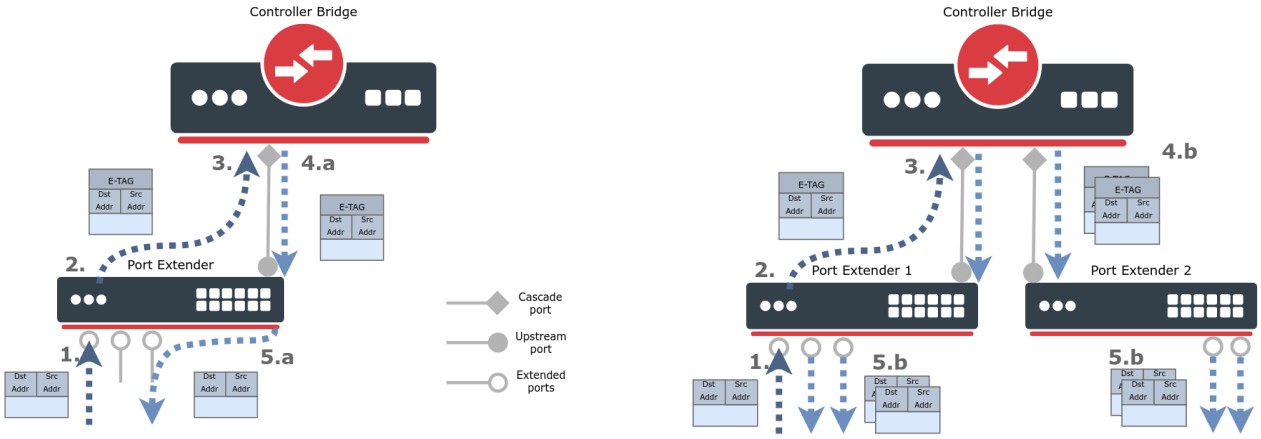
Once LLDP messages are exchanged between CB and PE, a Control and Status Protocol (CSP) over an Edge Control Protocol (ECP) will initiate. The CSP is used between CB and PE to assert control and receive status information from the associated PE - it assigns unique IDs for extended ports, controls data-path settings (e.g. port VLAN membership) and sends port status information (e.g. interface stats, PoE-out monitoring). The ECP provides a reliable and sequenced frame delivery (encoded with EtherType 0x8940).

⛔ The current CB implementation does not support any failover techniques. Once the CB device becomes unavailable, the PE devices will lose all the control and data forwarding rules.

Packet flow

To better understand the underlying principles of Controlling Bridge and Port Extender, a packet walkthrough is provided below:

1. An L2 packet is received on the extended port;
2. The Port Extender encapsulates the packet with an E-TAG header (EtherType 0x893F) and forwards it through an upstream port, towards the Controller Bridge. An E-TAG packet contains information regarding the PE source port ID. The PE device does not make any local switching decisions;
3. The Controller Bridge receives the E-TAG packet and knows exactly which extended interface received it. The CB then internally decapsulates the packet and proceed it through a regular switching decision (host learning, destination address lookup, VLAN filtering, etc.);
4. Once a switching decision is made, the CB will again encapsulate the original packet with an E-TAG and send it through a cascade port, towards Port Extender;
    a. For a single destination packet (unicast), the CB will include the PE destination port ID in the E-TAG and send it through a correct cascade port;
    b. For a multi-destination packet (broadcast, multicast or unknown-unicast), the CB will include a target group mark and source port ID in the E-TAG and send a single packet replica per every cascade port;
5. Once a PE device receives an E-TAG packet on the upstream port, PE decapsulates it and sends the original L2 packet through the extended port;
    a. For a single destination packet (unicast), the PE will send the packet only to the correct extended port;
    b. For a multi-destination packet (broadcast, multicast or unknown-unicast), the PE will send a single packet replica per every extended port (except for the source port where the packet was received).



Controller Bridge settings and monitoring

This section describes the Controller Bridge settings and monitoring options.

Sub-menu: `/interface bridge port-controller`

Property	Description
bridge (*name;* Default: **none**)	The bridge interface where ports will be extended. The CB will only enable when `bridge` and `switch` properties are specified, otherwise, it will be in a disabled state.
cascade-ports (*interfaces;* Default: **none**)	Interfaces that will act as cascade ports. A bonding interface with 802.3ad or balance-xor `mode` is also supported.
switch (*name;* Default: **none**)	The switch that will act as the CB and ensure the control and network traffic. The CB will only enable when `bridge` and `switch` properties are specified, otherwise, it will be in a disabled state.

After CB and PE devices are configured and connected, each PE device will be automatically visible on the device menu, use `print` and `monitor` commands to see more details.

```
[admin@Controller] > interface bridge port-controller device print
Flags: I - inactive
 0   name="pe1" pe-mac=64:D1:54:EB:AE:BC descr="MikroTik RouterOS 6.48beta35 (testing) CRS328-24P-4S+"
control-ports=pe1-sfpplus1,pe1-sfpplus2

 1   name="pe2" pe-mac=64:D1:54:C7:3A:58 descr="MikroTik RouterOS 6.48beta35 (testing) CRS326-24G-2S+"
control-ports=pe2-sfpplus1
[admin@Controller] > interface bridge port-controller device monitor pe2
                name: pe2
              status: active
  connected-via-ports: sfp-sfpplus1==pe1-sfpplus1,pe1-sfpplus2==pe2-sfpplus1
   connected-via-devs: controller,pe1
```

Sub-menu: `/interface bridge port-controller device`

Property	Description
connected-via-devs (*name*)	Shows the connected devices in the path from PE to CB.
connected-via-ports (*name*)	Shows the connection path from PE to CB.
control-ports (*interfaces*)	PE device control ports.
descr (*name*)	Short PE device description.
name (*name*)	Automatically assigned PE device name.
pe-mac (*MAC address*)	PE device MAC address.
status (*active* | *inactive*)	PE device status.

Additionally, each PE device interface can be monitored on the port menu, use `print` and `monitor` commands to see more details.

```
[admin@Controller] > interface bridge port-controller port print where !disabled
Flags: I - inactive, X - disabled, R - running, U - upstream-port, C - cascade-port
 #    NAME                                DEVICE
 0 I  pe1-ether1                          pe1
 1 R  pe1-ether2                          pe1
 2 R  pe1-ether3                          pe1
 3 R  pe1-ether4                          pe1
 4  U pe1-sfpplus1                        pe1
 5 RC pe1-sfpplus2                        pe1
 6 I  pe2-ether1                          pe2
 7 R  pe2-ether2                          pe2
 8 R  pe2-ether3                          pe2
 9 R  pe2-ether4                          pe2
10  U pe2-sfpplus1                        pe2
[admin@Controller] > interface bridge port-controller port monitor [find where !disabled]
          name: pe1-ether1 pe1-ether2 pe1-ether3 pe1-ether4 pe1-sfpplus1 pe1-sfpplus2 pe2-ether1 pe2-ether2
pe2-ether3 pe2-ether4 pe2-sfpplus1
         status: unknown    link-ok    link-ok    link-ok    no-link      link-ok      unknown    link-ok
link-ok    link-ok    no-link
          rate:            1Gbps      1Gbps      1Gbps      10Gbps       10Gbps                  1Gbps
1Gbps      1Gbps      10Gbps
    port-status: not-added  ok         ok         ok         ok           ok           not-added  ok
ok         ok         ok
          pcid:            457        458        459        480          481                     509
510        511        532
```

Sub-menu: `/interface bridge port-controller port`

Property	Description
device (*name*)	Automatically assigned PE device name.

name (*name*)	Automatically assigned PE port name.
pcid (*integer*)	Automatically assigned port identifier.
port-status (*dev-inactive* | *not-added* | *ok*)	PE port status.
rate (*bps*)	Data rate of the connection.
status (*link-ok* | *no-link* | *unknown*)	PE port link status.

The Controller Bridge can monitor the PoE-out related information from Port Extenders on the port poe menu, use `print` and `monitor` commands to see more details. For more information regarding PoE-out, please visit the PoE-out manual.

```
[admin@Controller] > interface bridge port-controller port poe print
 # NAME                                DEVICE
 0 pe1-ether1                          pe1
 1 pe1-ether2                          pe1
 2 pe1-ether3                          pe1
 3 pe1-ether4                          pe1
 4 pe1-ether5                          pe1
 5 pe1-ether6                          pe1
 6 pe1-ether7                          pe1
...
[admin@Controller] > interface bridge port-controller port poe monitor pe1-ether2,pe1-ether3
               name: pe1-ether2 pe1-ether3
     poe-out-status: powered-on powered-on
    poe-out-voltage: 52.8V      52.9V
    poe-out-current: 123mA      95mA
      poe-out-power: 6.4W       5W
```

Port Extender settings

This section describes the Port Extender settings.

Sub-menu: `/interface bridge port-extender`

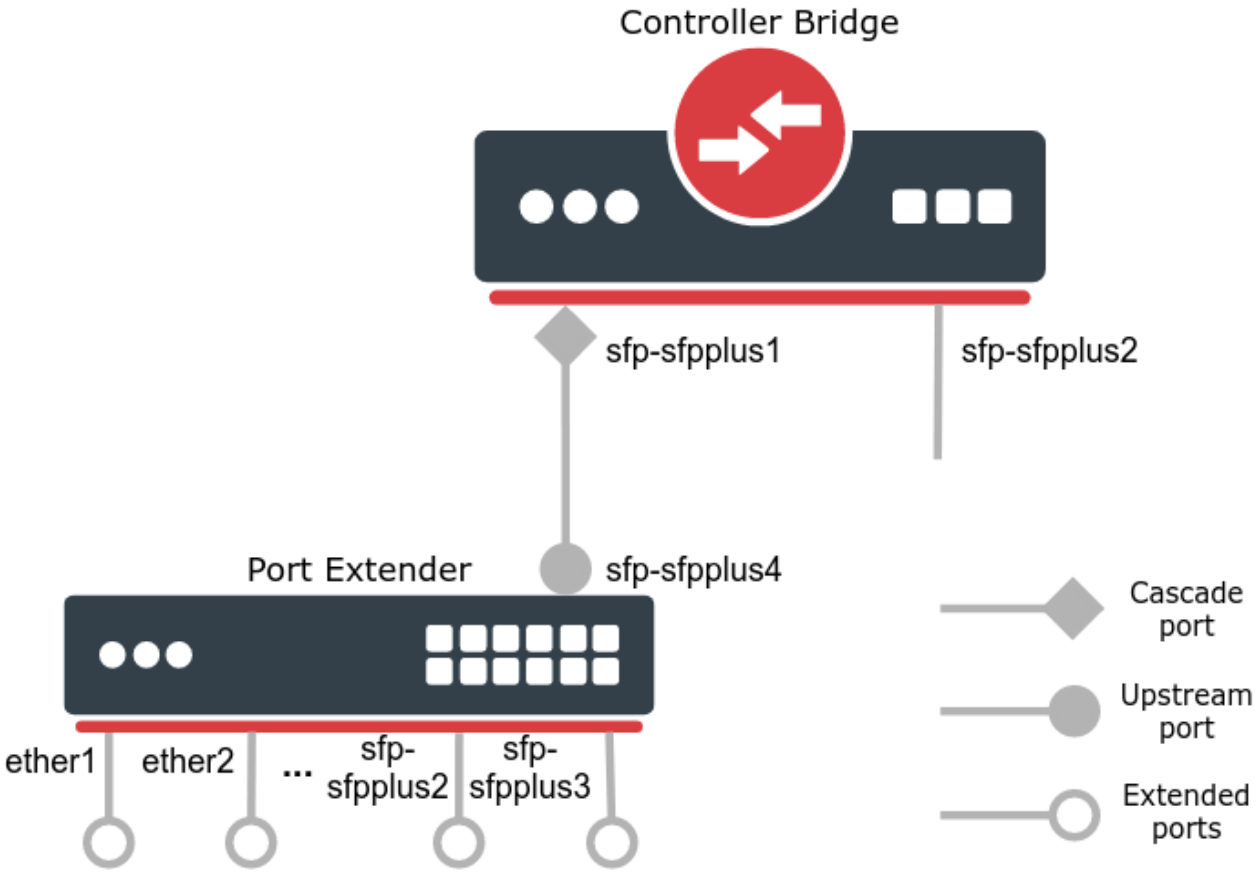
Property	Description
control-ports (*interfaces;* Default: **none**)	Interfaces that will either connect to the CB (upstream port) or connect other PE devices in series (cascade port). A bonding interface with 802.3ad or balance-xor `mode` is also supported.
excluded-ports (*interfaces ;* Default: **none**)	Interfaces that will not be extended.
switch (*name;* Default: **none**)	The switch that will act as the extender and ensure the control and network traffic. The PE will only enable when this property is specified, otherwise, it will be in a disabled state.

Configuration examples

Below are described the most common configuration examples. For CB and PE configuration to work properly, a bridge VLAN filtering needs to be enabled, so make sure to understand the filtering principles first - bridge VLAN filtering, bridge VLAN table.

Basic CB and PE configuration

In this example, a CRS317-1G-16S+ device is used as a Controller Bridge and CRS328-24P-4S+ as a Port Extender, see the connection scheme below.



First, configure the CB device. This can be done by adding a bridge interface with enabled VLAN filtering. Additionally, add any local interfaces to the same bridge, it allows to forward traffic between any local interfaces and extended interfaces. In this example, an sfp-sfpplus2 interface is added.

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=sfp-sfpplus2
```

To enable CB, specify the bridge, switch and at least one cascade port. Make sure that cascade ports are not included in the bridge or routing configurations. These ports are recommended only for a CB and PE usage.

```
/interface bridge port-controller
set bridge=bridge1 cascade-ports=sfp-sfpplus1 switch=switch1
```

To enable PE, configure control ports and switch. Additionally, configure one or multiple interfaces that should not be extended with `excluded-ports` property (e.g. for out-of-band management purposes). In this example, all switch ports will be extended.

```
/interface bridge port-extender
set control-ports=sfp-sfpplus4 switch=switch1
```

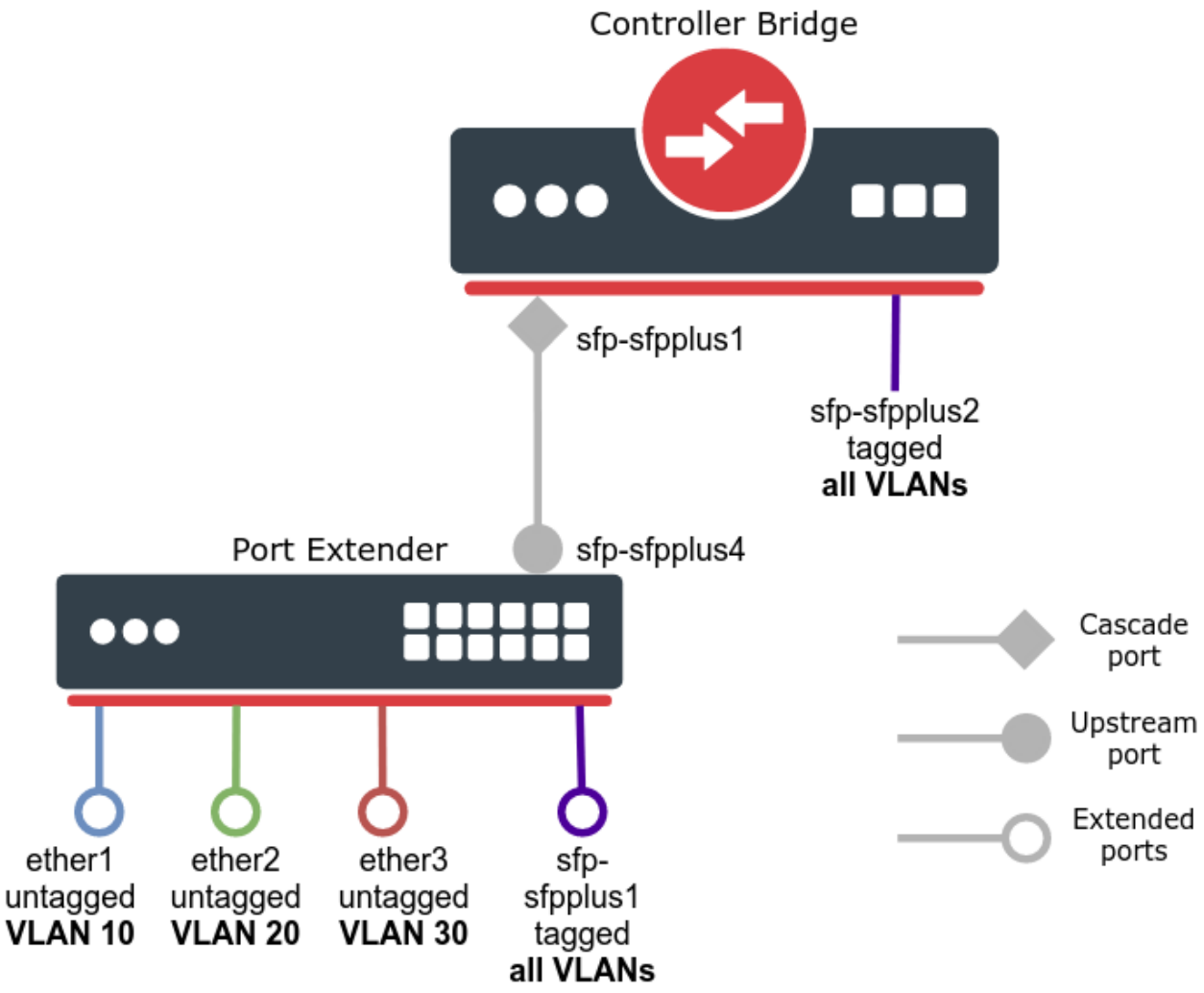
Once PE and CB devices finish the discovery and start the Control and Status Protocol (CSP), the RouterOS will permanently create new interfaces and add them into bridge on the CB device. Interfaces are named by the automatically assigned PE device name, plus the default interface name, these interface names can be modified afterwards. Note that control and excluded ports will be also displayed into the interface list, but they are not included into the bridge.

```
[admin@Controller_Bridge] > /interface print where name~"pe"
Flags: D - dynamic, X - disabled, R - running, S - slave
 #     NAME                                TYPE       ACTUAL-MTU L2MTU  MAX-L2MTU
 0  RS pe1-ether1                          extport          1500  1584
 1  RS pe1-ether2                          extport          1500  1584
 2  RS pe1-ether3                          extport          1500  1584
 3   S pe1-ether4                          extport          1500  1584
 4   S pe1-ether5                          extport          1500  1584
 5   S pe1-ether6                          extport          1500  1584
 6   S pe1-ether7                          extport          1500  1584
 7   S pe1-ether8                          extport          1500  1584
 8   S pe1-ether9                          extport          1500  1584
 9   S pe1-ether10                         extport          1500  1584
10   S pe1-ether11                         extport          1500  1584
11   S pe1-ether12                         extport          1500  1584
12   S pe1-ether13                         extport          1500  1584
13   S pe1-ether14                         extport          1500  1584
14   S pe1-ether15                         extport          1500  1584
15   S pe1-ether16                         extport          1500  1584
16   S pe1-ether17                         extport          1500  1584
17   S pe1-ether18                         extport          1500  1584
18   S pe1-ether19                         extport          1500  1584
19   S pe1-ether20                         extport          1500  1584
20   S pe1-ether21                         extport          1500  1584
21   S pe1-ether22                         extport          1500  1584
22   S pe1-ether23                         extport          1500  1584
23   S pe1-ether24                         extport          1500  1584
24  RS pe1-sfpplus1                        extport          1500  1584
25  RS pe1-sfpplus2                        extport          1500  1584
26  RS pe1-sfpplus3                        extport          1500  1584
27     pe1-sfpplus4                        extport          1500  1584
[admin@Controller_Bridge] > interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE             BRIDGE            HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0   H sfp-sfpplus2          bridge1           yes    1     0x80         10                 10       none
 1   H pe1-ether1            bridge1           yes    1     0x80         10                 10       none
 2   H pe1-ether2            bridge1           yes    1     0x80         10                 10       none
 3   H pe1-ether3            bridge1           yes    1     0x80         10                 10       none
 4 I H pe1-ether4            bridge1           yes    1     0x80         10                 10       none
 5 I H pe1-ether5            bridge1           yes    1     0x80         10                 10       none
 6 I H pe1-ether6            bridge1           yes    1     0x80         10                 10       none
 7 I H pe1-ether7            bridge1           yes    1     0x80         10                 10       none
 8 I H pe1-ether8            bridge1           yes    1     0x80         10                 10       none
 9 I H pe1-ether9            bridge1           yes    1     0x80         10                 10       none
10 I H pe1-ether10           bridge1           yes    1     0x80         10                 10       none
11 I H pe1-ether11           bridge1           yes    1     0x80         10                 10       none
12 I H pe1-ether12           bridge1           yes    1     0x80         10                 10       none
13 I H pe1-ether13           bridge1           yes    1     0x80         10                 10       none
14 I H pe1-ether14           bridge1           yes    1     0x80         10                 10       none
15 I H pe1-ether15           bridge1           yes    1     0x80         10                 10       none
16 I H pe1-ether16           bridge1           yes    1     0x80         10                 10       none
17 I H pe1-ether17           bridge1           yes    1     0x80         10                 10       none
18 I H pe1-ether18           bridge1           yes    1     0x80         10                 10       none
19 I H pe1-ether19           bridge1           yes    1     0x80         10                 10       none
20 I H pe1-ether20           bridge1           yes    1     0x80         10                 10       none
21 I H pe1-ether21           bridge1           yes    1     0x80         10                 10       none
22 I H pe1-ether22           bridge1           yes    1     0x80         10                 10       none
23 I H pe1-ether23           bridge1           yes    1     0x80         10                 10       none
24 I H pe1-ether24           bridge1           yes    1     0x80         10                 10       none
25   H pe1-sfpplus1          bridge1           yes    1     0x80         10                 10       none
26   H pe1-sfpplus2          bridge1           yes    1     0x80         10                 10       none
27   H pe1-sfpplus3          bridge1           yes    1     0x80         10                 10       none
```

Now the CRS317-1G-16S+ device has extended its ports using the CRS328-24P-4S+ device and packet forwarding can be done between all bridged ports.

Trunk and Access ports

In this example, untagged (access) and tagged (trunk) port configuration will be created on the Controller Bridge device, see the network diagram below.



First, configure the CB and PE devices, the configuration is identical to the previous example. Use this configuration for CB device.

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=sfp-sfpplus2
/interface bridge port-controller
set bridge=bridge1 cascade-ports=sfp-sfpplus1 switch=switch1
```

Use this configuration for PE device.

```
/interface bridge port-extender
set control-ports=sfp-sfpplus4 switch=switch1
```

After extended ports are successfully created and added to the bridge on the CB device, we can start configuring VLAN related properties. First, configure access ports to their respective VLAN ID using a `pvid` property. Use a `print` command in "`/interface bridge port`" menu to find out the exact interface name.

```
/interface bridge port
set [find interface=pe1-ether1] pvid=10
set [find interface=pe1-ether2] pvid=20
set [find interface=pe1-ether3] pvid=30
```

Then add bridge VLAN entries and specify tagged, untagged ports. Note that there are two tagged ports - local port named sfp-sfpplus2 and extended port named pe1-sfpplus1.

```
/interface bridge vlan
add bridge=bridge1 tagged=pe1-sfpplus1,sfp-sfpplus2 untagged=pe1-ether1 vlan-ids=10
add bridge=bridge1 tagged=pe1-sfpplus1,sfp-sfpplus2 untagged=pe1-ether2 vlan-ids=20
add bridge=bridge1 tagged=pe1-sfpplus1,sfp-sfpplus2 untagged=pe1-ether3 vlan-ids=30
```

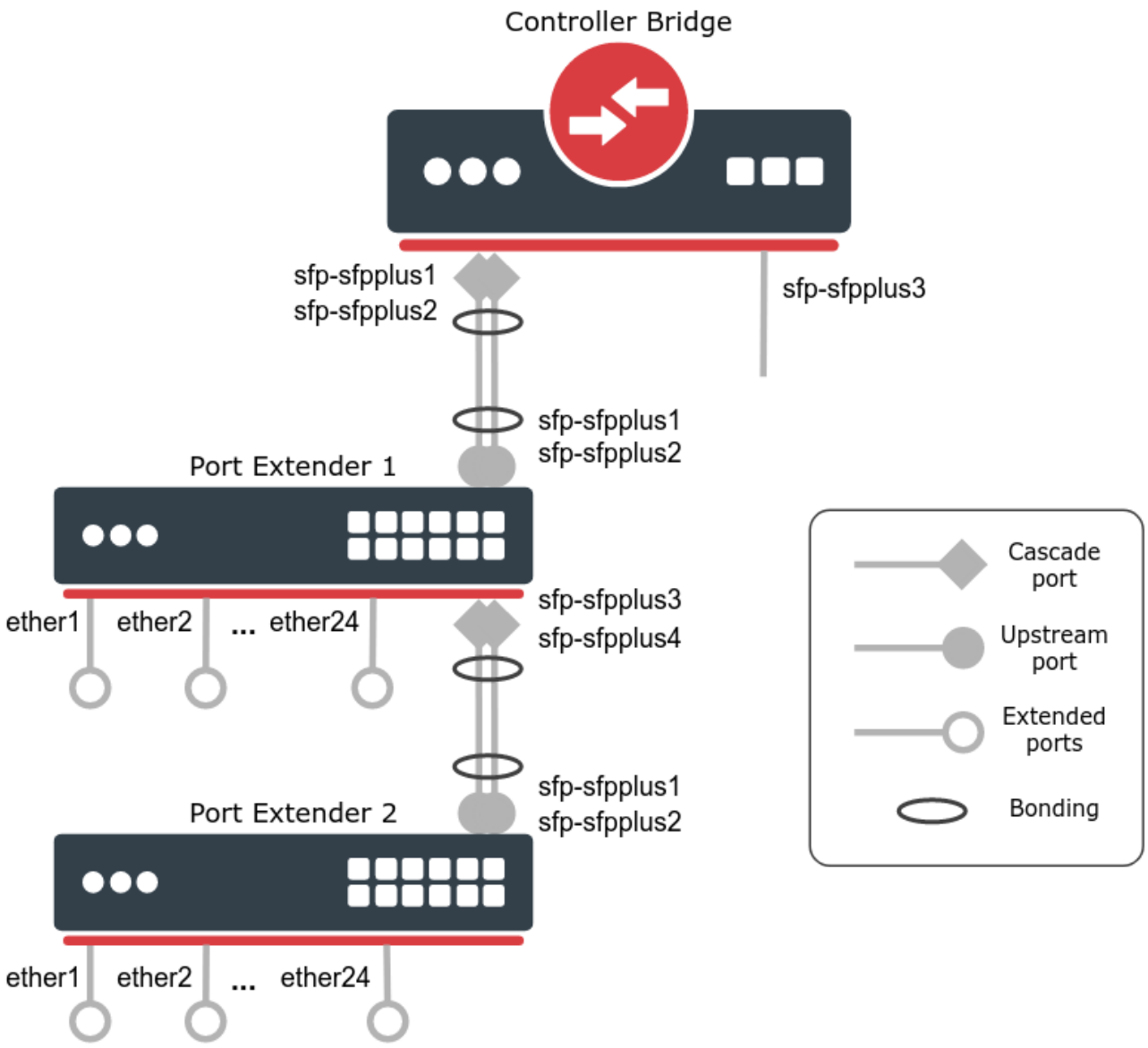
At this point VLANs are configured and devices should be able to communicate through the ports. However, it is recommended to go even a step further and apply some additional filtering options. Enable port `ingress-filtering` on local bridge ports and use frame filtering based on the packet type with `frame-types` setting.

```
/interface bridge port
set [find interface=pe1-ether1] frame-types=admit-only-untagged-and-priority-tagged
set [find interface=pe1-ether2] frame-types=admit-only-untagged-and-priority-tagged
set [find interface=pe1-ether3] frame-types=admit-only-untagged-and-priority-tagged
set [find interface=pe1-sfpplus1] frame-types=admit-only-vlan-tagged
set [find interface=sfp-sfpplus2] frame-types=admit-only-vlan-tagged ingress-filtering=yes
```

⚠ Port ingress VLAN filtering is not supported on extended ports.

Cascading multiple Port Extenders and using bonding interface

In this example, two PE devices (CRS328-24P-4S+ and CRS326-24G-2S+) will be added to the CB (CRS317-1G-16S+). To increase throughput for upstream and cascade ports, bonding interfaces will be created. See the network diagram below.



The CB and PE configuration is similar to the first example, the main difference is the bonding interface usage. First, configure the CB device - create a bonding interface for cascade port, create a bridge and add any needed local bridge ports, last enable the CB. Use the following commands:

```
/interface bonding
add mode=802.3ad name=bond1 slaves=sfp-sfpplus1,sfp-sfpplus2
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=sfp-sfpplus3
/interface bridge port-controller
set bridge=bridge1 cascade-ports=bond1 switch=switch1
```

Then configure the Port Extender 1 device. This device needs two bonding interfaces - the first one will be used as an upstream port and the second one will be a cascade port for the Port Extender 2 device. Additionally, configure one or multiple interfaces that should not be extended with `excluded-ports` property (e.g. for out-of-band management purposes). In this example, all switch ports will be extended.

```
/interface bonding
add mode=802.3ad name=bond1 slaves=sfp-sfpplus1,sfp-sfpplus2
add mode=802.3ad name=bond2 slaves=sfp-sfpplus3,sfp-sfpplus4
/interface bridge port-extender
set control-ports=bond1,bond2 switch=switch1
```

Last, configure the Port Extender 2 device - create a bonding interface and enable PE. Additionally, configure one or multiple `excluded-ports` if necessary. In this example, all switch ports will be extended.

```
/interface bonding
add mode=802.3ad name=bond1 slaves=sfp-sfpplus1,sfp-sfpplus2
/interface bridge port-extender
set control-ports=bond1 switch=switch1
```

Now the CRS317-1G-16S+ device has extended its ports with additional 48 Gigabit Ethernet ports and packet forwarding can be achieved between all bridged ports.

Use the `monitor` command in the device menu to see the PE device connection path. Also, use `print` command in the port menu to see which PE interfaces are used as upstream and cascade ports.

```
[admin@Controller_Bridge] > interface bridge port-controller device monitor [find]
                   name: pe1                   pe2
                 status: active                active
    connected-via-ports: bond1==pe1-cntrl-bond1 bond1==pe1-cntrl-bond1
                                               pe1-cntrl-bond2==pe2-cntrl-bond1
     connected-via-devs: controller            controller
                                               pe1
[admin@Controller_Bridge] > interface bridge port-controller port print where running or upstream-port
Flags: I - inactive, X - disabled, R - running, U - upstream-port, C - cascade-port
 #    NAME                                                DEVICE

 0 R  pe1-
ether2                                          pe1
 1 R  pe1-
ether3                                          pe1
 2 R  pe1-
ether4                                          pe1
 3  U pe1-
sfpplus1                                        pe1
 4  U pe1-
sfpplus2                                        pe1
 5 RC pe1-
sfpplus3                                        pe1
 6 RC pe1-
sfpplus4                                        pe1
 7 R  pe2-
ether1                                          pe2
 8 R  pe2-
ether2                                          pe2
 9 R  pe2-
ether3                                          pe2
10 R  pe2-
ether4                                          pe2
11  U pe2-
sfpplus1                                        pe2
12  U pe2-
sfpplus2                                        pe2
```

Configuration modification and removal

In certain situations, CB and PE device configuration needs to be adjusted (e.g. PE device needs new control ports) or removed completely. To modify the PE device configuration, all related PE device configuration should be removed from the CB device first. Only then the new configuration can be applied.

First, to remove PE configuration from CB, disable the PE using the following command:

```
/interface bridge port-extender set switch=none control-ports="" excluded-ports=""
```

Then, on the CB device, remove the related bridge and other RouterOS configuration where PE interfaces were used (e.g. see the export from "`/interface bridge port`" and "`/interface bridge vlan`" menus). For example, to remove all bridge ports from a specific PE device, use the command below:

```
/interface bridge port remove [find interface~"pe1"]
```

Once the configuration is removed, PE can be removed from the CB device list. This command will also automatically remove all the PE device interfaces from the CB interface list. In case some PE interface configuration is still applied on the CB, it will not be valid anymore. Use `print` command to find out the PE device name.

```
/interface bridge port-controller device remove [find name=pe1]
```

CRS1xx/2xx series switches examples

Summary

Basic use cases and configuration examples for Cloud Router Switch features.

> This article applies to CRS1xx and CRS2xx series switches and not to CRS3xx series switches. For CRS3xx series devices read the CRS3xx, CRS5xx series switches and CCR2116, CCR2216 manual.

Port switching

In order to set up port switching on CRS1xx/2xx series switches, check the Bridge Hardware Offloading page.

> It is possible to create multiple isolated switch group by using multiple bridges with enabled hardware offloading, this is possible only on CRS1xx /2xx series switches. For more complex setups (for example, VLAN filtering) you should use the port isolation feature instead.

Management access configuration

In general, switches are only supposed to forward packets by using the built-in switch chip, but not allow access to the device itself for security reasons. It is possible to use the device's serial port for management access, but in most cases, such an access method is not desired and access using an IP address is more suitable. In such cases, you will need to configure management access.

In all types of management access it is assumed that ports must be switched together, use the following commands to switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

You should also assign an IP address to the bridge interface so the device is reachable using an IP address (the device is also reachable using a MAC address):

```
/ip address
add address=192.168.88.1/24 interface=bridge1
```

Untagged

If invalid VLAN filtering is not enabled, management access to the device using tagged or untagged (**VLAN 0**) traffic is already allowed from any port, though this is not a good practice, this can cause security issues and can cause the device's CPU to be overloaded in certain situations (most commonly with a broadcast type of traffic).

If you are intending to use invalid VLAN filtering (which you should), then ports, from which you are going to access the switch, must be added to the VLAN table for untagged (**VLAN 0**) traffic, for example, in case you want to access the switch from **ether2**:

```
/interface ethernet switch vlan
add vlan-id=0 ports=ether2,switch1-cpu
```

Tagged

Allowing only tagged traffic to have management access to the device through a specific port is a much better practice. For example, to allow only **VLAN99** to access the device through **ether2** you should first add an entry to the VLAN table, which will allow the selected port and the CPU port (**switch1-cpu**) to forward the selected VLAN ID, therefore allowing management access:

```
/interface ethernet switch vlan
add ports=ether2,switch1-cpu vlan-id=99
```

Packets that will be sent out from the CPU, for example, ping replies will not have a VLAN tag, to solve this you need to specify which ports should always send out packets with a VLAN tag for a specific VLAN ID:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2,switch1-cpu vlan-id=99
```

After valid VLAN99 configuration has been set up, you can enable unknown/invalid VLAN filtering, which will not allow the management access through different ports than specified in the VLAN table:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether3,ether4,ether5
```

In this example VLAN99 will be used to access the device, a VLAN interface on the bridge must be created and an IP address must be assigned to it.

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT
```

VLAN

It is recommended to get Serial Console cable and test it before configuring VLANs because you may lose access to the CPU and/or the port you are connected to.

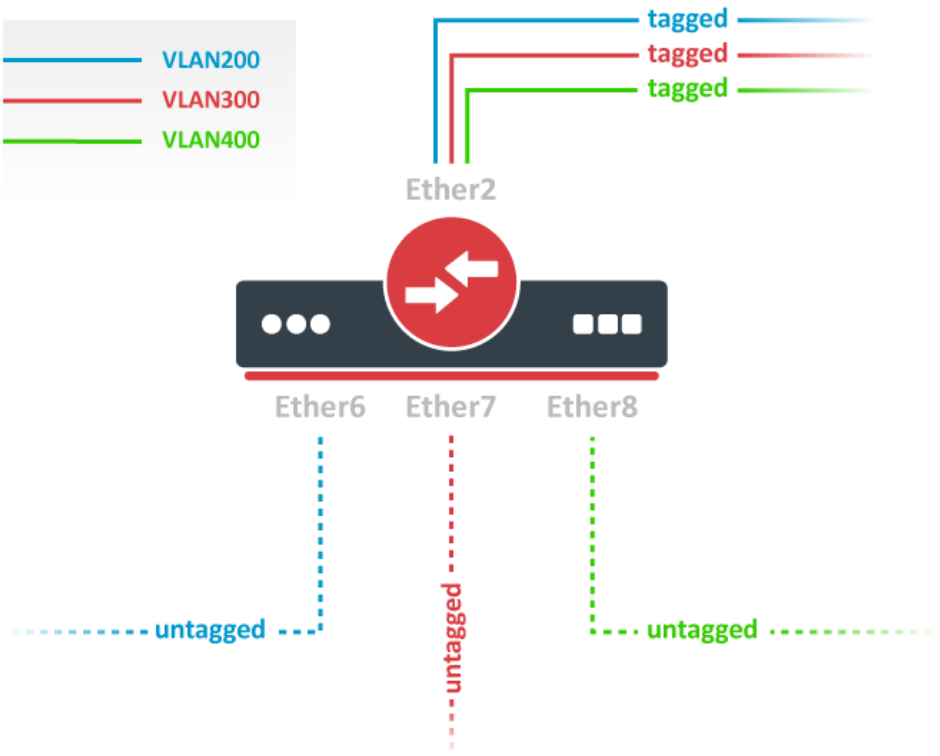
Some changes may take some time to take effect due to already learned MAC addresses. In such cases flushing Unicast Forwarding Database can help: `/interface ethernet switch unicast-fdb flush`

Multiple hardware offloaded bridge configuration is designed as fast and simple port isolation solution, but it limits part of VLAN functionality supported by CRS switch-chip. For advanced configurations use one bridge within CRS switch chip for all ports, configure VLANs and isolate port groups with port isolation profile configuration.

Port Based VLAN

For CRS3xx series devices, you must use bridge VLAN filtering, you can read more about it in the Bridge VLAN Filtering section.

Example 1 (Trunk and Access ports)



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Specify the VLAN ID that the switch must set on untagged (VLAN0) traffic for each access port:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```


When an entry is created under `/interface ethernet switch ingress-vlan-translation`, then the switch chip will add a VLAN tag on ingress frames on the specified port. To remove the VLAN tag on the same port for egress frames, an `/interface ethernet switch egress-vlan-tag` entry should be created for the same VLAN ID where only tagged ports are specified. If a specific VLAN is forwarded only between access ports, the `/interface ethernet switch egress-vlan-tag` entry should still be created without any tagged ports. Another option is to create extra entries under `/interface ethernet switch egress-vlan-translation` menu to set untagged (VLAN0) traffic.

You must also specify which VLANs should be sent out to the trunk port with a VLAN tag. Use the tagged-ports property to set up a trunk port:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=200
add tagged-ports=ether2 vlan-id=300
add tagged-ports=ether2 vlan-id=400
```

Add entries to the VLAN table to specify VLAN memberships for each port and each VLAN ID:

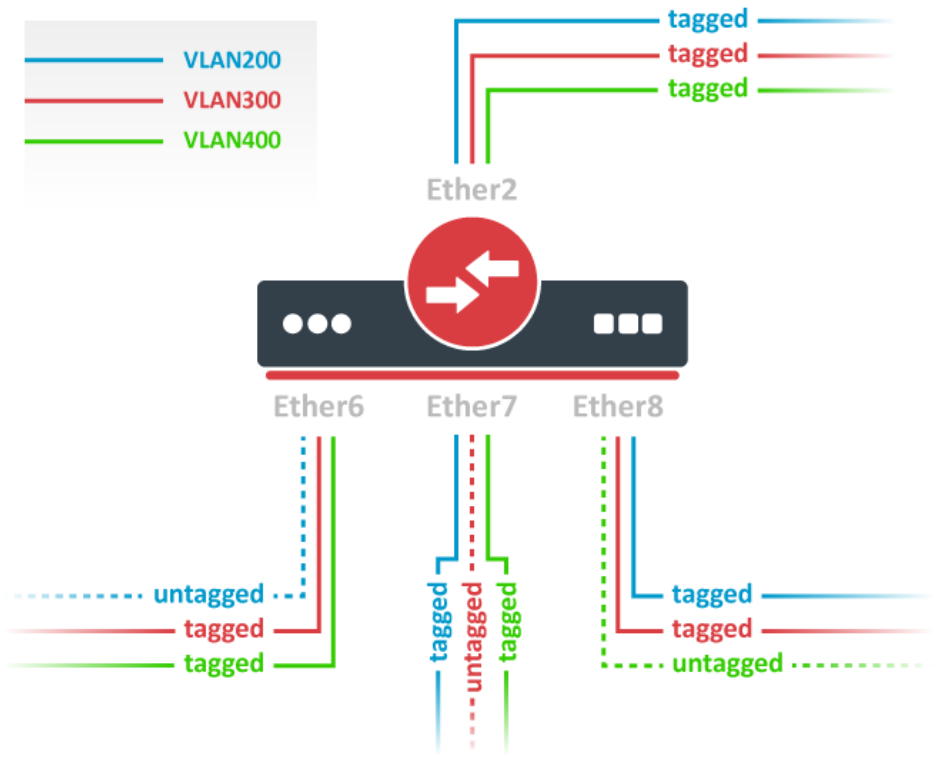
```
/interface ethernet switch vlan
add ports=ether2,ether6 vlan-id=200
add ports=ether2,ether7 vlan-id=300
add ports=ether2,ether8 vlan-id=400
```

After a valid VLAN configuration has been set up, you can enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```


It is possible to use the built-in switch chip and the CPU at the same time to create a Switch-Router setup, where a device acts as a switch and as a router at the same time. You can find a configuration example in the CRS-Router guide.

Example 2 (Trunk and Hybrid Ports)



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Specify the VLAN ID that the switch must set on untagged (VLAN0) traffic for each access port:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```

By specifying ports as tagged-ports, the switch will always send out packets as tagged packets with the corresponding VLAN ID. Add appropriate entries according to the diagram above:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2,ether7,ether8 vlan-id=200
add tagged-ports=ether2,ether6,ether8 vlan-id=300
add tagged-ports=ether2,ether6,ether7 vlan-id=400
```

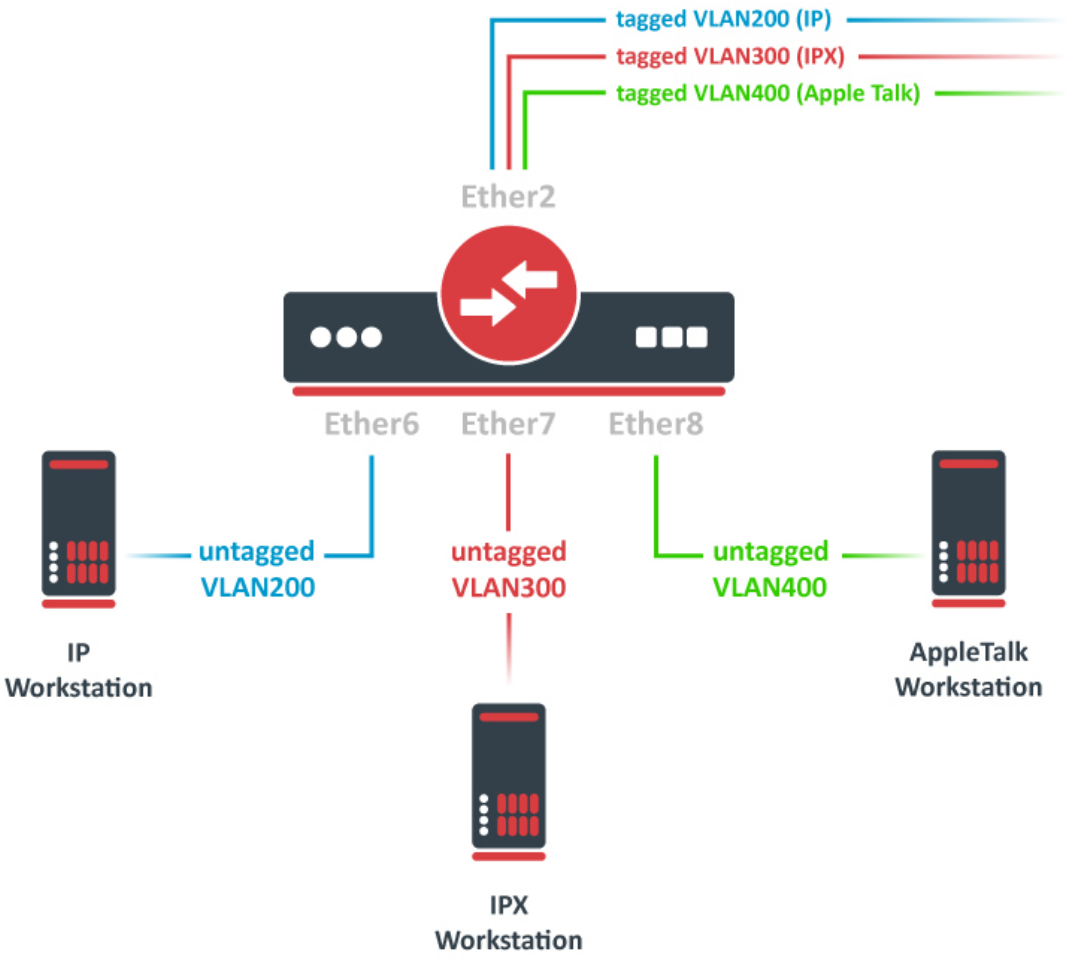
Add entries to the VLAN table to specify VLAN memberships for each port and each VLAN ID:

```
/interface ethernet switch vlan
add ports=ether2,ether6,ether7,ether8 vlan-id=200 learn=yes
add ports=ether2,ether6,ether7,ether8 vlan-id=300 learn=yes
add ports=ether2,ether6,ether7,ether8 vlan-id=400 learn=yes
```

After a valid VLAN configuration has been set up, you can enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```

Protocol Based VLAN



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Set VLAN for IP and ARP protocols:

```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=arp set-customer-vid-for=all new-customer-vid=0
add port=ether6 protocol=arp set-customer-vid-for=all new-customer-vid=200
add port=ether2 protocol=ip set-customer-vid-for=all new-customer-vid=0
add port=ether6 protocol=ip set-customer-vid-for=all new-customer-vid=200
```

Set VLAN for IPX protocol:

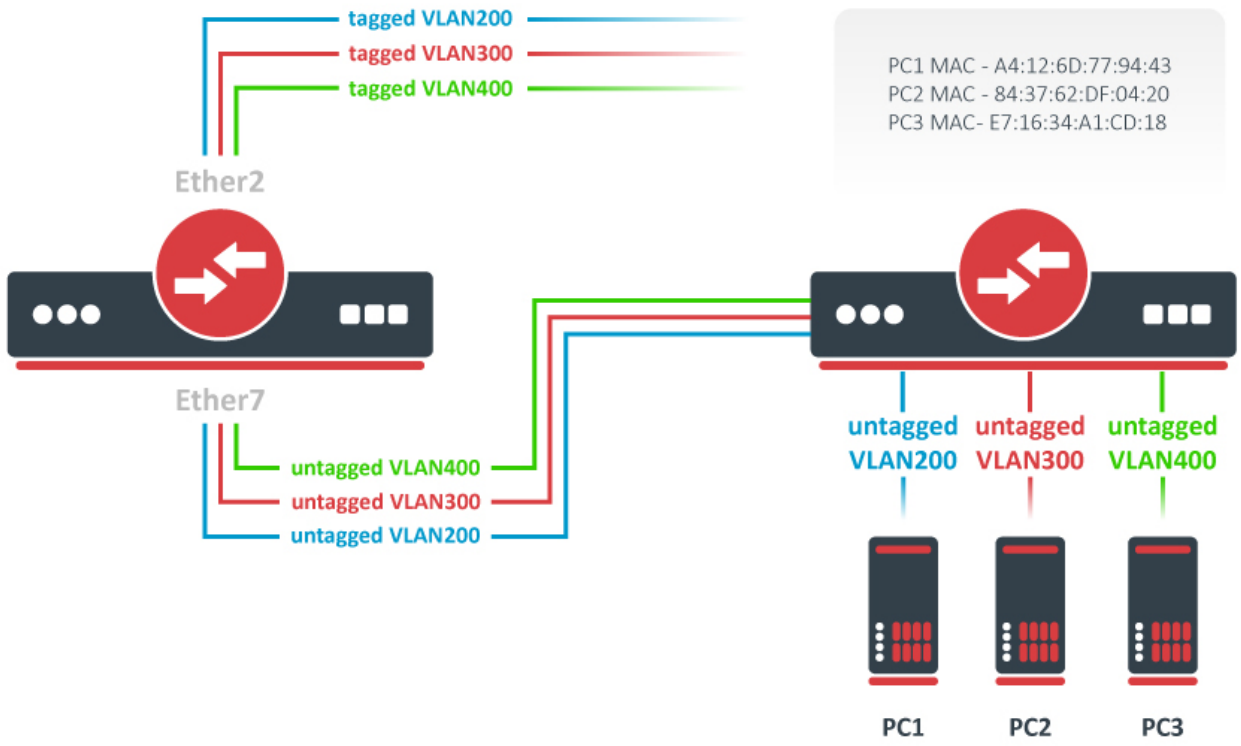
```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=ipx set-customer-vid-for=all new-customer-vid=0
add port=ether7 protocol=ipx set-customer-vid-for=all new-customer-vid=300
```

Set VLAN for AppleTalk AARP and AppleTalk DDP protocols:

```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=0x80F3 set-customer-vid-for=all new-customer-vid=0
add port=ether8 protocol=0x80F3 set-customer-vid-for=all new-customer-vid=400
add port=ether2 protocol=0x809B set-customer-vid-for=all new-customer-vid=0
add port=ether8 protocol=0x809B set-customer-vid-for=all new-customer-vid=400
```

MAC Based VLAN

Internally all MAC addresses in MAC-based VLANs are hashed, certain MAC addresses can have the same hash, which will prevent a MAC address from being loaded into the switch chip if the hash matches with a hash from a MAC address that has been already loaded, for this reason, it is recommended to use Port bases VLANs in combination with MAC-based VLANs. This is a hardware limitation.



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
```

Enable MAC-based VLAN translation on access port:

```
/interface ethernet switch port
set ether7 allow-fdb-based-vlan-translate=yes
```

Add MAC-to-VLAN mapping entries in the MAC-based VLAN table:

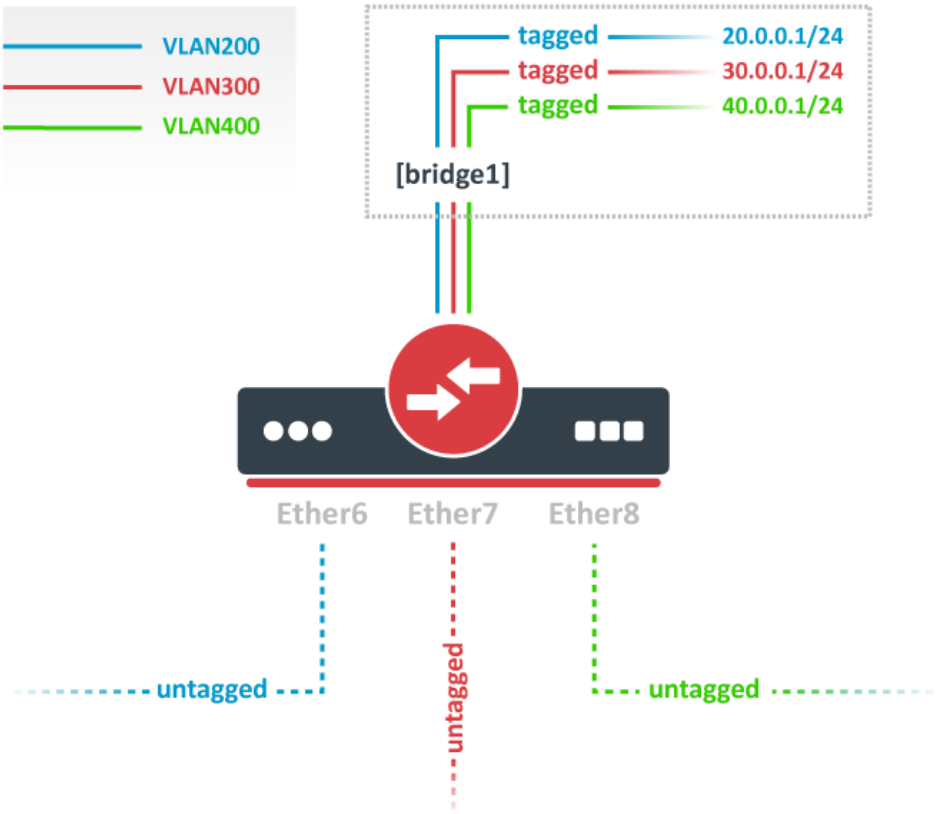
```
/interface ethernet switch mac-based-vlan
add src-mac=A4:12:6D:77:94:43 new-customer-vid=200
add src-mac=84:37:62:DF:04:20 new-customer-vid=300
add src-mac=E7:16:34:A1:CD:18 new-customer-vid=400
```

Add VLAN200, VLAN300, and VLAN400 tagging on the ether2 port to create it as a VLAN trunk port:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=200
add tagged-ports=ether2 vlan-id=300
add tagged-ports=ether2 vlan-id=400
```

Additionally, add entries to the VLAN table, specify VLAN membership for each port, and enable unknown/invalid VLAN filtering, see an example below - U nknown/Invalid VLAN filtering. This is required for network setups where more interfaces are added to the bridge, as it allows to define VLAN boundaries.

InterVLAN Routing



InterVLAN routing configuration consists of two main parts – VLAN tagging in switch-chip and routing in RouterOS. This configuration can be used in many applications by combining it with DHCP server, Hotspot, PPP, and other features for each VLAN.

Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Set VLAN tagging on CPU port for all VLANs to make packets tagged before they are routed:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=switch1-cpu vlan-id=200
add tagged-ports=switch1-cpu vlan-id=300
add tagged-ports=switch1-cpu vlan-id=400
```

add ingress VLAN translation rules to ensure that the correct VLAN ID assignment is done on access ports:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```

Create the VLAN interfaces on top of the bridge interface:

```
/interface vlan
add name=VLAN200 interface=bridge1 vlan-id=200
add name=VLAN300 interface=bridge1 vlan-id=300
add name=VLAN400 interface=bridge1 vlan-id=400
```

Make sure the VLAN interfaces are created on top of the bridge interface instead of any of the physical interfaces. If the VLAN interfaces are created on a slave interface, then the packet might not be received correctly and therefore routing might fail. More detailed information can be found in the VLAN interface on a slave interface manual page.

Add IP addresses on created VLAN interfaces. In this example, three 192.168.x.1 addresses are added to VLAN200, VLAN300, and VLAN400 interfaces:

```
/ip address
add address=192.168.20.1/24 interface=VLAN200
add address=192.168.30.1/24 interface=VLAN300
add address=192.168.40.1/24 interface=VLAN400
```

Unknown/Invalid VLAN filtering

VLAN membership is defined in the VLAN table. Adding entries with VLAN ID and ports makes that VLAN traffic valid on those ports. After a valid VLAN configuration has been set up, unknown/invalid VLAN filtering can be enabled. This VLAN filtering configuration example applies to the InterVLAN Routing s etup.

```
/interface ethernet switch vlan
add ports=switch1-cpu,ether6 vlan-id=200
add ports=switch1-cpu,ether7 vlan-id=300
add ports=switch1-cpu,ether8 vlan-id=400
```

- Option 1: disable invalid VLAN forwarding on specific ports (more common):

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```

- Option 2: disable invalid VLAN forwarding on all ports:

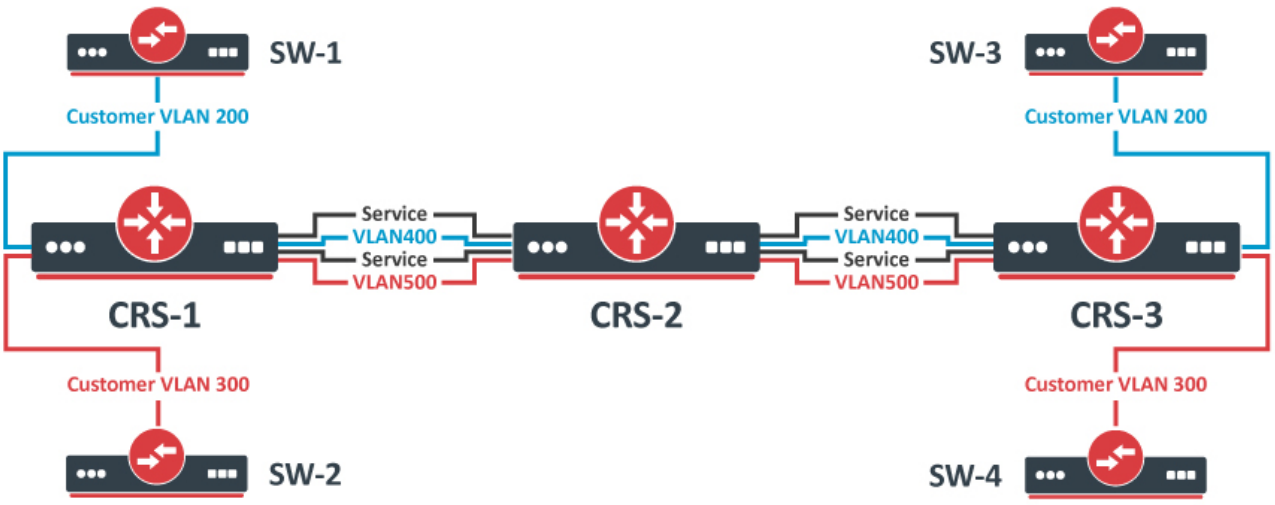
```
/interface ethernet switch
set forward-unknown-vlan=no
```

Using multiple bridges on a single switch chip with enabled unknown/invalid VLAN filtering can cause unexpected behavior. You should always use a single bridge configuration whenever using VLAN filtering. If port isolation is required, then port isolation feature should be used instead of using multiple bridges.

VLAN Tunneling (Q-in-Q)

This example covers a typical VLAN tunneling use case where service provider devices add another VLAN tag for independent forwarding in the meantime allowing customers to use their own VLANs.

This example contains only the Service VLAN tagging part. It is recommended to additionally set Unknown/Invalid VLAN filtering configuration on ports.



CRS-1: The first switch on the edge of the service provider network has to properly identify traffic from customer VLAN id on port and assign new service VLAN id with ingress VLAN translation rules. VLAN trunk port configuration for service provider VLAN tags is in the same *egress-vlan-tag* table. The main difference from basic Port-Based VLAN configuration is that the CRS switch-chip has to be set to do forwarding according to service (*outer*) VLAN id instead of customer (*inner*) VLAN id.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether9 hw=yes

/interface ethernet switch ingress-vlan-translation
add customer-vid=200 new-service-vid=400 ports=ether1
add customer-vid=300 new-service-vid=500 ports=ether2

/interface ethernet switch egress-vlan-tag
add tagged-ports=ether9 vlan-id=400
add tagged-ports=ether9 vlan-id=500

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CRS-2: The second switch in the service provider network require only switched ports to do forwarding according to service (*outer*) VLAN id instead of customer (*inner*) VLAN id.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether9 hw=yes
add bridge=bridge1 interface=ether10 hw=yes

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CRS-3: The third switch has a similar configuration to CRS-1:

- Ports in a switch group using a bridge;
- Ingress VLAN translation rules to define new service VLAN assignments on ports;
- tagged-ports for service provider VLAN trunks;
- CRS switch-chip set to use service VLAN id in switching lookup.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether10 hw=yes

/interface ethernet switch ingress-vlan-translation
add customer-vid=200 new-service-vid=400 ports=ether3
add customer-vid=300 new-service-vid=500 ports=ether4

/interface ethernet switch egress-vlan-tag
add tagged-ports=ether10 vlan-id=400
add tagged-ports=ether10 vlan-id=500

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CVID Stacking

It is possible to use CRS1xx/CRS2xx series switches for CVID Stacking setups. CRS1xx/CRS2xx series switches are capable of VLAN filtering based on the outer tag of tagged packets that have two CVID tags (double CVID tag), these switches are also capable of adding another CVID tag on top of an existing CVID tag (CVID Stacking). For example, in a setup where **ether1** is receiving tagged packets with CVID 10, but it is required that **ether2** sends out these packets with another tag CVID 20 (VLAN10 inside VLAN20) while filtering out any other VLANs, the following must be configured:

Switch together **ether1** and **ether2**:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
```

Set the switch to filter VLANs based on service tag (0x88a8):

```
/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

Add a service tag SVID 20 to packets that have a CVID 10 tag on **ether1**:

```
/interface ethernet switch ingress-vlan-translation
add customer-vid=10 new-service-vid=20 ports=ether1
```

Specify **ether2** as the tagged/trunk port for SVID 20:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=20
```

Allow **ether1** and **ether2** to forward SVID 20:

```
/interface ethernet switch vlan
add ports=ether1,ether2 vlan-id=20
```

Override the SVID EtherType (0x88a8) to CVID EtherType (0x8100) on **ether2**:

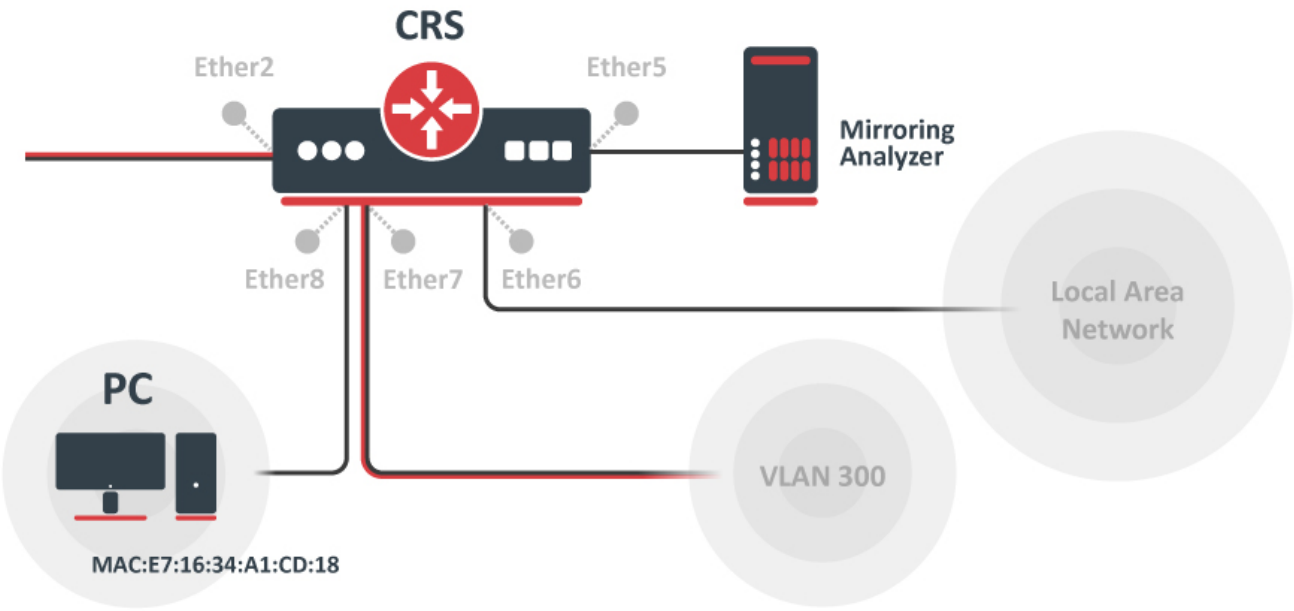
```
/interface ethernet switch port
set ether2 egress-service-tpid-override=0x8100 ingress-service-tpid-override=0x8100
```

Enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether1,ether2
```

⚠ Since the switch is set to look up VLAN ID based on service tag, which is overridden with a different EtherType, then VLAN filtering is only done on the outer tag of a packet, the inner tag is not checked.

Mirroring



The Cloud Router Switches support three types of mirroring. Port-based mirroring can be applied to any switch-chip ports, VLAN-based mirroring works for all specified VLANs regardless of switch-chip ports, and MAC-based mirroring copies traffic sent or received from specific device reachable from the port configured in Unicast Forwarding Database.

Port-Based Mirroring

The first configuration sets the ether5 port as a mirror0 analyzer port for both ingress and egress mirroring, mirrored traffic will be sent to this port. Port-based ingress and egress mirroring are enabled from the ether6 port.

```
/interface ethernet switch
set ingress-mirror0=ether5 egress-mirror0=ether5

/interface ethernet switch port
set ether6 ingress-mirror-to=mirror0 egress-mirror-to=mirror0
```

VLAN Based Mirroring

The second example requires ports to be switched in a group. Mirroring configuration sets ether5 port as a mirror0 analyzer port and sets mirror0 port to be used when mirroring from VLAN occurs. VLAN table entry enables mirroring only for VLAN 300 traffic between ether2 and ether7 ports.

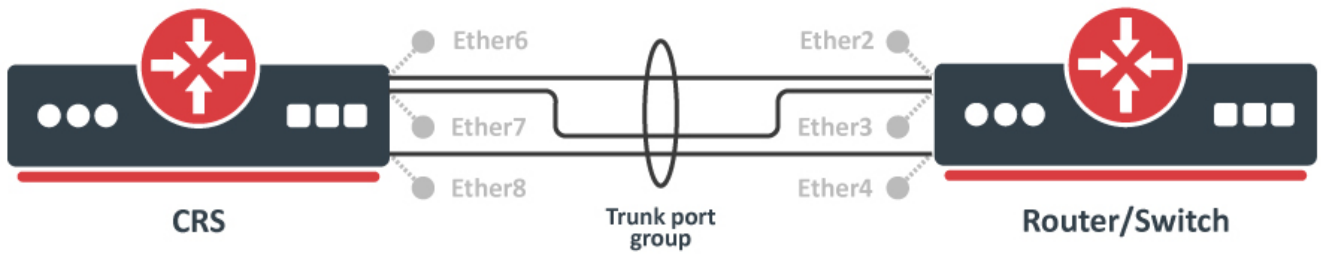
```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether7 hw=yes

/interface ethernet switch
set ingress-mirror0=ether5 vlan-uses=mirror0

/interface ethernet switch vlan
add ports=ether2,ether7 vlan-id=300 learn=yes ingress-mirror=yes
```

MAC Based Mirroring

The third configuration also requires ports to be switched in a group. Mirroring configuration sets ether5 port as a mirror0 analyzer port and sets mirror0 port to be used when mirroring from Unicast Forwarding database occurs. The entry from the Unicast Forwarding database enables mirroring for packets with source or destination MAC address E7:16:34:A1:CD:18 from ether8 port.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether8 hw=yes

/interface ethernet switch
set ingress-mirror0=ether5 fdb-uses=mirror0

/interface ethernet switch unicast-fdb
add port=ether8 mirror=yes svl=yes mac-address=E7:16:34:A1:CD:18
```

Trunking



The Trunking in the Cloud Router Switches provides static link aggregation groups with hardware automatic failover and load balancing. IEEE802.3ad and IEEE802.1ax compatible Link Aggregation Control Protocol is not supported yet. Up to 8 Trunk groups are supported with up to 8 Trunk member ports per Trunk group.

Configuration requires a group of switched ports and an entry in the Trunk table:

```
/interface bridge
add name=bridge1 protocol-mode=none
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes

/interface ethernet switch trunk
add name=trunk1 member-ports=ether6,ether7,ether8
```

This example also shows proper bonding configuration in RouterOS on the other end:

```
/interface bonding
add name=bonding1 slaves=ether2,ether3,ether4 mode=balance-xor transmit-hash-policy=layer-2-and-3
```

⚠ You can find a working example for trunking and port-based VLANs on CRS VLANs with Trunks page.

Bridge (R)STP is not aware of the underlying switch trunking configuration and some trunk ports can move to discarding or blocking state. When trunking member ports are connected to other bridges, you should either disable the (R)STP or filter out any BPDU between trunked devices (e. g. with ACL rules).

Limited MAC Access per Port

Disabling MAC learning and configuring static MAC addresses gives the ability to control what exact devices can communicate to CRS1xx/2xx switches and through them.

Configuration requires a group of switched ports, disabled MAC learning on those ports, and static UFDB entries:

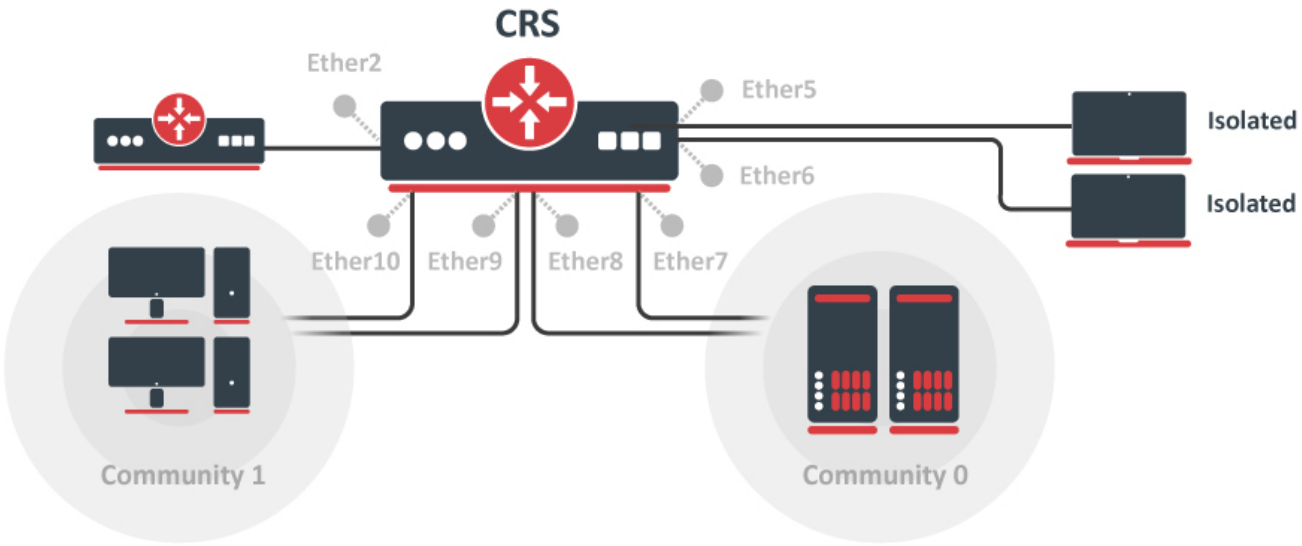
```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes learn=no unknown-unicast-flood=no
add bridge=bridge1 interface=ether7 hw=yes learn=no unknown-unicast-flood=no

/interface ethernet switch unicast-fdb
add mac-address=4C:5E:0C:00:00:01 port=ether6 svl=yes
add mac-address=D4:CA:6D:00:00:02 port=ether7 svl=yes

/interface ethernet switch acl
add action=drop src-mac-addr-state=sa-not-found src-ports=ether6,ether7 table=egress
add action=drop src-mac-addr-state=static-station-move src-ports=ether6,ether7 table=egress
```

CRS1xx/2xx switches also allow to learn one dynamic MAC per port to ensure only one end-user device is connected no matter its MAC address:

```
/interface ethernet switch port
set ether6 learn-limit=1
set ether7 learn-limit=1
```

Isolation

Port Level Isolation



Port-level isolation is often used for Private VLAN, where:

- One or multiple uplink ports are shared among all users for accessing the gateway or router.
- Port group Isolated Ports is for guest users. Communication is through the uplink ports only.
- Port group Community 0 is for department A. Communication is allowed between the group members and through uplink ports.
- Port group Community X is for department X. Communication is allowed between the group members and through uplink ports.

The Cloud Router Switches use port-level isolation profiles for Private VLAN implementation:

- Uplink ports – port-level isolation profile 0
- Isolated ports – port-level isolation profile 1
- Community 0 ports - port-level isolation profile 2
- Community X (X <= 30) ports - port-level isolation profile X

This example requires a group of switched ports. Assume that all ports used in this example are in one switch group.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
add bridge=bridge1 interface=ether9 hw=yes
add bridge=bridge1 interface=ether10 hw=yes
```

The first part of port isolation configuration is setting the Uplink port – set a port profile to 0 for ether2:

```
/interface ethernet switch port
set ether2 isolation-leakage-profile-override=0
```

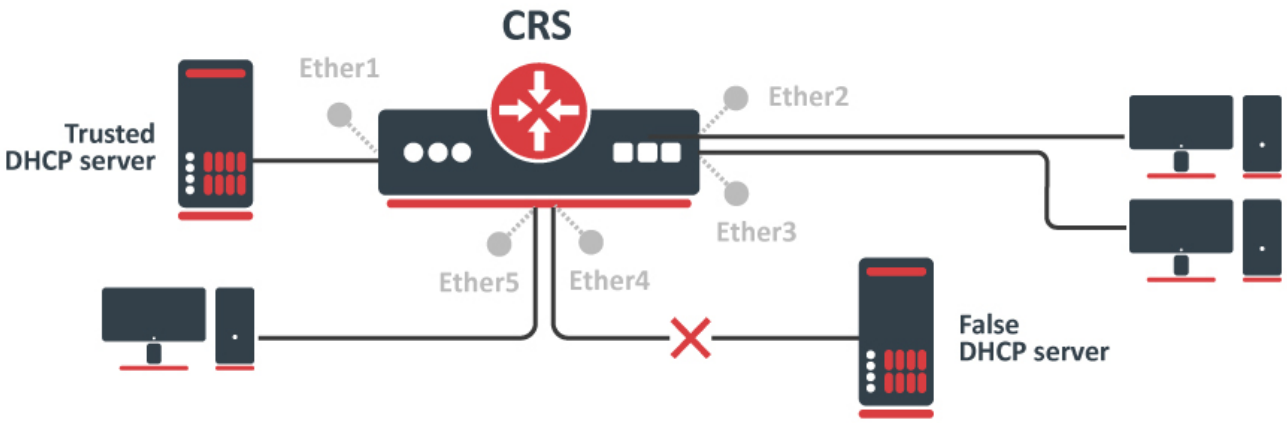
Then continue with setting isolation profile 1 to all isolated ports and adding the communication port for port isolation profile 1:

```
/interface ethernet switch port
set ether5 isolation-leakage-profile-override=1
set ether6 isolation-leakage-profile-override=1

/interface ethernet switch port-isolation
add port-profile=1 ports=ether2 type=dst
```

Configuration to set Community 2 and Community 3 ports is similar:

```
/interface ethernet switch port
set ether7 isolation-leakage-profile-override=2
set ether8 isolation-leakage-profile-override=2

/interface ethernet switch port-isolation
add port-profile=2 ports=ether2,ether7,ether8 type=dst

/interface ethernet switch port
set ether9 isolation-leakage-profile-override=3
set ether10 isolation-leakage-profile-override=3

/interface ethernet switch port-isolation
add port-profile=3 ports=ether2,ether9,ether10 type=dst
```

Protocol Level Isolation



Protocol level isolation on CRS switches can be used to enhance network security. For example, restricting DHCP traffic between the users and allowing it only to trusted DHCP server ports can prevent security risks like DHCP spoofing attacks. The following example shows how to configure it on CRS.

Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

Set the same Community port profile for all DHCP client ports. Community port profile numbers are from 2 to 30.

```
/interface ethernet switch port
set ether2 isolation-leakage-profile-override=2
set ether3 isolation-leakage-profile-override=2
set ether4 isolation-leakage-profile-override=2
set ether5 isolation-leakage-profile-override=2
```

And configure port isolation/leakage profile for selected Community (2) to allow DHCP traffic destined only to port where the trusted DHCP server is located. registration status and traffic-type properties have to be set empty in order to apply restrictions only for DHCP protocol.

```
/interface ethernet switch port-isolation
add port-profile=2 protocol-type=dhcpv4 type=dst forwarding-type=bridged ports=ether1 registration-status=""
traffic-type=""
```

Quality of Service (QoS)

QoS configuration schemes

MAC based traffic scheduling and shaping: [MAC address in UFDB] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

VLAN based traffic scheduling and shaping: [VLAN id in VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

Protocol based traffic scheduling and shaping: [Protocol in Protocol VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

PCP/DEI based traffic scheduling and shaping: [Switch port PCP/DEI mapping] -> [Priority] -> [Queue] -> [Shaper]

DSCP based traffic scheduling and shaping: [QoS DSCP mapping] -> [Priority] -> [Queue] -> [Shaper]

MAC-based traffic scheduling using internal Priority

In Strict Priority scheduling mode, the highest priority queue is served first. The queue number represents the priority and the queue with the highest queue number has the highest priority. Traffic is transmitted from the highest priority queue until the queue is empty, and then moves to the next highest priority queue, and so on. If no congestion is present at the egress port, a packet is transmitted as soon as it is received. If congestion occurs in the port where high-priority traffics keeps coming, the lower priority queues starve.

On all CRS switches the scheme where MAC based egress traffic scheduling is done according to internal Priority would be following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue];
In this example, host1 (E7:16:34:00:00:01) and host2 (E7:16:34:00:00:02) will have higher priority 1 and the rest of the hosts will have lower priority 0 for transmitted traffic on port ether7. Note that CRS has a maximum of 8 queues per port.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entries to match specific MACs on ether7 and apply the QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:00:00:01 port=ether7 qos-group=group1 svl=yes
add mac-address=E7:16:34:00:00:02 port=ether7 qos-group=group1 svl=yes
```

Configure ether7 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 qos-scheme-
precedence=da-based
```

MAC-based traffic shaping using internal Priority

The scheme where MAC based traffic shaping is done according to internal Priority would be following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper];
In this example, unlimited traffic will have priority 0 and limited traffic will have priority 1 with a bandwidth limit 10Mbit. Note that CRS has maximum of 8 queues per port.

Create a group of ports for switching:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create a QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entry to match specific MAC on ether8 and apply QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:A1:CD:18 port=ether8 qos-group=group1 svl=yes
```

Configure ether8 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 qos-scheme-precedence=da-based
```

Apply bandwidth limit for queue1 on ether8:

```
/interface ethernet switch shaper
add port=ether8 rate=10M target=queue1
```

If the CRS switch supports Access Control List, this configuration is simpler:

```
/interface ethernet switch acl policer
add name=policer1 yellow-burst=100k yellow-rate=10M

/interface ethernet switch acl
add mac-dst-address=E7:16:34:A1:CD:18 policer=policer1
```

VLAN based traffic scheduling + shaping using internal Priorities

The best practice is to assign lower internal QoS Priority for traffic limited by shaper to make it also less important in the Strict Priority scheduler. (higher priority should be more important and unlimited)

In this example:
Switch port ether6 is using a shaper to limit the traffic that comes from ether7 and ether8.
When the link has reached its capacity, the traffic with the highest priority will be sent out first.
VLAN10 -> QoS group0 = lowest priority
VLAN20 -> QoS group1 = normal priority
VLAN30 -> QoS group2 = highest priority

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create QoS groups for use in VLAN table.

```
/interface ethernet switch qos-group
add name=group0 priority=0
add name=group1 priority=1
add name=group2 priority=2
```

Add VLAN entries to apply QoS groups for certain VLANs.

```
/interface ethernet switch vlan
add ports=ether6,ether7,ether8 qos-group=group0 vlan-id=10
add ports=ether6,ether7,ether8 qos-group=group1 vlan-id=20
add ports=ether6,ether7,ether8 qos-group=group2 vlan-id=30
```

Configure ether6, ether7, and ether8 port queues to work according to Strict Priority and QoS schemes only for VLAN-based QoS.

```
/interface ethernet switch port
set ether6 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
```

Apply bandwidth limit on ether6.

```
/interface ethernet switch shaper
add port=ether6 rate=10M
```

PCP based traffic scheduling

By default, CRS1xx/CRS2xx series devices will ignore the PCP/CoS/802.1p value and forward packets based on FIFO (First-In-First-Out) manner. When the device's internal queue is not full, then packets are sent in FIFO manner, but as soon as a queue is filled, then higher priority traffic can be sent out first. Let us consider a scenario when **ether1** and **ether2** are forwarding data to **ether3**, but when **ether3** is congested, then packets are going to be scheduled, we can configure the switch to hold the lowest priority packets until all higher priority packets are sent out, this is a very common scenario for VoIP type setups, where some traffic needs to be prioritized.

To achieve such a behavior, switch together **ether1**, **ether2,** and **ether3** ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
```

Enable **Strict Policy** for each internal queue on each port:

```
/interface ethernet switch port
set ether1,ether2,ether3 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0"
```

Map each PCP value to an internal priority value, for convenience reasons simply map PCP to an internal priority 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 pcp-based-qos-priority-mapping=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Since the switch will empty the largest queue first and you need the highest priority to be served first, then you can assign this internal priority to a queue 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 priority-to-queue=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Finally, set each switch port to schedule packets based on the PCP value:

```
/interface ethernet switch port
set ether1,ether2,ether3 qos-scheme-precedence=pcp-based
```

Bandwidth Limiting

Both Ingress Port policer and Shaper provide bandwidth limiting features for CRS switches.

- Ingress Port Policer sets RX limit on port:

```
/interface ethernet switch ingress-port-policer
add port=ether5 meter-unit=bit rate=10M
```

- Shaper sets TX limit on port:

```
/interface ethernet switch shaper
add port=ether5 meter-unit=bit rate=10M
```

Traffic Storm Control

The same Ingress Port policer also can be used for the traffic storm control to prevent disruptions on Layer 2 ports caused by broadcast, multicast, or unicast traffic storms.

- Broadcast storm control example on ether5 port with 500 packet limit per second:

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=500 meter-unit=packet packet-types=broadcast
```

- Example with multiple packet types which includes ARP and ND protocols and unregistered multicast traffic. Unregistered multicast is traffic that is not defined in the Multicast Forwarding Database.

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=5k meter-unit=packet packet-types=broadcast,arp-or-nd,unregistered-multicast
```

See also

- CRS1xx/2xx series switches examples
- CRS Router
- CRS1xx/2xx VLANs with Trunks
- Basic VLAN switching
- Bridge Hardware Offloading
- Spanning Tree Protocol
- IGMP Snooping
- DHCP Snooping and Option 82
- MTU on RouterBOARD
- Layer2 misconfiguration

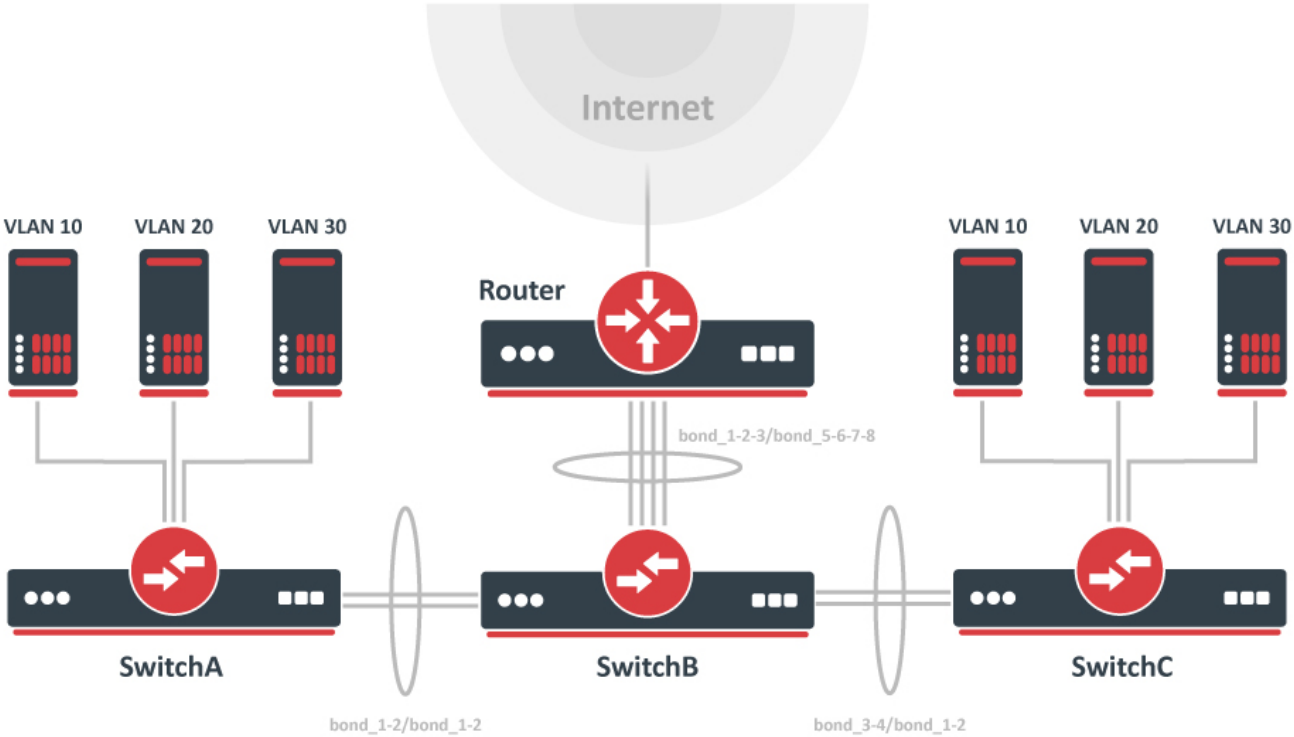
CRS3xx, CRS5xx, CCR2116, CCR2216 VLANs with Bonds

Summary

This page will show how to configure multiple switches to use bonding interfaces and port-based VLANs, it will also show a working example with a DHCP-Server, interVLAN routing, management IP, and invalid VLAN filtering configuration.

> ⚠ This article applies to CRS3xx, CRS5xx, CCR2116, and CCR2216 devices and not CRS1xx/CRS2xx.

For this network topology, we will be using two CRS326-24G-2S+, one CRS317-1G-16S+, and one CCR1072-1G-8S+, but the same principles can be applied to any CRS3xx, CRS5xx series devices, and a router.



In this setup, SwitchA and SwitchC will tag all traffic from ports ether1-ether8 to VLAN ID 10, ether9-ether16 to VLAN ID 20, ether17-ether24 to VLAN ID 30. Management will only be possible if a user is connecting with tagged traffic with VLAN ID 99 from ether1 on SwitchA or SwitchB, connecting to all devices will also be possible from the router using tagged traffic with VLAN ID 99. The SFP+ ports in this setup are going to be used as VLAN trunk ports while being in a bond to create a LAG interface.

Bonding

Bonding interfaces are used when a larger amount of bandwidth is required, this is done by creating a link aggregation group, which also provides hardware automatic failover and load balancing for switches. By adding two 10Gbps interfaces to bonding, you can increase the theoretical bandwidth limit to 20Gbps. Make sure that all bonded interfaces are linked to the same speed rates.

> When using the hardware-offloaded bridge, the CRS3xx, CRS5xx, CCR2116, and CCR2216 devices aggregate traffic using the built-in switch chip without using CPU resources. To route the traffic a router with a powerful CPU is required to handle the aggregated traffic.

To create a 20Gbps bonding interface from sfp-sfpplus1 and sfp-sfpplus2 between SwitchA to SwitchB and between SwitchC to SwitchB, use these commands on **SwitchA** and **SwitchC**:

```
/interface bonding
add mode=802.3ad name=bond_1-2 slaves=sfp-sfpplus1,sfp-sfpplus2
```

To create a 40Gbps bonding interface between SwitchB and the Router and 20Gbps bonding interfaces between SwitchA and SwitchC, use these commands on **SwitchB**:

```
/interface bonding
add mode=802.3ad name=bond_1-2 slaves=sfp-sfpplus1,sfp-sfpplus2
add mode=802.3ad name=bond_3-4 slaves=sfp-sfpplus3,sfp-sfpplus4
add mode=802.3ad name=bond_5-6-7-8 slaves=sfp-sfpplus5,sfp-sfpplus6,sfp-sfpplus7,sfp-sfpplus8
```

In our case the Router needs a software-based bonding interface, use these commands on the **Router**:

```
/interface bonding
add mode=802.3ad name=bond_1-2-3-4 slaves=sfp-sfpplus1,sfp-sfpplus2,sfp-sfpplus3,sfp-sfpplus4
```

> Interface bonding does not create an interface with a larger link speed. Interface bonding creates a virtual interface that can load balance traffic over multiple interfaces. More details can be found in the LAG interfaces and load balancing page.

Port switching

All switches in this setup require that all used ports are switched together. For bonding, you should add the bonding interface as a bridge port, instead of individual bonding ports. Use these commands on **SwitchA** and **SwitchC**:

```
/interface bridge
add name=bridge vlan-filtering=no
/interface bridge port
add bridge=bridge interface=ether1 pvid=10
add bridge=bridge interface=ether2 pvid=10
add bridge=bridge interface=ether3 pvid=10
add bridge=bridge interface=ether4 pvid=10
add bridge=bridge interface=ether5 pvid=10
add bridge=bridge interface=ether6 pvid=10
add bridge=bridge interface=ether7 pvid=10
add bridge=bridge interface=ether8 pvid=10
add bridge=bridge interface=ether9 pvid=20
add bridge=bridge interface=ether10 pvid=20
add bridge=bridge interface=ether11 pvid=20
add bridge=bridge interface=ether12 pvid=20
add bridge=bridge interface=ether13 pvid=20
add bridge=bridge interface=ether14 pvid=20
add bridge=bridge interface=ether15 pvid=20
add bridge=bridge interface=ether16 pvid=20
add bridge=bridge interface=ether17 pvid=30
add bridge=bridge interface=ether18 pvid=30
add bridge=bridge interface=ether19 pvid=30
add bridge=bridge interface=ether20 pvid=30
add bridge=bridge interface=ether21 pvid=30
add bridge=bridge interface=ether22 pvid=30
add bridge=bridge interface=ether23 pvid=30
add bridge=bridge interface=ether24 pvid=30
add bridge=bridge interface=bond_1-2
```

Add all bonding interfaces to a single bridge on SwitchB by using these commands on **SwitchB**:

```
/interface bridge
add name=bridge vlan-filtering=no
/interface bridge port
add bridge=bridge interface=bond_1-2
add bridge=bridge interface=bond_3-4
add bridge=bridge interface=bond_5-6-7-8
```

Management IP

It is very useful to create a management interface and assign an IP address to it in order to preserve access to the switch. This is also very useful when updating your switches since such traffic to the switch will be blocked when enabling invalid VLAN filtering.

Create a routable VLAN interface on **SwitchA**, **SwitchB,** and **SwitchC**:

```
/interface vlan
add interface=bridge name=MGMT vlan-id=99
```

The Router needs a routable VLAN interface to be created on the bonding interface, use these commands to create a VLAN interface on the **Router**:

```
/interface vlan
add interface=bond_1-2-3-4 name=MGMT vlan-id=99
```

For this guide we are going to use these addresses for each device:

Device	Address
Router	192.168.99.1
SwitchA	192.168.99.2

SwitchB	192.168.99.3
SwitchC	192.168.99.4

Add an IP address for each switch device on the VLAN interface (change X to the appropriate number):

```
/ip address
add address=192.168.99.X/24 interface=MGMT
```

Do not forget to add the default gateway and specify a DNS server on the switch devices:

```
/ip route
add gateway=192.168.99.1
/ip dns
set servers=192.168.99.1
```

Add the IP address on the **Router**:

```
/ip address
add address=192.168.99.1/24 interface=MGMT
```

Invalid VLAN filtering

Since most ports on SwitchA and SwitchC are going to be access ports, you can set all ports to accept only certain types of packets, in this case, we will want SwitchA and SwitchC to only accept untagged packets, use these commands on **SwitchA** and **SwitchC**:

```
/interface bridge port
set [ find ] frame-types=admit-only-untagged-and-priority-tagged
```

There is an exception for frame types on SwitchA and SwitchC, in this setup access to the management is required from ether1 and bonding interfaces, they require that tagged traffic can be forwarded. Use these commands on **SwitchA** and **SwitchC**:

```
/interface bridge port
set [find where interface=ether1] frame-types=admit-all
set [find where interface=bond_1-2] frame-types=admit-only-vlan-tagged
```

On SwitchB only tagged packets should be forwarded, use these commands on **SwitchB**:

```
/interface bridge port
set [ find ] frame-types=admit-only-vlan-tagged
```

An optional step is to set `frame-types=admit-only-vlan-tagged` on the bridge interface in order to disable the default untagged VLAN 1 (`pvid=1`). We are using the tagged VLAN on the bridge for management access, so there is no need to accept untagged traffic on the bridge. Use these commands on the **SwitchA**, **SwitchB** and **SwitchC**:

```
/interface bridge set [find name=bridge] frame-types=admit-only-vlan-tagged
```

It is required to set up a bridge VLAN table. In this network setup, we need to allow VLAN 10 on ether1-ether8, VLAN 20 on ether9-ether16, VLAN 30 on ether17-ether24, VLAN 10,20,30,99 on bond_1-2, and a special case for ether1 to allow to forward VLAN 99 on SwitchA and SwitchC. Use these commands on **SwitchA** and **SwitchC**:

```
/interface bridge vlan
add bridge=bridge tagged=bond_1-2 vlan-ids=10
add bridge=bridge tagged=bond_1-2 vlan-ids=20
add bridge=bridge tagged=bond_1-2 vlan-ids=30
add bridge=bridge tagged=bridge,bond_1-2,ether1 vlan-ids=99
```

⚠ Bridge ports with `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged` will be automatically added as untagged ports for the `pvid` VLAN.

Similarly, it is required to set up a bridge VLAN table for SwitchB. Use these commands on **SwitchB**:

```
/interface bridge vlan
add bridge=bridge tagged=bond_1-2,bond_3-4,bond_5-6-7-8 vlan-ids=10,20,30
add bridge=bridge tagged=bond_1-2,bond_3-4,bond_5-6-7-8,bridge vlan-ids=99
```

When everything is configured, VLAN filtering can be enabled. Use these commands on **SwitchA**, **SwitchB,** and **SwitchC**:

```
/interface bridge
set bridge vlan-filtering=yes
```

Double-check if port-based VLANs are set up properly. If a mistake was made, you might lose access to the switch and it can only be regained by resetting the configuration or by using the serial console.

VLAN filtering is described more in the Bridge VLAN Filtering section.

InterVLAN routing

To create InterVLAN routing, the VLAN interface for each customer VLAN ID must be created on the router and must have an IP address assigned to it. The VLAN interface must be created on the bonding interface created previously.

Use these commands on the **Router**:

```
/interface vlan
add interface=bond_1-2-3-4 name=VLAN10 vlan-id=10
add interface=bond_1-2-3-4 name=VLAN20 vlan-id=20
add interface=bond_1-2-3-4 name=VLAN30 vlan-id=30
/ip address
add address=192.168.10.1/24 interface=VLAN10
add address=192.168.20.1/24 interface=VLAN20
add address=192.168.30.1/24 interface=VLAN30
```

These commands are required for DHCP-Server. In case interVLAN routing is not desired but a DHCP-Server on a single router is required, then use Firewall Filter to block access between different subnets.

Since RouterOS v7, it is possible to route traffic using the L3 HW offloading on certain devices. See more details on L3 Hardware Offloading.

DHCP-Server

To get the DHCP-Server working for each VLAN ID, the server must be set up on the previously created VLAN interfaces (one server for each VLAN ID). Preferably each VLAN ID should have its own subnet and its own IP pool. DNS Server could be specified as the router's IP address for a particular VLAN ID or a global DNS Server could be used, but this address must be reachable.

To set up the DHCP-Server, use these commands on the **Router**:

```
/ip pool
add name=VLAN10_POOL ranges=192.168.10.100-192.168.10.200
add name=VLAN20_POOL ranges=192.168.20.100-192.168.20.200
add name=VLAN30_POOL ranges=192.168.30.100-192.168.30.200
/ip dhcp-server
add address-pool=VLAN10_POOL disabled=no interface=VLAN10 name=VLAN10_DHCP
add address-pool=VLAN20_POOL disabled=no interface=VLAN20 name=VLAN20_DHCP
add address-pool=VLAN30_POOL disabled=no interface=VLAN30 name=VLAN30_DHCP
/ip dhcp-server network
add address=192.168.10.0/24 dns-server=192.168.10.1 gateway=192.168.10.1
add address=192.168.20.0/24 dns-server=192.168.20.1 gateway=192.168.20.1
add address=192.168.30.0/24 dns-server=192.168.30.1 gateway=192.168.30.1
```

In case the router's DNS Server is being used, don't forget to allow remote requests and make sure DNS Servers are configured on the router. Use these commands on the **Router**:

```
/ip dns
set allow-remote-requests=yes servers=8.8.8.8
```

Make sure to secure your local DNS Server with Firewall from the outside when using `allow-remote-requests` set to `yes` since your DNS Server can be used for DDoS attacks if it is accessible from the Internet by anyone.

Don't forget to create NAT, assuming that sfp-sfpplus8 is used as WAN port, use these commands on the **Router**:

```
/ip firewall nat
add action=masquerade chain=srcnat out-interface=sfp-sfpplus8
```

Jumbo frames

One can increase the total throughput in such a setup by enabling jumbo frames. This reduces the packet overhead by increasing the Maximum Transmission Unit (MTU). If a device in your network does not support jumbo frames, then it will not benefit from a larger MTU. Usually, the whole network does not support jumbo frames, but you can still benefit when sending data between devices that support jumbo frames, including all switches in the path.

In this case, if clients behind SwitchA and client behind SwitchC supports jumbo frames, then enabling jumbo frames will be beneficial. Before enabling jumbo frames, determine the MAX-L2MTU by using this command:

```
[admin@SwitchA] > interface print
Flags: R - RUNNING
Columns: NAME, TYPE, ACTUAL-MTU, L2MTU, MAX-L2MTU, MAC-ADDRESS
 #   NAME          TYPE   ACTUAL-MTU  L2MTU  MAX-L2MTU  MAC-ADDRESS
 1 R sfp-sfpplus1  ether        1500   1584      10218  64:D1:54:FF:E3:7F
```

More information can be found in MTU manual page.

When MAX-L2MTU is determined, choose the MTU size depending on the traffic on your network, use this command on **SwitchA**, **SwitchB** and **SwitchC**:

```
/interface ethernet
set [ find ] l2mtu=10218 mtu=10218
```

Don't forget to change the MTU on your client devices too, otherwise, the above-mentioned settings will not have any effect.

Layer2 misconfiguration

Introduction

There are certain configurations that are known to have major flaws by design and should be avoided by all means possible. Misconfigured Layer2 can sometimes cause hard to detect network errors, random performance drops, certain segments of a network to be unreachable, certain networking services to be malfunctioning, or a complete network failure. This page will contain some common and not so very common configurations that will cause issues in your network.

Bridges on a single switch chip

Consider the following scenario, you have a device with a built-in switch chip and you need to isolate certain ports from each other, for this reason, you have created multiple bridges and enabled hardware offloading on them. Since each bridge is located on a different Layer2 domain, then Layer2 frames will not be forwarded between these bridges, as a result, ports in each bridge are isolated from other ports on a different bridge.

Configuration

```
/interface bridge
add name=bridge1
add name=bridge2
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge2 interface=ether3
add bridge=bridge2 interface=ether4
```

Problem

After a simple performance test, you might notice that one bridge is capable of forwarding traffic at wire speed while the second, third, etc. bridge is not able to forward as much data as the first bridge. Another symptom might be that there exists a huge latency for packets that need to be routed. After a quick inspection, you might notice that the CPU is always at full load, this is because hardware offloading is not available on all bridges, but is available only on one bridge. By checking the hardware offloading status you will notice that only one bridge has it active:

```
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                                 BRIDGE                                    HW
 0   H ether1                                    bridge1                                   yes
 1   H ether2                                    bridge1                                   yes
 2     ether3                                    bridge2                                   yes
 3     ether4                                    bridge2                                   yes
```

The reason why only one bridge has the hardware offloading flag available is that the device does not support port isolation. If port isolation is not supported, then only one bridge will be able to offload the traffic to the switch chip.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Missing "H" flag to bridge ports
- Low throughput
- High CPU usage

Solution

Not all device devices support port isolation, currently only CRS1xx/CRS2xx series devices support it and only 7 isolated and hardware offloaded bridges are supported at the same time, other devices will have to use the CPU to forward the packets on other bridges. This is usually a hardware limitation and a different device might be required. Bridge split-horizon parameter is a software feature that disables hardware offloading and when using bridge filter rules you need to enable forward all packets to the CPU, which requires the hardware offloading to be disabled. You can control which bridge will be hardware offloaded with the `hw=yes` flag and by setting `hw=no` to other bridges, for example:

```
/interface bridge port set [find where bridge=bridge1] hw=no
/interface bridge port set [find where bridge=bridge2] hw=yes
```

Sometimes it is possible to restructure a network topology to use VLANs, which is the proper way to isolate Layer2 networks.

Packet flow with hardware offloading and MAC learning

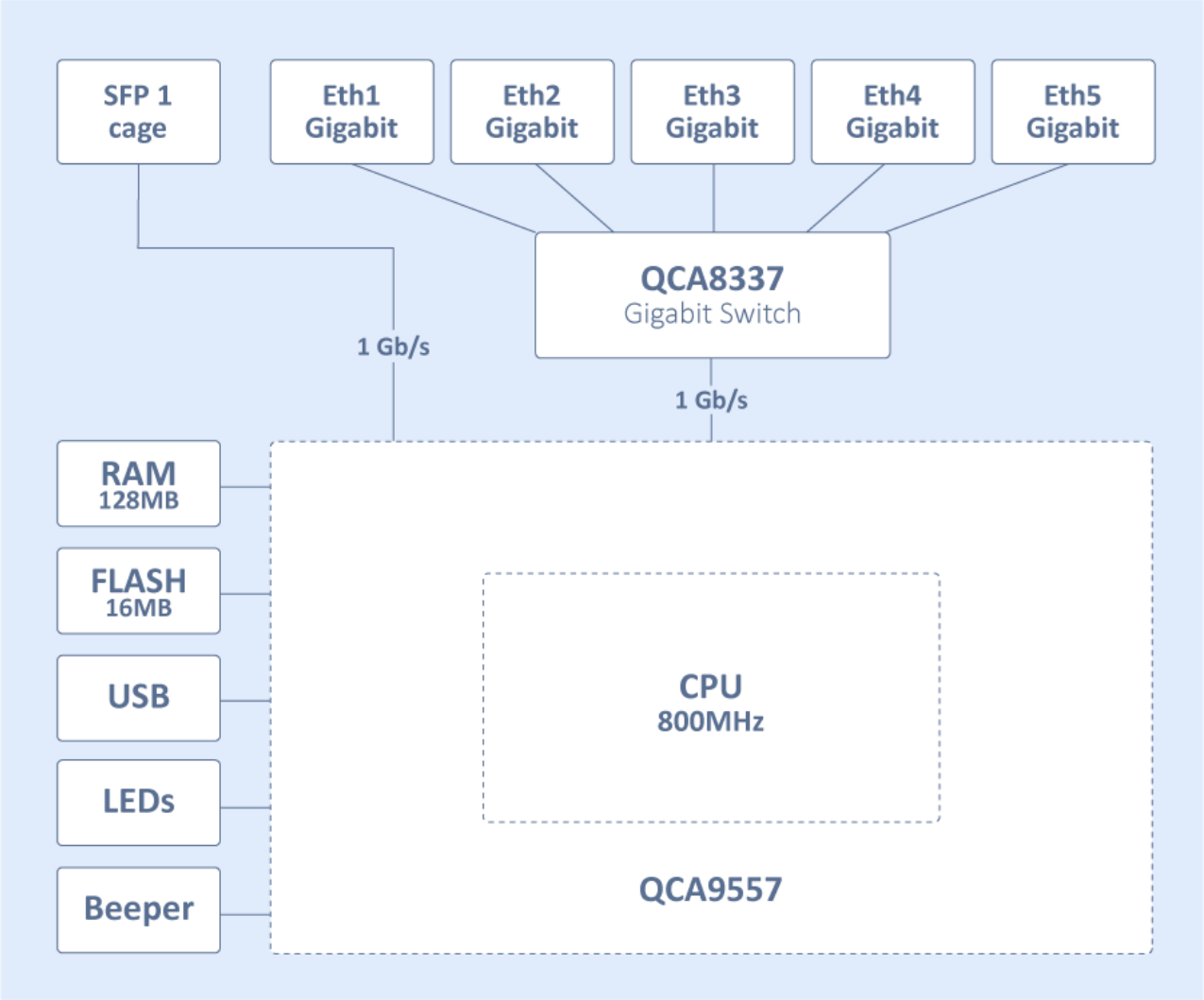
Consider the following scenario, you set up a bridge and have enabled hardware offloading in order to maximize the throughput for your device, as a result, your device is working as a switch, but you want to use Sniffer or Torch tools for debugging purposes, or maybe you want to implement packet logging.

Configuration

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 hw=yes interface=ether1 learn=yes
add bridge=bridge1 hw=yes interface=ether2 learn=yes
```

Problem

When running Sniffer or Torch tools to capture packets you might notice that barely any packets are visible, only some unicast packets, but mostly broadcast/multicast packets are captured, while the interfaces report that much larger traffic is flowing through certain interfaces than the traffic that was captured. Since RouterOS v6.41 if you add two or more Ethernet interfaces to a bridge and enable Hardware Offloading, then the switch chip will be used to forward packets between ports. To understand why only some packets are captured, we must first examine how the switch chip is interconnected with the CPU, in this example, we can use a block diagram from a generic 5-Port Ethernet router:



For this device, each Ethernet port is connected to the switch chip and the switch chip is connected to the CPU using the CPU port (sometimes called the **switch-cpu** port). For packets to be visible in Sniffer or Torch tools, the packet must be sent from an Ethernet port to the CPU port, this means that the packet must be destined to the CPU port (destination MAC address of the packet matches the bridge's MAC address) or the packet's MAC address has not be learnt (packet is flooded to all ports), this behavior is because of **MAC learning**.

The switch chip keeps a list of MAC addresses and ports called the **Host table**. Whenever a packet needs to be forwarded, the switch chip checks the packet's destination MAC address against the hosts table to find which port should it use to forward the packet. If the switch chip cannot find the destination MAC address, then the packet is flooded to all ports (including the CPU port). In situations where a packet is supposed to be forwarded from, for example, ether1 to ether2 and the MAC address for the device behind ether2 is in the host table, then the packet is never sent to the CPU and therefore will not be visible to Sniffer or Torch tool.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Packets not visible by Sniffer or Torch tool
- Filter rules not working

Solution

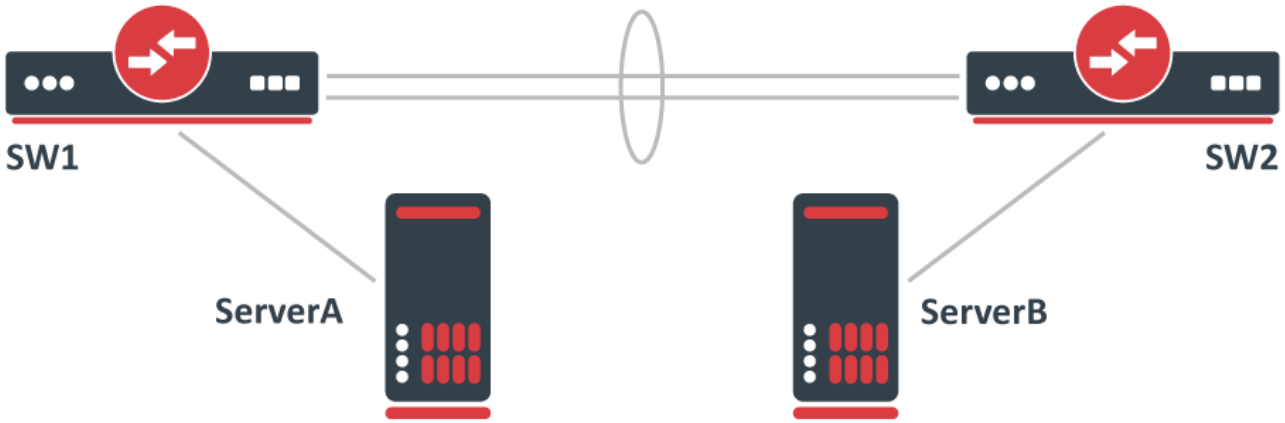
Packets with a destination MAC address that has been learned will not be sent to the CPU since the packets are not being flooded to all ports. If you do need to send certain packets to the CPU for a packet analyzer or a firewall, then it is possible to copy or redirect the packet to the CPU by using ACL rules. Below is an example of how to send a copy of packets that are meant for **4C:5E:0C:4D:12:4B**:

```
/interface ethernet switch rule
add copy-to-cpu=yes dst-mac-address=4C:5E:0C:4D:12:4B/FF:FF:FF:FF:FF:FF ports=ether1 switch=switch1
```

⚠ If the packet is sent to the CPU, then the packet must be processed by the CPU, this increases the CPU load.

LAG interfaces and load balancing

Consider the following scenario, you have created a LAG interface to increase total bandwidth between 2 network nodes, usually, these are switches. For testing purposes to make sure that the LAG interface is working properly you have attached two servers that transfer data, most commonly the well-known network performance measurement tool https://en.wikipedia.org/wiki/Iperf is used to test such setups. For example, you might have made a LAG interface out of two Gigabit Ethernet ports, which gives you a virtual interface that can load balance traffic over both interfaces and theoretically reach 2Gbps throughput, while the servers are connected using a 10Gbps interface, for example, SFP+.



Configuration

The following configuration is relevant to **SW1** and **SW2**:

```
/interface bonding
add mode=802.3ad name=bond1 slaves=ether1,ether2
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=bond1
add bridge=bridge1 interface=sfp-sfpplus1
```

Problem

After initial tests, you immediately notice that your network throughput never exceeds the 1Gbps limit even though the CPU load on the servers is low as well as on the network nodes (switches in this case), but the throughput is still limited to only 1Gbps. The reason behind this is because LACP (802.ad) uses transmit hash policy in order to determine if traffic can be balanced over multiple LAG members, in this case, a LAG interface does not create a 2Gbps interface, but rather an interface that can balance traffic over multiple slave interface whenever it is possible. For each packet a transmit hash is generated, this determines through which LAG member will the packet be sent, this is needed in order to avoid packets being out of order, there is an option to select the transmit hash policy, usually, there is an option to choose between Layer2 (MAC), Layer3 (IP) and Layer4 (Port), in RouterOS, this can be selected by using the `transmit-hash-policy` parameter. In this case, the transmit hash is the same since you are sending packets to the same destination MAC address, as well as the same IP address and Iperf uses the same port as well, this generates the same transmit hash for all packets and load balancing between LAG members is not possible. Note that not always packets will get balanced over LAG members even though the destination is different, this is because the standardized transmit hash policy can generate the same transmit hash for different destinations, for example, 192.168.0.1 /192.168.0.2 will get balanced, but 192.168.0.2/192.168.0.4 will **NOT** get balanced in case `layer2-and-3` transmit hash policy is used and the destination MAC address is the same.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Traffic going through only one LAG member

Solution

Choose the proper transmit hash policy and test your network's throughput properly. The simplest way to test such setups is to use multiple destinations, for example, instead of sending data to just one server, rather send data to multiple servers, this will generate a different transmit hash for each packet and will make load balancing across LAG members possible. For some setups, you might want to change the bonding interface mode to increase the total throughput, for UDP traffic `balance-rr` mode might be sufficient, but can cause issues for TCP traffic, you can read more about selecting the right mode for your setup here.

VLAN interface on a slave interface

Consider the following scenario, you have created a bridge and you want a DHCP Server to give out IP addresses only to a certain tagged VLAN traffic, for this reason, you have created a VLAN interface, specified a VLAN ID and created a DHCP Server on it, but for some reason, it is not working properly.

Configuration

```
/interface bridge
add name=bridge1
/interface bridge port
add interface=ether1 bridge=bridge1
add interface=ether2 bridge=bridge1
/interface vlan
add name=VLAN99 interface=ether1 vlan-id=99
/ip pool
add name=VLAN99_POOL range=192.168.99.100-192.168.99.200
/ip address add address=192.168.99.1/24 interface=VLAN99
/ip dhcp-server
add interface=VLAN99 address-pool=VLAN99_POOL disabled=no
/ip dhcp-server network
add address=192.168.99.0/24 gateway=192.168.99.1 dns-server=192.168.99.1
```

Problem

When you add an interface to a bridge, the bridge becomes the master interface and all bridge ports become slave ports, this means that all traffic that is received on a bridge port is captured by the bridge interface and all traffic is forwarded to the CPU using the bridge interface instead of the physical interface. As a result VLAN interface that is created on a slave interface will never capture any traffic at all since it is immediately forwarded to the master interface before any packet processing is being done. The usual side effect is that some DHCP clients receive IP addresses and some don't.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- DHCP Client/Server not working properly;
- Device is unreachable;
- The device behind a bridge is unreachable with tagged traffic;

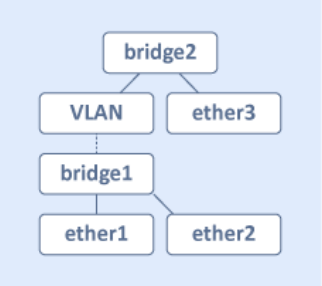
Solution

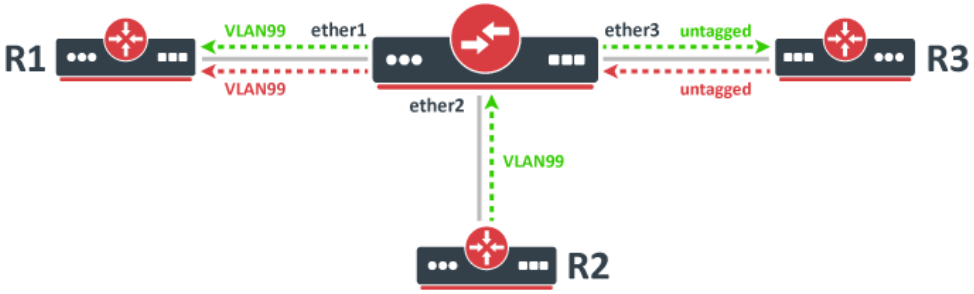
Change the interface on which the VLAN interface will be listening for traffic, change it to the master interface:

```
/interface vlan set VLAN99 interface=bridge1
```

VLAN on a bridge in a bridge

Consider the following scenario, you have a set of interfaces (don't have to be physical interfaces) and you want all of them to be in the same Layer2 segment, the solution is to add them to a single bridge, but you require that traffic from one port tags all traffic into a certain VLAN. This can be done by creating a VLAN interface on top of the bridge interface and by creating a separate bridge that contains this newly created VLAN interface and an interface, which is supposed to add a VLAN tag to all received traffic. A network diagram can be found below:





Configuration

```
/interface bridge
add name=bridge1
add name=bridge2
/interface vlan
add interface=bridge1 name=VLAN vlan-id=99
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge2 interface=VLAN
add bridge=bridge2 interface=ether3
```

Problem

To better understand the underlying problems, let's first look at the bridge host table.

```
[admin@switch] /interface bridge host print where !local
Flags: X - disabled, I - invalid, D - dynamic, L - local, E - external
 #       MAC-ADDRESS        VID ON-INTERFACE    BRIDGE
 0   D   CC:2D:E0:E4:B3:A1      ether1          bridge1
 1   D   CC:2D:E0:E4:B3:A2      ether2          bridge1
 2   D   CC:2D:E0:E4:B3:A1      VLAN            bridge2
 3   D   CC:2D:E0:E4:B3:A2      VLAN            bridge2
 4   D   CC:2D:E0:E4:B3:A3      ether3          bridge2
```

Devices on **ether1** and **ether2** need to send tagged packets with VLAN-ID 99 in order to reach the host on **ether3** (other packets do not get passed towards VLAN interface and further bridged with ether3). We can see in the host table that **bridge2** has learned these hosts. Packets coming from **ether3** to **ether1** w ill be correctly sent out tagged and traffic will not be flooded in **bridge1**. But since MAC learning is only possible between bridge ports and not on interfaces that are created on top of the bridge interface, packets sent from **ether2** to **ether3** will be flooded in **bridge1**.

Also if a device behind **ether3** is using (R)STP, then **ether1** and **ether2** will send out tagged BPDUs which violates the IEEE 802.1W standard. Because of the broken MAC learning functionality and broken (R)STP this setup and configuration must be avoided. It is also known that in some setups this kind of configuration can prevent you from connecting to the device by using MAC telnet.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Port blocked by RSTP
- Loops in network

- Port flapping
- Traffic is flooded to all ports
- MAC telnet is unable to connect
- Device inaccessible

Solution

Use bridge VLAN filtering. The proper way to tag traffic is to assign a VLAN ID whenever traffic enters a bridge, this behavior can easily be achieved by specifying **PVID** value for a bridge port and specifying which ports are **tagged** (trunk) ports and which are **untagged** (access) ports. Below is an example of how such a setup should have been configured:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3 pvid=99
/interface bridge vlan
add bridge=bridge1 tagged=ether1,ether2 untagged=ether3 vlan-ids=99
```

By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

VLAN in a bridge with a physical interface

Very similar case to VLAN on a bridge in a bridge. The most popular use case is when you want to bridge a physical interface with a VLAN (simplified trunk /access port setup). In such setup you might want to send out tagged traffic on one side and untagged on the other side. To acomplish this, you create a VLAN interface on the trunk port (the tagged side), then create a bridge and add both the VLAN interface and the physical interface (the untagged side) as bridge ports.

Configuration

```
/interface vlan
add interface=ether1 name=VLAN99 vlan-id=99
/interface bridge
add name=bridge1
/interface bridge port
add interface=ether2 bridge=bridge1
add interface=VLAN99 bridge=bridge1
```

Problem

This setup and configuration will work in most cases, but it violates the IEEE 802.1W standard when (R/M)STP is used. If this is the only device in your Layer2 domain, then this should not cause problems, but problems can arise when there are other vendor switches. The reason for this is that RSTP on a bridge interface is enabled by default, allowing Bridge Protocol Data Units (BPDUs) to be sent from each bridge port. While **ether2** transmits BPDUs correctly without tagging, **VLAN99** interface, being a bridge port, sends tagged BPDUs over ether1. Not all switches can understand tagged BPDUs. Precautions should be made with this configuration in a more complex network where there are multiple network topologies for certain (group of) VLANs, this is relevant to MSTP and PVSTP(+) with mixed vendor devices. In a ring-like topology with multiple network topologies for certain VLANs, one port from the switch will be blocked, but in MSTP and PVSTP(+) a path can be opened for a certain VLAN, in such a situation it is possible that devices that don't support PVSTP(+) will untag the BPDUs and forward the BPDU, as a result, the switch will receive its own packet, trigger a loop detection and block a port, this can happen to other protocols as well, but (R)STP is the most common case. If a switch is using a BPDU guard function, then this type of configuration can trigger it and cause a port to be blocked by STP. It has been reported that this type of configuration can prevent traffic from being forwarded over certain bridge ports over time when using 6.41 or later. This type of configuration does not only break (R/M)STP, but it can cause loop warnings, this can be caused by MNDP packets or any other packets that are directly sent out from an interface.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Port blocked by RSTP
- Loops in network
- Port flapping
- Traffic stops forwarding over time
- BPDUs ignored by other RSTP enabled devices

Solution

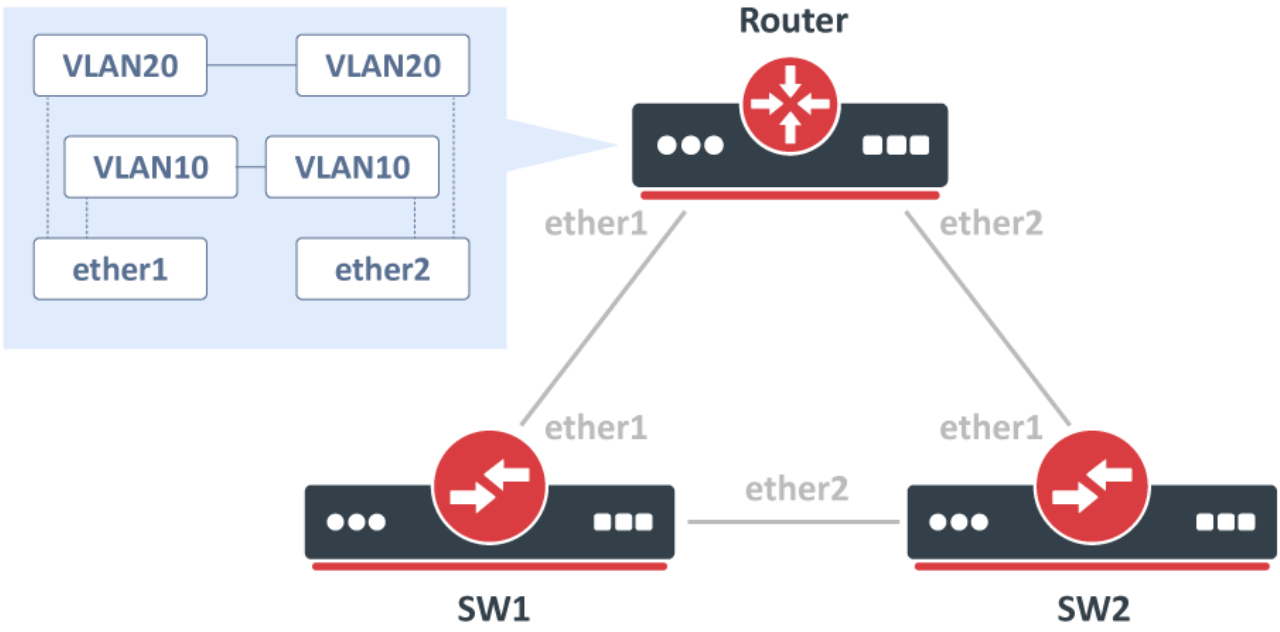
To avoid compatibility issues you should use bridge VLAN filtering. Below you can find an example of how the same traffic tagging effect can be achieved with a bridge VLAN filtering configuration:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2 pvid=99
/interface bridge vlan
add bridge=bridge1 tagged=ether1 untagged=ether2 vlan-ids=99
```

> By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

Bridged VLAN on physical interfaces

A very similar case to VLAN on a bridge in a bridge, consider the following scenario, you have a couple of switches in your network and you are using VLANs to isolate certain Layer2 domains and connect these switches to a router that assigns addresses and routes the traffic to the world. For redundancy, you connect all switches directly to the router and have enabled RSTP, but to be able to setup DHCP Server you decide that you can create a VLAN interface for each VLAN on each physical interface that is connected to a switch and add these VLAN interfaces in a bridge. A network diagram can be found below:



Configuration

Only the router part is relevant to this case, switch configuration doesn't really matter as long as ports are switched. Router configuration can be found below:

```
/interface bridge
add name=bridge10
add name=bridge20
/interface vlan
add interface=ether1 name=ether1_v10 vlan-id=10
add interface=ether1 name=ether1_v20 vlan-id=20
add interface=ether2 name=ether2_v10 vlan-id=10
add interface=ether2 name=ether2_v20 vlan-id=20
/interface bridge port
add bridge=bridge10 interface=ether1_v10
add bridge=bridge10 interface=ether2_v10
add bridge=bridge20 interface=ether1_v20
add bridge=bridge20 interface=ether2_v20
```

Problem

You might notice that the network is having some weird delays or even the network is unresponsive, you might notice that there is a loop detected (packet received with own MAC address) and some traffic is being generated out of nowhere. The problem occurs because a broadcast packet that is coming from either one of the VLAN interface created on the **Router** will be sent out the physical interface, packet will be forwarded through the physical interface, through a switch and will be received back on a different physical interface, in this case, broadcast packets sent out **ether1_v10** will be received on **ether2**, packet will be captured by **ether2_v10**, which is bridged with **ether1_v10** and will get forwarded again the same path (loop). (R)STP might not always detect this loop since (R)STP is not aware of any VLANs, a loop does not exist with untagged traffic, but exists with tagged traffic. In this scenario, it is quite obvious to spot the loop, but in more complex setups it is not always easy to detect the network design flaw. Sometimes this network design flaw might get unnoticed for a very long time if your network does not use broadcast traffic, usually, Neighbor Discovery Protocol is broadcasting packets from the VLAN interface and will usually trigger a loop detection in such a setup. Sometimes it is useful to capture the packet that triggered a loop detection, this can by using sniffer and analyzing the packet capture file:

```
/tool sniffer
set filter-mac-address=4C:5E:0C:4D:12:44/FF:FF:FF:FF:FF:FF \
filter-interface=ether1 filter-direction=rx file-name=loop_packet.pcap
```

Or a more convenient way using logging:

```
/interface bridge filter
add action=log chain=forward src-mac-address=4C:5E:0C:4D:12:44/FF:FF:FF:FF:FF:FF
add action=log chain=input src-mac-address=4C:5E:0C:4D:12:44/FF:FF:FF:FF:FF:FF
```

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Port blocked by (R)STP;
- Loops in network;
- Low throughput;
- Port flapping;
- Network inaccessible;

Solution

A solution is to use bridge VLAN filtering in order to make all bridges compatible with IEEE 802.1W and IEEE 802.1Q.

```
/interface bridge
add name=bridge vlan-filtering=yes
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2
/interface bridge vlan
add bridge=bridge tagged=ether1,ether2,bridge vlan-ids=10
add bridge=bridge tagged=ether1,ether2,bridge vlan-ids=20
/interface vlan
add name=vlan10 interface=bridge vlan-id=10
add name=vlan20 interface=bridge vlan-id=20
```

By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

Bridged VLAN

A more simplified scenario of Bridged VLAN on physical interfaces, but in this case, you simply want to bridge two or more VLANs together that are created on different physical interfaces. This is a very common type of setup that deserves a separate article since misconfiguring this type of setup has caused multiple network failures. This type of setup is also used for VLAN translation.

Configuration

```
/interface vlan
add interface=ether1 name=ether1_v10 vlan-id=10
add interface=ether2 name=ether2_v10 vlan-id=10
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1_v10
add bridge=bridge1 interface=ether2_v10
```

Problem

You may notice that certain parts of the network are not accessible and/or certain links keep flapping. This is due to (R)STP, this type of configuration forces the device to send out tagged BPDUs, that might not be supported by other devices, including RouterOS. Since a device receives a malformed packet (tagged BPDUs should not exist in your network when running (R)STP, this violates IEEE 802.1W and IEEE 802.1Q), the device will not interpret the packet correctly and can have unexpected behavior.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Port blocked by (R)STP;
- Port flapping;
- Network inaccessible;

Solution

The easiest solution is to simply disable (R)STP on the bridge:

```
/interface bridge
set bridge1 protocol-mode=none
```

though it is still recommended to rewrite your configuration to use bridge VLAN filtering:

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
/interface bridge vlan
add bridge=bridge1 tagged=ether1,ether2 vlan-ids=10
```

By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port.

Bridge VLAN filtering on non-CRS3xx

Consider the following scenario, you found out the new bridge VLAN filtering feature and you decided to change the configuration on your device, you have a very simple trunk/access port setup and you like the concept of bridge VLAN filtering.

Configuration

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2 pvid=20
add bridge=bridge1 interface=ether3 pvid=30
add bridge=bridge1 interface=ether4 pvid=40
/interface bridge vlan
add bridge=bridge1 tagged=ether1 untagged=ether2 vlan-ids=20
add bridge=bridge1 tagged=ether1 untagged=ether3 vlan-ids=30
add bridge=bridge1 tagged=ether1 untagged=ether4 vlan-ids=40
```

Problem

For example, you use this configuration on a CRS1xx/CRS2xx series device and you started to notice that the CPU usage is very high and when running a performance test to check the network's throughput you notice that the total throughput is only a fraction of the wire-speed performance that it should easily reach. The cause of the problem is that not all devices support bridge VLAN filtering on a hardware level. All devices are able to be configured with bridge VLAN filtering, but only a few of them will be able to offload the traffic to the switch chip. If an improper configuration method is used on a device with a built-in switch chip, then the CPU will be used to forward the traffic.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Missing "H" flag on bridge ports
- Low throughput
- High CPU usage

Solution

Before using bridge VLAN filtering check if your device supports it at the hardware level, a table with compatibility can be found at the Bridge Hardware Offloading section. Each type of device currently requires a different configuration method, below is a list of which configuration should be used on a device in order to use the benefits of hardware offloading:

- CRS3xx series devices
- CRS1xx/CRS2xx series devices
- Other devices with a switch chip

VLAN filtering with multiple switch chips

Consider the following scenario, you have a device with two or more switch chips and you have decided to use a single bridge and set up VLAN filtering (by using the `/interface ethernet switch` menu) on a hardware level to be able to reach wire-speed performance on your network. This is very relevant for RB2011 and RB3011 series devices. In this example, let's assume that you want to have a single trunk port and all other ports are access ports, for example, **ether10** is our trunk port and **ether1-ether9** are our access ports.

Configuration

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
add bridge=bridge1 interface=ether4
add bridge=bridge1 interface=ether5
add bridge=bridge1 interface=ether6
add bridge=bridge1 interface=ether7
add bridge=bridge1 interface=ether8
add bridge=bridge1 interface=ether9
add bridge=bridge1 interface=ether10
/interface vlan
add interface=bridge1 name=VLAN10 vlan-id=10
/interface ethernet switch port
set ether1,ether2,ether3,ether4,ether5,ether6,ether7,ether8,ether9 default-vlan-id=10 vlan-header=always-strip
vlan-mode=secure
set ether10 vlan-header=add-if-missing vlan-mode=secure
set switch1-cpu,switch2-cpu vlan-mode=secure
/interface ethernet switch vlan
add ports=ether1,ether2,ether3,ether4,ether5,switch1-cpu switch=switch1 vlan-id=10
add ports=ether6,ether7,ether8,ether9,ether10,switch2-cpu switch=switch2 vlan-id=10
```

Problem

After running a few tests you might notice that packets from **ether6-ether10** are forwarded as expected, but packets from **ether1-ether5** are not always forwarded correctly (especially through the trunk port). The most noticeable issue would be that packets from **ether1-ether5** through **ether10** are simply dropped, this is because these ports are located on different switch chip, this means that VLAN filtering is not possible on a hardware level since the switch chip is not aware of the VLAN table's contents on a different switch chip. Packets that are being forwarded between ports that are located on different switch chips are also processed by the CPU, which means you won't be able to achieve wire-speed performance.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Packets being dropped;
- Low throughput;

Solution

The proper solution is to take into account this hardware design and plan your network topology accordingly. To solve this issue you must create two separate bridges and configure VLAN filtering on each switch chip, this limits the possibility to forward packets between switch chip, though it is possible to configure routing between both bridges (if devices that are connected on each switch chip are using different network subnets).

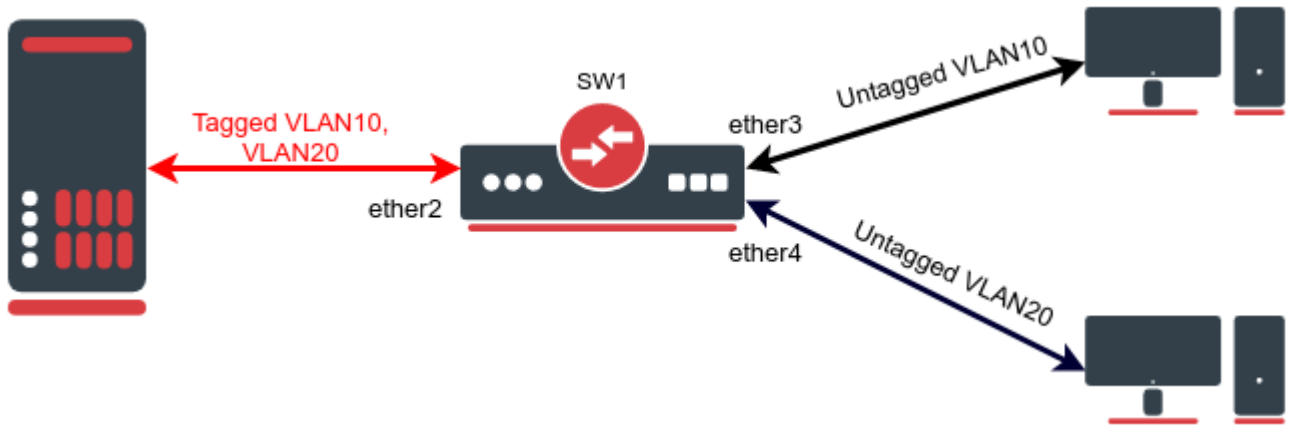
There is a way to configure the device to have all ports switch together and yet be able to use VLAN filtering on a hardware level, though this solution has some caveats. The idea is to sacrifice a single Ethernet port on each switch chip that will act as a trunk port to forward packets between switch chip, this can be done by plugging an Ethernet cable between both switch chip, for example, lets plug in an Ethernet cable between **ether5** and **ether6** then reconfigure your device assuming that these ports are trunk ports:

```
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
add bridge=bridge1 interface=ether4
add bridge=bridge1 interface=ether5
add bridge=bridge2 interface=ether6
add bridge=bridge2 interface=ether7
add bridge=bridge2 interface=ether8
add bridge=bridge2 interface=ether9
add bridge=bridge2 interface=ether10
/interface ethernet switch port
set ether1,ether2,ether3,ether4,ether7,ether8,ether9 default-vlan-id=10 vlan-header=always-strip vlan-
mode=secure
set ether5,ether6,ether10 vlan-header=add-if-missing vlan-mode=secure default-vlan-id=auto
set switch1-cpu,switch2-cpu vlan-mode=secure
/interface ethernet switch vlan
add ports=ether1,ether2,ether3,ether4,ether5,switch1-cpu switch=switch1 vlan-id=10
add ports=ether6,ether7,ether8,ether9,ether10,switch2-cpu switch=switch2 vlan-id=10
```

⚠ For 100Mbps switch chips, use `default-vlan-id=0` instead of `default-vlan-id=auto`

VLAN filtering with simplified bridge VLAN table

You need to create a network setup where multiple clients are connected to separate access ports and isolated by different VLANs, this traffic should be tagged and sent to the appropriate trunk port. Access ports are configured using a pvid property. As the trunk port is used on both VLANs, you decided to simplify configuration by adding a single bridge VLAN table entry and separate VLANs by a comma. This is especially useful when tagged trunk ports are used across large numbers of VLANs or even certain VLAN ranges (e.g. vlan-id=100-200). See a network diagram and configuration below.



Configuration

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3 pvid=10
add bridge=bridge1 interface=ether4 pvid=20
/interface bridge vlan
add bridge=bridge1 tagged=ether2 vlan-ids=10,20
```

Problem

Traffic is correctly forwarded and tagged from access ports to trunk port, but you might notice that some broadcast or multicast packets are actually flooded between both untagged access ports, although they should be on different VLANs. Furthermore, broadcast and multicast traffic from the tagged port is also flooded to both access ports. This might raise some security concerns as traffic from different networks can be sniffed. When you look at the bridge VLAN table, you notice that a single entry has been created for VLANs 10 and 20, and both untagged ports are part of the same VLAN group.

```
[admin@SW1] /interface bridge vlan print where tagged=ether2
Columns: BRIDGE, VLAN-IDS, CURRENT-TAGGED, CURRENT-UNTAGGED
# BRIDGE   VLAN-IDS  CURRENT-TAGGED  CURRENT-UNTAGGED
;;; port with pvid added to untagged group which might cause problems, consider adding a separate VLAN entry
0 bridge1        10  ether2          ether3
                 20                  ether4
```

Symptoms

- Traffic is flooded between different VLANs
- Red warning: `port with pvid added to untagged group which might cause problems, consider adding a separate VLAN entry`

Solution

When access ports have been configured using the pvid property, they get dynamically added to the appropriate VLAN entry. After creating a static VLAN entry with multiple VLANs or VLAN range, the untagged access port with a matching pvid also gets included in the same VLAN group or range. It might be useful to define a large number of VLANs using a single configuration line, but extra caution should be taken when access ports are configured. For this example, separate VLAN entries should be created:

```
/interface bridge vlan
add bridge=bridge1 tagged=ether2 untagged=ether3 vlan-ids=10
add bridge=bridge1 tagged=ether2 untagged=ether4 vlan-ids=20
```

MTU on the master interface

Consider the following scenario, you have created a bridge, added a few interfaces to it and have created a VLAN interface on top of the bridge interface, but you need to increase the MTU size on the VLAN interface in order to receive larger packets.

Configuration

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
/interface vlan
add interface=bridge1 name=VLAN99 vlan-id=99
```

Problem

As soon as you try to increase the MTU size on the VLAN interface, you receive an error that RouterOS **Could not set MTU**. This can happen when you are trying to set MTU larger than the L2MTU. In this case, you need to increase the L2MTU size on all slave interfaces, which will update the L2MTU size on the bridge interface. After this has been done, you will be able to set a larger MTU on the VLAN interface. The same principle applies to bond interfaces. You can increase the MTU on interfaces like VLAN, MPLS, VPLS, Bonding and other interfaces only when all physical slave interfaces have proper L2MTU set.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Cannot change MTU

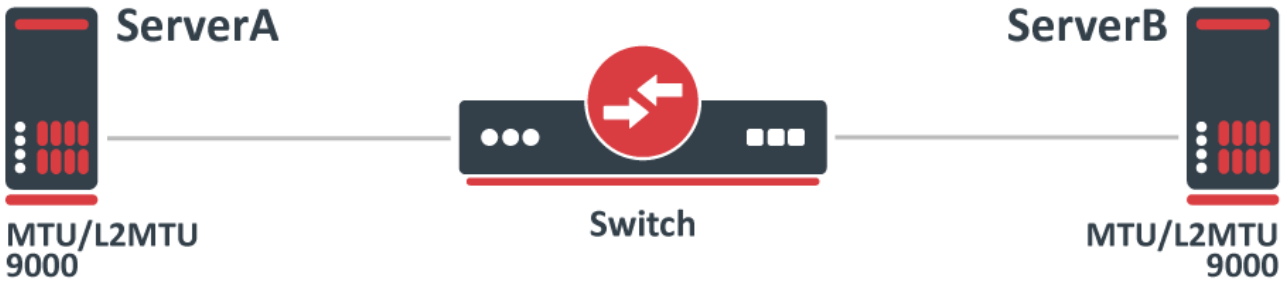
Solution

Increase the L2MTU on slave interfaces before changing the MTU on a master interface.

```
/interface ethernet
set ether1,ether2 l2mtu=9018
/interface vlan
set VLAN99 mtu=9000
```

MTU inconsistency

Consider the following scenario, you have multiple devices in your network, most of them are used as a switch/bridge in your network and there are certain endpoints that are supposed to receive and process traffic. To decrease the overhead in your network, you have decided to increase the MTU size so you set a larger MTU size on both endpoints, but you start to notice that some packets are being dropped.



Configuration

In this case, both endpoints can be any type of device, we will assume that they are both Linux servers that are supposed to transfer a large amount of data. In such a scenario, you would have probably set interface MTU to 9000 on **ServerA** and **ServerB a**nd on your **Switch** you have probably have set something similar to this:

```
/interface bridge
add name=bridge1
/interface bridge port
add interface=ether1 bridge=bridge1
add interface=ether2 bridge=bridge1
```

Problem

This is a very simplified problem, but in larger networks, this might not be very easy to detect. For instance, ping might be working since a generic ping packet will be 70 bytes long (14 bytes for Ethernet header, 20 bytes for IPv4 header, 8 bytes for ICMP header, 28 bytes for ICMP payload), but data transfer might not work properly. The reason why some packets might not get forwarded is that MikroTik devices running RouterOS by default has MTU set to 1500 and L2MTU set to something around 1580 bytes (depends on the device), but the Ethernet interface will silently drop anything that does not fit into the L2MTU size. Note that the L2MTU parameter is not relevant to x86 or CHR devices. For a device that is only supposed to forward packets, there is no need to increase the MTU size, it is only required to increase the L2MTU size, RouterOS will not allow you to increase the MTU size that is larger than the L2MTU size. If you require the packet to be received on the interface and the device needs to process this packet rather than just forwarding it, for example, in the case of routing, then it is required to increase the L2MTU and the MTU size, but you can leave the MTU size on the interface to the default value if you are using only IP traffic (that supports packet fragmentation) and don't mind that packets are being fragmented. You can use the ping utility to make sure that all devices are able to forward jumbo frames:

```
/ping 192.168.88.1 size=9000 do-not-fragment
```

Remember that the L2MTU and MTU size needs to be larger or equal to the ping packet size on the device from which and to which you are sending a ping packet since ping (ICMP) is IP traffic that is sent out from an interface over Layer3.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Web pages are not able to load up, but ping works properly;
- Tunnels dropping traffic;
- Specific protocols are broken;
- Large packet loss;

Solution

Increase the L2MTU size on your **Switch**:

```
/interface ethernet
set ether1,ether2 l2mtu=9000
```

In case your traffic is encapsulated (VLAN, VPN, MPLS, VPLS, or other), then you might need to consider setting an even larger L2MTU size. In this scenario, it is not needed to increase the MTU size for the reason described above.

Full frame MTU is not the same as L2MTU. L2MTU size does not include the Ethernet header (14 bytes) and the CRC checksum (FCS) field. The FCS field is stripped by the Ethernet's driver and RouterOS will never show the extra 4 bytes to any packet. For example, if you set MTU and L2MTU to 9000, then the full-frame MTU is 9014 bytes long, this can also be observed when sniffing packets with "`/tool sniffer quick`" command.

Bridge and reserved MAC addresses

Consider the following scenario, you want to transparently bridge two network segments together, either those are tunnel interfaces like EoIP, Wireless interfaces, Ethernet interface, or any other kind of interfaces that can be added to a bridge. Such a setup allows you to seamlessly connect two devices together like there was only a physical cable between them, this is sometimes called a **transparent bridge** from **DeviceA** to **DeviceB**.

Configuration

For both devices **DeviceA** and **DeviceB** there should be a very similar configuration.

```
/interface bridge
add name=bridge1 protocol-mode=rstp
/interface bridge port
add interface=ether1 bridge=bridge1
add interface=eoip1 bridge=bridge1
```

Problem

Both devices are able to communicate with each other, but some protocols do not work properly. The reason is that as soon as you use any STP variant (STP, RSTP, MSTP), you make the bridge compliant with IEEE 802.1D and IEEE 802.1Q, these standards recommend that packets that are destined to **01:80:C2:00:00:0X** should **NOT** be forwarded. In cases where there are only 2 ports added to a bridge (R/M)STP should not be used since a loop cannot occur from 2 interfaces and if a loop does occur, the cause is elsewhere and should be fixed on a different bridge. Since (R/M)STP is not needed in transparent bridge setups, it can be disabled. As soon as (R/M)STP is disabled, the RouterOS bridge is not compliant with IEEE 802.1D and IEEE 802.1Q and therefore will forward packets that are destined to **01:80:C2:00:00:0X**.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- LLDP neighbors not showing up;
- 802.1x authentication (dot1x) not working;
- LACP interface not passing traffic;

Solution

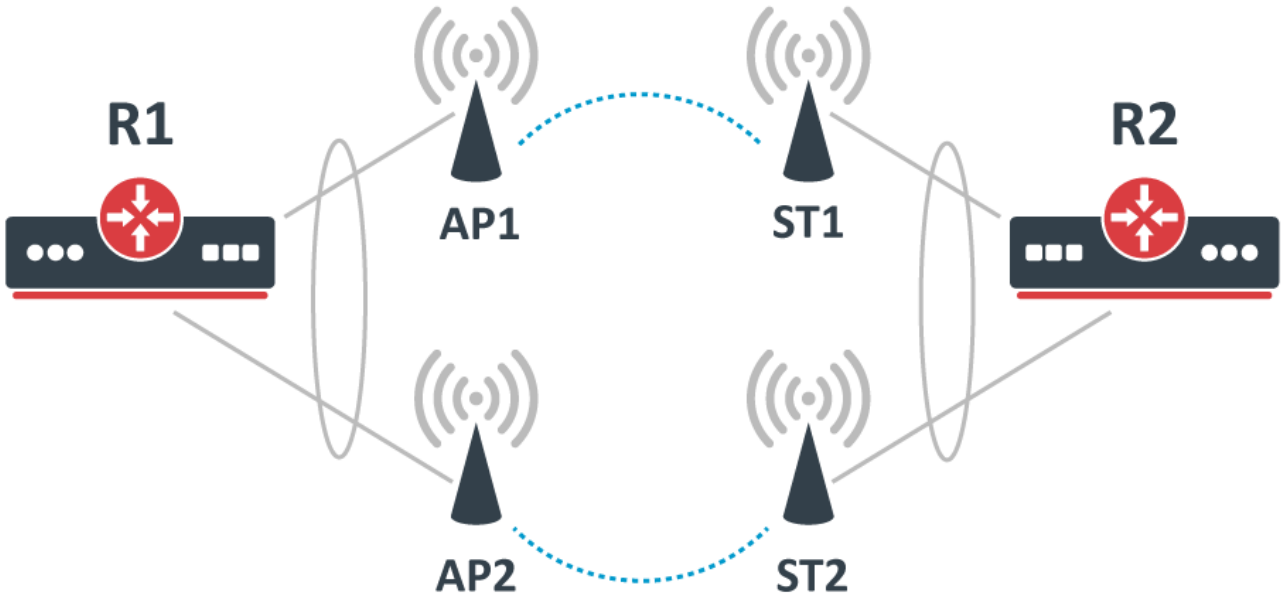
Since RouterOS v6.43 it is possible to partly disable compliance with IEEE 802.1D and IEEE 802.1Q, this can be done by changing the bridge protocol mode.

```
/interface bridge
set bridge1 protocol-mode=none
```

The IEEE 802.1x standard is meant to be used between a switch and a client directly. If it is possible to connect a device between the switch and the client, then this creates a security threat. For this reason, it is not recommended to disable the compliance with IEEE 802.1D and IEEE 802.1Q, but rather design a proper network topology.

Bonding between Wireless links

Consider the following scenario, you have set up multiple Wireless links and to achieve maximum throughput and yet to achieve redundancy you have decided to place Ethernet interfaces into a bond and depending on the traffic that is being forwarded you have chosen a certain bonding mode. This scenario can be applied to any case, where a bonding interface is created between links, that are not directly connected to each other.



Configuration

The following configuration is relevant to **R1** and **R2**:

```
/interface bonding
add mode=802.3ad name=bond1 slaves=ether1,ether2 transmit-hash-policy=layer-2-and-3
/ip address
add address=192.168.1.X/24 interface=bond1
```

While the following configuration is relevant to **AP1**, **AP2**, **ST1,** and **ST2**, where **X** corresponds to an IP address for each device.

```
/interface bridge
add name=bridge1 protocol-mode=none
/interface bridge port
add interface=ether1 bridge=bridge1
add interface=wlan1 bridge=bridge1
/ip address
add address=192.168.1.X/24 interface=bridge1
```

Problem

While traffic is being forwarded properly between **R1** and **R2**, load balancing, link failover is working properly as well, but devices between **R1** and **R2** are not always accessible or some of them are completely inaccessible (in most cases **AP2** and **ST2** are inaccessible). After examining the problem you might notice that packets do not always get forwarded over the required bonding slave and as a result, never is received by the device you are trying to access. This is a network design and bonding protocol limitation. As soon as a packet needs to be sent out through a bonding interface (in this case you might be trying to send ICMP packets to **AP2** or **ST2**), the bonding interface will create a hash based on the selected bonding mode and transmit-hash-policy and will select an interface, through which to send the packet out, regardless of the destination is only reachable through a certain interface. Some devices will be accessible because the generated hash matches the interface, on which the device is located on, but it might not choose the needed interface as well, which will result in inaccessible device. Only broadcast bonding mode does not have this kind of protocol limitation, but this bonding mode has a very limited use case.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Limited connectivity
- Unstable links (in case of balance-rr)

Solution

Bonding interfaces are not supposed to be connected using in-direct links, but it is still possible to create a workaround. The idea behind this workaround is to find a way to bypass packets being sent out using the bonding interface. There are multiple ways to force a packet not to be sent out using the bonding interface, but essentially the solution is to create new interfaces on top of physical interfaces and add these newly created interfaces to a bond instead of the physical interfaces. One way to achieve this is to create EoIP tunnels on each physical interface, but that creates a huge overhead and will reduce overall throughput. You should create a VLAN interface on top of each physical interface instead, this creates a much smaller overhead and will not impact overall performance noticeably. Here is an example of how **R1** and **R2** should be reconfigured:

```
/interface vlan
add interface=ether1 name=VLAN_ether1 vlan-id=999
add interface=ether2 name=VLAN_ether2 vlan-id=999
/interface bonding
add mode=balance-xor name=bond1 slaves=VLAN_ether1,VLAN_ether2 transmit-hash-policy=layer-2-and-3
/ip address
add address=192.168.1.X/24 interface=bond1
add address=192.168.11.X/24 interface=ether1
add address=192.168.22.X/24 interface=ether2
```

AP1 and **ST1** only need updated IP addresses to the correct subnet:

```
/ip address
add address=192.168.11.X/24 interface=bridge1
```

Same changes must be applied to **AP2** and **ST2** (make sure to use the correct subnet):

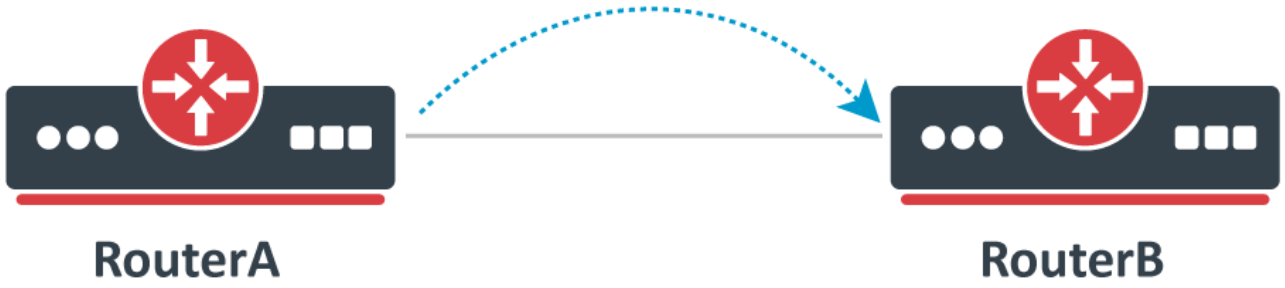
```
/ip address
add address=192.168.22.X/24 interface=bridge1
```

With this approach, you create the least overhead and the least configuration changes are required.

> ⚠ LACP (802.3ad) is not mean to be used in setups, where devices bonding slaves are not directly connected, in this case, it is not recommended to use LACP if there are Wireless links between both routers. LACP requires both bonding slaves to be at the same link speeds, Wireless links can change their rates at any time, which will decrease overall performance and stability. Other bonding modes should be used instead.

Bandwidth testing

Consider the following scenario, you set up a link between two devices, this can be any link, an Ethernet cable, a wireless link, a tunnel or any other connection. You decide that you want to test the link's bandwidth, but for convenience reasons, you decide to start testing the link with the same devices that are running the link.



Problem

As soon as you start Bandwidth test or Traffic generator you notice that the throughput is much smaller than expected. For very powerful routers, which should be able to forward many Gigabits per second (Gbps) you notice that only a few Gigabits per second gets forwarded. The reason why this is happening is because of the testing method you are using, you should never test throughput on a router while using the same router for generating traffic because you are adding an additional load on the CPU that reduces the total throughput.

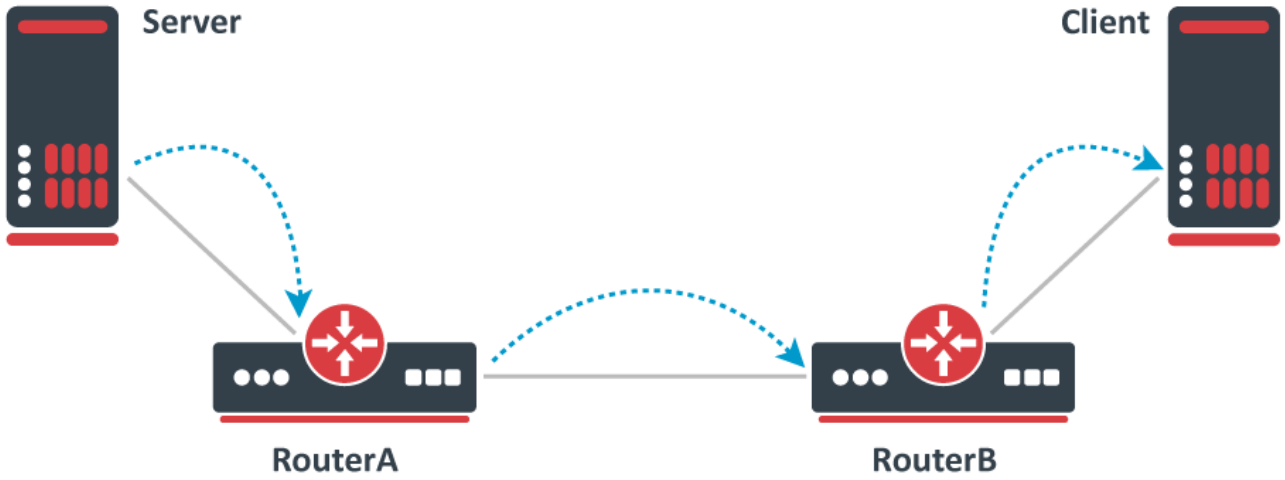
Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Low throughput;
- High CPU usage;

Solution

Use a proper testing method. Don't use Bandwidth-test to test large capacity links and don't run any tool that generates traffic on the same device you are testing. Design your network properly so you can attach devices that will generate and receive traffic on both ends. If you are familiar with **Iperf**, then this concept should be clear. Remember that in real-world a router or a switch does not generate large amounts of traffic (at least it shouldn't, otherwise, it might indicate an existing security issue), a server/client generates the traffic while a router/switch forwards the traffic (and does some manipulations to the traffic in appropriate cases).



Bridge split-horizon usage

Consider the following scenario, you have a bridge and you need to isolate certain bridge ports from each other. There are options to use a built-in switch chip to isolate certain ports on certain switch chips, you can use bridge firewall rules to prevent certain ports to be able to send any traffic to other ports, you can isolate ports in a PVLAN type of setup using port isolation, but there is also a software-based solution to use bridge split-horizon (which disables hardware offloading on all switch chips).

Configuration

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 horizon=1 hw=no interface=ether1
add bridge=bridge1 horizon=2 hw=no interface=ether2
add bridge=bridge1 horizon=3 hw=no interface=ether3
add bridge=bridge1 horizon=4 hw=no interface=ether4
```

Problem

After setting the bridge split-horizon on each port, you start to notice that each port is still able to send data between each other. The reason for this is the misuse of bridge split-horizon. A bridge port is only not able to communicate with ports that are in the same horizon, for example, horizon=1 is not able to communicate with horizon=1, but is able to communicate with horizon=2, horizon=3, and so on.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- Traffic is being forwarded on different bridge split-horizons

Solution

Set a proper value as the bridge split-horizon. In case you want to isolate each port from each other (a common scenario for PPPoE setups) and each port is only able to communicate with the bridge itself, then all ports must be in the same bridge split-horizon.

```
/interface bridge port
set [f] horizon=1
```

⚠ Setting all bridge ports in the same bridge split-horizon will result in traffic being only able to reach the bridge interface itself, then packets can only be routed. This is useful when you want other devices to filter out certain traffic. Similar behavior can be achieved using bridge filter rules.

Unsupported SFP modules

Consider the following scenario, you have decided to use optical fiber cables to connect your devices together by using SFP or SFP+ optical modules, but for convenience reasons, you have decided to use SFP optical modules that were available.

Problem

As soon as you configure your devices to have connectivity on the ports that are using these SFP optical modules, you might notice that either the link is working properly or experiencing random connectivity issues. There are many vendors that manufacture SFP optical modules, but not all vendors strictly follow SFP MSA, SFF, and IEEE 802.3 standards, which can lead to unpredictable compatibility issues, which is a very common issue when using not well known or unsupported SFP optical modules in MikroTik devices.

Symptoms

Below is a list of possible symptoms that might be a result of this kind of misconfiguration:

- SFP interface does not link up
- Random packet drop
- Unstable link (flapping)
- SFP module not running after a reboot
- SFP module not running after power-cycle
- SFP module running only on one side

Solution

You should only use supported SFP modules. Always check the SFP compatibility table if you are intending to use SFP modules manufactured by MikroTik. There are other SFP modules that do work with MikroTik devices as well, check the Supported peripherals table to find other SFP modules that have been confirmed to work with MikroTik devices. Some unsupported modules might not be working properly at certain speeds and with auto-negotiation, you might want to try to disable it and manually set a link speed.

Loop Protect

Loop protect feature can prevent Layer2 loops by sending loop protect protocol packets and shutting down interfaces in case they receive loop protect packets originated from themselves. The feature works by checking the source MAC address of the received loop protect packet against MAC addresses of loop protect enabled interfaces. If the match is found, loop protect disables the interface which received the loop protect packet. Log message warns about this event and interface is marked with a loop protect comment by the system. RouterOS loop protect feature can be used on bridged interfaces as well as on ethernet interfaces which are set for switching in RouterBoard switch chips.

Loop Protect works on Ethernet, VLAN, EoIP, VxLAN interfaces and its packets are encapsulated with EtherType 0x9003.
There is support for adjusting loop protect packet sending interval and interface disable time. Configuration changes or expiration of disable time, resets loop protection on an interface.

⚠ Even though loop-protect can work on interfaces that are added to a bridge, it is still recommended to use (R/M)STP rather than loop-protect since (R/M)STP is compatible with most switches STP variants provide much more configuration options to fine-tune your network.

Sub-menu: `/interface ethernet` `/interface vlan` `/interface eoip` `/interface eoipv6` `/interface vxlan`

Property	Description
loop-protect (*on* | *off* | *default*; Default: **default**)	Enables or disables loop protect on the selected interface. **default** works as turned off.
loop-protect-send-interval (*time interval*; Default: **5s**)	Sets how often loop protect packets are sent on selected interface.
loop-protect-disable-time (*time interval* | *0*; Default: **5m**)	Sets how long selected interface is disabled when loop is detected. **0** - forever.

Read-only properties

Property	Description
loop-protect-status (*on* | *off* | *disable*)	- **on** - loop-protect feature is turned on, the interface is sending and listening for loop protect packets - **off** - loop-protect feature is turned off - **disable** - loop-protect feature is turned on, the interface has received loop protect packet and disabled itself to prevent loop.

QoS with Switch Chip

Introduction

Queues in RouterOS are processed using CPU resources, so limiting traffic with queues on the devices with relatively weak CPU is not an effective configuration. In other words, switch-based units will be overloaded very soon, because they are meant to process layer 2 traffic. To avoid such inefficiency, RouterOS allows limiting traffic using switch chips.

CRS3xx, CRS5xx series, and CCR2116, CCR2216 devices

> This paragraph applies to CCR2116, CCR2216 devices and CRS3xx, CRS5xx series switches, not to CRS1xx/CRS2xx series switches!

For CRS3xx series switches, it is possible to limit ingress traffic that matches certain parameters with ACL rules and it is possible to limit ingress/egress traffic per port basis. The policer is used for ingress traffic, the shaper is used for egress traffic. The ingress policer controls the received traffic with packet drops. Everything that exceeds the defined limit will get dropped. This can affect the TCP congestion control mechanism on end hosts and achieved bandwidth can be actually less than defined. The egress shaper tries to queue packets that exceed the limit instead of dropping them. Eventually, it will also drop packets when the output queue gets full, however, it should allow utilizing the defined throughput better.

Port-based traffic police (ingress) and shaper (egress):

```
/interface ethernet switch port
set ether1 ingress-rate=10M egress-rate=5M
```

MAC-based traffic policer:

```
/interface ethernet switch rule
add ports=ether1 switch=switch1 src-mac-address=64:D1:54:D9:27:E6/FF:FF:FF:FF:FF:FF rate=10M
```

VLAN-based traffic policer:

```
/interface bridge
set bridge1 vlan-filtering=yes
/interface ethernet switch rule
add ports=ether1 switch=switch1 vlan-id=11 rate=10M
```

Protocol-based traffic policer:

```
/interface ethernet switch rule
add ports=ether1 switch=switch1 mac-protocol=ipx rate=10M
```

CRS1xx/CRS2xxSeries devices

> This subsection does not apply to CRS3xx series devices!

Configuration schemes

MAC based traffic scheduling and shaping: [MAC address in UFDB] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

VLAN based traffic scheduling and shaping: [VLAN id in VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

Protocol based traffic scheduling and shaping: [Protocol in Protocol VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

PCP/DEI based traffic scheduling and shaping: [Switch port PCP/DEI mapping] -> [Priority] -> [Queue] -> [Shaper]

DSCP based traffic scheduling and shaping: [QoS DSCP mapping] -> [Priority] -> [Queue] -> [Shaper]

MAC based traffic scheduling using internal Priority

In Strict Priority scheduling mode, the highest priority queue is served first. The queue number represents the priority and the queue with highest queue number has the highest priority. Traffic is transmitted from highest priority queue until the queue is empty, and then moves to the next highest priority queue, and so on. If no congestion is present on the egress port, packet is transmitted as soon as it is received. If congestion occurs on the port where high priority traffics keep coming, the lower priority queues starve.

On all CRS switches the scheme where MAC based egress traffic scheduling is done according to internal Priority would be following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue];
In this example, host1 (E7:16:34:00:00:01) and host2 (E7:16:34:00:00:02) will have higher priority 1 and the rest of the hosts will have lower priority 0 for transmitted traffic on port ether7. Note that CRS has a maximum of 8 queues per port.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create a QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entries to match specific MACs on ether7 and apply QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:00:00:01 port=ether7 qos-group=group1 svl=yes
add mac-address=E7:16:34:00:00:02 port=ether7 qos-group=group1 svl=yes
```

Configure ether7 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
prior\
    ity:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 \
    qos-scheme-precedence=da-based
```

MAC based traffic shaping using internal Priority

The scheme where MAC based traffic shaping is done according to internal Priority would be following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper];
In this example, unlimited traffic will have priority 0 and limited traffic will have priority 1 with the bandwidth limit 10Mbit. Note that CRS has a maximum of 8 queues per port.

Create a group of ports for switching:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entry to match specific MAC on ether8 and apply QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:A1:CD:18 port=ether8 qos-group=group1 svl=yes
```

Configure ether8 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
prior\
    ity:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 \
    qos-scheme-precedence=da-based
```

Apply bandwidth limit for queue1 on ether8:

```
/interface ethernet switch shaper
add port=ether8 rate=10M target=queue1
```

If CRS switch supports Access Control List, this configuration is simpler:

```
/interface ethernet switch acl policer
add name=policer1 yellow-burst=100k yellow-rate=10M

/interface ethernet switch acl
add mac-dst-address=E7:16:34:A1:CD:18 policer=policer1
```

VLAN based traffic scheduling + shaping using internal Priorities

A best practice is to assign lower internal QoS Priority for traffic limited by shaper to make it also less important in the Strict Priority scheduler. (higher priority should be more important and unlimited)

In this example:
Switch port ether6 is using a shaper to limit the traffic that comes from ether7 and ether8.
When a link has reached its capacity, the traffic with the highest priority will be sent out first.
VLAN10 -> QoS group0 = lowest priority
VLAN20 -> QoS group1 = normal priority
VLAN30 -> QoS group2 = highest priority

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create QoS groups for use in the VLAN table:

```
/interface ethernet switch qos-group
add name=group0 priority=0
add name=group1 priority=1
add name=group2 priority=2
```

Add VLAN entries to apply QoS groups for certain VLANs:

```
/interface ethernet switch vlan
add ports=ether6,ether7,ether8 qos-group=group0 vlan-id=10
add ports=ether6,ether7,ether8 qos-group=group1 vlan-id=20
add ports=ether6,ether7,ether8 qos-group=group2 vlan-id=30
```

Configure ether6, ether7, ether8 port queues to work according to Strict Priority and QoS scheme only for VLAN based QoS:

```
/interface ethernet switch port
set ether6 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
prior\
    ity:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 \
    qos-scheme-precedence=vlan-based
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
prior\
    ity:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 \
    qos-scheme-precedence=vlan-based
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
prior\
    ity:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 \
    qos-scheme-precedence=vlan-based
```

Apply bandwidth limit on ether6:

```
/interface ethernet switch shaper
add port=ether6 rate=10M
```

PCP based traffic scheduling

By default, CRS1xx/CRS2xx series devices will ignore the PCP/CoS/802.1p value and forward packets based on FIFO (First-In-First-Out) manner. When the device's internal queue is not full, then packets are in a FIFO manner, but as soon as a queue is filled, then higher priority traffic can be sent out first. Let's consider a scenario when **ether1** and **ether2** is forwarding data to **ether3**, but when **ether3** is congested, then packets are going to be scheduled, we can configure the switch to hold lowest priority packets until all higher priority packets are sent out, this is a very common scenario for VoIP type setups, where some traffic needs to be prioritized.

To achieve such a behavior, switch together **ether1**, **ether2**, and **ether3** ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
```

Enable **Strict Policy** for each internal queue on each port:

```
/interface ethernet switch port
set ether1,ether2,ether3 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0"
```

Map each PCP value to an internal priority value, for convenience reasons simply map PCP to an internal priority 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 pcp-based-qos-priority-mapping=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Since the switch will empty the largest queue first and you need the highest priority to be served first, then you can assign this internal priority to a queue 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 priority-to-queue=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Finally, set each switch port to schedule packets based on the PCP value:

```
/interface ethernet switch port
set ether1,ether2,ether3 qos-scheme-precedence=pcp-based
```

Bandwidth Limiting

Both Ingress Port policer and Shaper provide bandwidth limiting features for CRS switches:

Ingress Port Policer sets RX limit on port:

```
/interface ethernet switch ingress-port-policer
add port=ether5 meter-unit=bit rate=10M
```

Shaper sets TX limit on port:

```
/interface ethernet switch shaper
add port=ether5 meter-unit=bit rate=10M
```

Traffic Storm Control

The same Ingress Port policer also can be used for the traffic storm control to prevent disruptions on Layer 2 ports caused by broadcast, multicast or unicast traffic storms.

Broadcast storm control example on ether5 port with 500 packet limit per second:

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=500 meter-unit=packet packet-types=broadcast
```

Example with multiple packet types that includes ARP and ND protocols and unregistered multicast traffic. Unregistered multicast is the traffic which is not defined in the Multicast Forwarding database:

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=5k meter-unit=packet packet-types=broadcast,arp-or-nd,unregistered-multicast
```

Spanning Tree Protocol

Summary

The purpose of spanning tree protocol is to provide the ability to create loop-free Layer 2 topologies while having redundant links. While connecting multiple bridges or just cross-connecting bridge ports, it's possible to create network loops that can severely impact the stability of the network. Spanning tree protocol aims to resolve this problem by introducing the concept of the root bridge, all bridges in the same Layer 2 domain will exchange information about the shortest path to the root bridge. Afterward, each bridge will negotiate which ports to use to reach the root bridge. This information exchange is done with the help of Bridge Protocol Data Units (BPDUs). STP will disable certain ports for each bridge in order to avoid loops, while still ensuring that all bridges can communicate with each other. For an in-depth description of protocol please refer to IEEE 802.1D.

As a best practice, it is always recommended to manually set up each bridge's priority, port priority, and port path cost to ensure proper Layer2 functionality at all times. Leaving STP related values to defaults are acceptable for a network that consists of 1 to 2 bridges running with (R/M)STP enabled, but it is highly recommended to manually set these values for larger networks. Since STP elects a root bridge and root ports by checking STP related values from bridges over the network, then leaving STP settings to automatic may elect an undesired root bridge and root ports and in case of a hardware failure can result in an inaccessible network.

> RouterOS bridge does not work with PVST and its variants. The PVST BPDUs (with a MAC destination 01:00:0C:CC:CC:CD) are treated by RouterOS bridges as typical multicast packets. In simpler terms, they undergo RouterOS bridge/switch forwarding logic and may get tagged or untagged.

Monitoring

You can check the STP status of a bridge by using the `/interface bridge monitor` command, for example:

```
/interface bridge monitor bridge1
                  state: enabled
    current-mac-address: B8:69:F4:30:19:FE
            root-bridge: no
         root-bridge-id: 0x1000.B8:69:F4:30:19:FD
         root-path-cost: 4000
              root-port: sfp-sfpplus2
             port-count: 2
  designated-port-count: 1
           fast-forward: yes
```

Note that the root bridge doesn't have any root ports, only designated ports.

You can check the STP status of a bridge port by using the `/interface bridge port monitor` command, for example:

```
/interface bridge port monitor [find interface=sfp-sfpplus2]
              interface: sfp-sfpplus2
                 status: in-bridge
            port-number: 1
                   role: root-port
              edge-port: no
    edge-port-discovery: yes
    point-to-point-port: yes
           external-fdb: no
           sending-rstp: yes
               learning: yes
             forwarding: yes
              path-cost: 2000
         root-path-cost: 4000
      designated-bridge: 0x8000.DC:2C:6E:9E:11:1C
        designated-cost: 2000
  designated-port-number: 2
```

Note that `root-bridge-id` consists of the bridge priority and the bridge's MAC address, for non-root bridges the root bridge will be shown as `designated-bridge`. One port can have one role in an STP enabled network, below is a list of possible port roles:

- **root-port** - port that is facing towards the root bridge and will be used to forward traffic from/to the root bridge.
- **alternate-port** - port that is facing towards root bridge, but is not going to forward traffic (a backup for root port).
- **backup-port** - port that is facing away from the root bridge, but is not going to forward traffic (a backup for non-root port).
- **designated-port** - port that is facing away from the root bridge and is going to forward traffic.
- **disabled-port** - disabled or inactive port.

When using bridges that are set to use 802.1Q as EtherType, they will send out BPDUs to 01:80:C2:00:00:00, which are used by MSTP, RSTP, and STP. When using 802.1ad as bridge VLAN protocol, the BPDUs are not compatible with 802.1Q bridges and they are sent to 01:80:C2:00:00:08. (R/M)STP will not function properly if there are different bridge VLAN protocols across the Layer2 network.

STP and RSTP

STP and Rapid STP are used widely across many networks, but almost all networks have switched over using only RSTP since of its benefits. STP is a very old protocol and has a convergence time (the time needed to fully learn network topology changes and to continue properly forwarding traffic) of up to 50 seconds. RSTP has a lot of smaller convergence time, a few seconds or even a few milliseconds. It is recommended to use RSTP instead of STP since it is a lot faster and is also backward compatible with STP. One of the reasons why RSTP is faster is because of reduced possible port states, below is a list of possible STP port states:

- **Forwarding** - port participates in traffic forwarding and is learning MAC addresses, is receiving BPDUs.
- **Listening** - port does not participate in traffic forwarding and is not learning MAC addresses, is receiving BPDUs.
- **Learning** - port does not participate in traffic forwarding but is learning MAC addresses.
- **Blocking** - port is blocked since it is causing loops but is receiving BPDUs.
- **Disabled** - port is disabled or inactive.

In RSTP the disabled, listening and blocking port states are replaced with just one state called the **Discarding** state:

- **Forwarding** - port participates in traffic forwarding and is learning MAC addresses, is receiving BPDUs (forwarding=yes).
- **Learning** - port does not participate in traffic forwarding but is learning MAC addresses (learning=yes).
- **Discarding** - port does not participate in traffic forwarding and is not learning MAC addresses, is receiving BPDUs (forwarding=no).

In STP connectivity between bridges is determined by sending and receiving BPDUs between neighbor bridges. Designated ports are sending BPDUs to root ports. If a BPDU is not received 3 times the **HelloTime** in a row, then the connection is considered as unavailable and network topology convergence will commence. It is possible for STP to reduce the convergence time in certain scenarios by reducing the `forward-delay` timer, which is responsible for how long can the port be in the learning/listening state.

In RouterOS, it is possible to specify which bridge ports are edge ports. Edge ports are ports that are not supposed to receive any BPDUs, this is beneficial since this allows STP to skip the learning and the listening state and directly go to the forwarding state. This feature is sometimes called **PortFast**. You can leave this parameter to the default value, which is **auto**, but you can also manually specify it, you can set a port as edge port manually for ports that should not have any more bridges behind it, usually these are access ports.

Additionally, bridge port `point-to-point` , specifies if a bridge port is connected to a bridge using a point-to-point link for faster convergence in case of failure. By setting this property to `yes`, you are forcing the link to be a point-to-point link, which will skip the checking mechanism, which detects and waits for BPDUs from other devices from this single link, by setting this property to `no`, you are implying that a link can receive BPDUs from multiple devices. By setting the property to `yes`, you are significantly improving (R/M)STP convergence time. In general, you should only set this property to `no` , if it is possible that another device can be connected between a link, this is mostly relevant to Wireless mediums and Ethernet hubs. If the Ethernet link is full-duplex, `auto` enables point-to-point functionality. This property has no effect when protocol-mode is set to `none`.

Default values

When creating a bridge or adding a port to the bridge the following are the default values that are assigned by RouterOS:

- Default bridge priority: **32768** / **0x8000**
- Default bridge port path cost: **based on interface speed**
- Default bridge port priority: **0x80**
- BPDU message age increment: **1**
- HelloTime: **2**
- Default max message age: **20**

The bridge interface setting `port-cost-mode` changes the port path-cost and internal-path-cost mode for bridged ports, utilizing automatic values based on interface speed. This setting does not impact bridged ports with manually configured `path-cost` or `internal-path-cost` properties. Below are examples illustrating the path-costs corresponding to specific data rates (with proportionate calculations for intermediate rates):

Data rate	Long	Short
10 Mbps	2,000,000	100
100 Mbps	200,000	19
1 Gbps	20,000	4
10 Gbps	2,000	2
25 Gbps	800	1
40 Gbps	500	1
50 Gbps	400	1
100 Gbps	200	1

For bonded interfaces, the highest path-cost among all bonded member ports is applied, this value remains unaffected by the total link speed of the bonding. For virtual interfaces (such as VLAN, EoIP, VXLAN), as well as wifi, wireless, and 60GHz interfaces, a path-cost of 20,000 is assigned for long mode, and 10 for short mode. For dynamically bridged interfaces (e.g. wifi, wireless, PPP, VPLS), the path-cost defaults to 20,000 for long mode and 10 for short mode. However, this can be manually overridden by the service that dynamically adds interfaces to bridge, for instance, by using the CAPsMAN `data path.bridge-cost` setting. RouterOS versions prior to 7.13 does not change port path cost based on the link speed, for 10M, 100M, 1000M, and 10000M link speeds the default path cost value when a port is added to a bridge was always **10**.

The age of a BPDU is determined by how many bridges have the BPDU passed times the message age since RouterOS uses **1** as the message age increment, then the BPDU packet can pass as many bridges as specified in the `max-message-age` parameter. By default this value is set to **20**, this means that after the 20th bridge the BPDU packet will be discarded and the next bridge will become a root bridge, note that if `max-message-age=20`on is set, then it is hard to predict which ports will be the designated port on the 21st bridge and may result in traffic not being able to be forwarded properly.

> ⚠ In case bridge filter rules are used, make sure you allow packets with DST-MAC address **01:80:C2:00:00:00** since these packets carry BPDUs that are crucial for STP to work properly.

Election process

To properly configure STP in your network you need to understand the election process and which parameters are involved in which order. In RouterOS the root bridge will be elected based on the smallest priority and the smallest MAC address in this particular order:

1. Bridge priority (lowest)
2. Bridge MAC address (lowest)

In RouterOS root ports are elected based on lowest Root port path cost, lowest bridge identifier, and lowest bridge port ID in this particular order:

1. Root port path cost (lowest)
2. Bridge identifier (lowest)
3. Bridge port ID (lowest)

First, when the device considers which of its ports to elect as the root port, it will check the **root path cost** seen by its ports. If root path cost is the same for two or more ports then the **Bridge identifier** of the **upstream** device will be checked and port connected to the lowest bridge identifier will become the root port. If the same bridge identifier is seen on two or more ports, then the **Bridge port ID** of the **upstream** device will be checked.

Explanation of attributes:

Root path cost, all bridges have a Root Path Cost. Root bridge has a root path cost of 0. For all other Bridges, it is the sum of the Port Path Costs on the least-cost path to the Root Bridge. You can modify local port path cost under "/interface bridge port".

Bridge identifier is a combination of "bridge priority" and "bridge MAC", configurable under "/interface bridge"

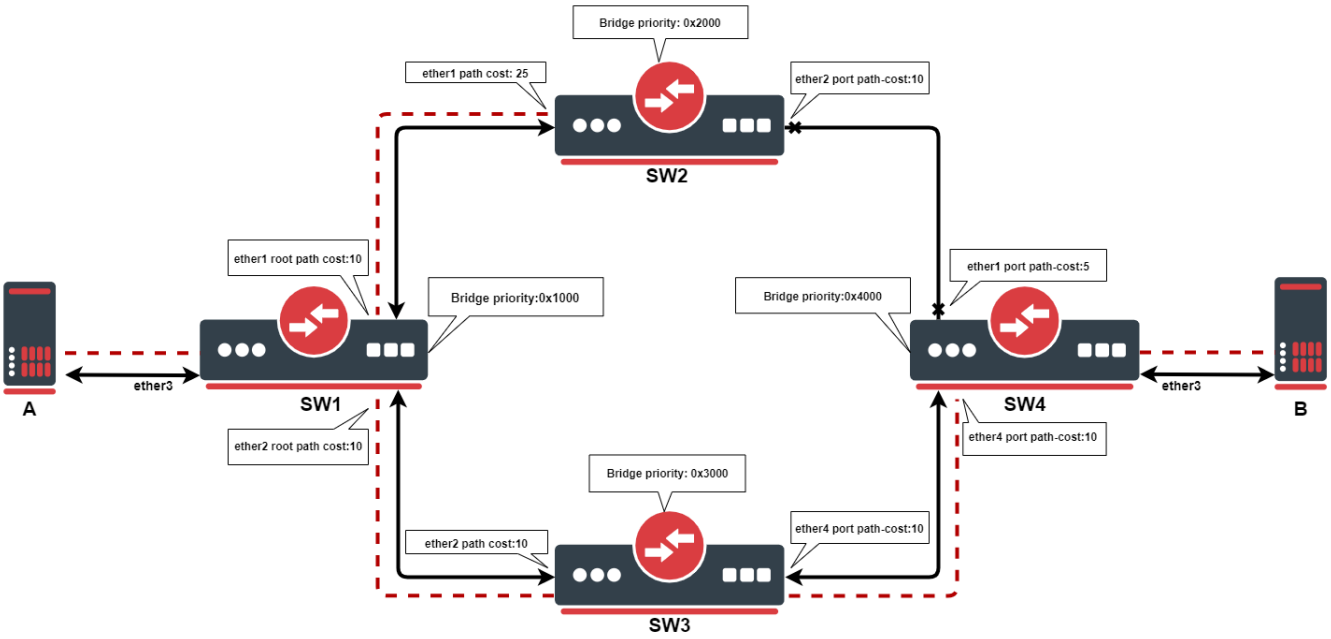
Bridge port ID is a combination of "unique ID" and "bridge port priority", the unique ID is automatically assigned to bridge port upon adding it to the bridge, it cannot be edited. It can be seen in WinBox under "Bridge Port" "Port Number" column, or with "/interface bridge port monitor", as "port-number".

Make sure you are using path cost and priority on the right ports. For example, setting path cost on ports that are in a root bridge has no effect, only port priority has an effect on them. Root path cost has an effect on ports that are facing towards the root bridge and port priority has an effect on ports that are facing away from the root bridge. And bridge identifier doesn't impact the device's own root port election, instead, it affects the root port election for downstream devices.

In RouterOS it is possible to set any value for bridge priority between 0 and 65535, the IEEE 802.1W standard states that the bridge priority must be in steps of 4096. This can cause incompatibility issues between devices that do not support such values. To avoid incompatibility issues, it is recommended to use only these priorities: 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672, 32768, 36864, 40960, 45056, 49152, 53248, 57344, 61440.

Examples

Root path cost example



This example outlines how the root path cost works. SW1 will be the root bridge, due to it having the lowest priority of 0x1000, as the root bridge. Each bridge will calculate the path cost to the root bridge. When calculating root path cost bridges take into account configured path cost on their ports + root path cost advertised by neighboring bridges.

SW1: due to it being the root bridge, it advertises root path cost of 0 to its neighbors, even though it has a configured path cost of 10.

SW2: ether1. has root path cost of 0 + 25=**25**. On the **ether2** path cost will be 10+10+10+0=**30**

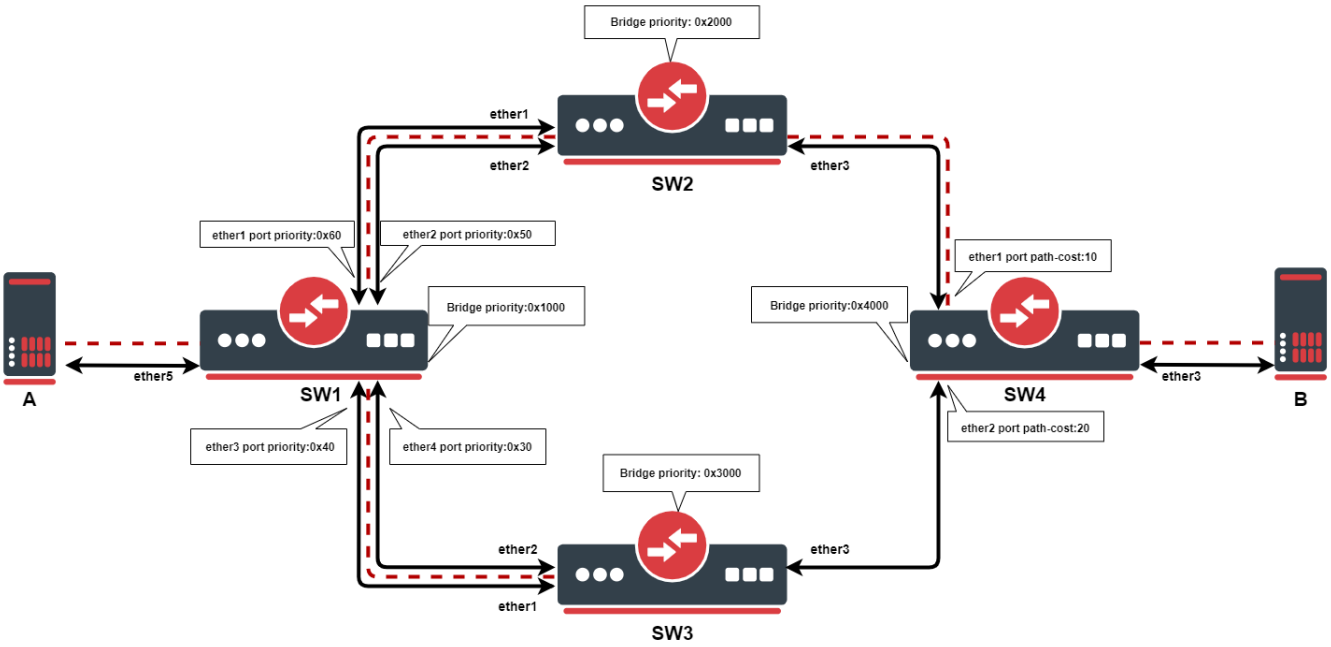
SW3: **ether2**, has root path cost of 0 + 10=**10**. On the **ether4** path cost will be 10+5+25+0=**40**

SW4: **ether1**, has root path cost of 0+25+5=**30**. On **ether4** path cost will be 10+10+0=**20**

Port with the lowest path cost will be elected as the root port. Every bridge in STP topology needs a path to root bridge, after the best path has been found, the redundant path will be blocked, in this case, path between SW2 and SW4.

> ⚠ You can configure path cost on the root bridge, but it will only be taken into account when the bridge loses its root status.

STP example



In this example, we want to ensure Layer2 redundancy for connections from ServerA to ServerB. If a port is connected to a device that is not a bridge and not running (R)STP, then this port is considered as an edge port, in this case ServerA and ServerB is connected to an edge port. This is possible by using STP in a network. Below are configuration examples for each switch.

- Configuration for SW1:

```
/interface bridge
add name=bridge priority=0x1000
/interface bridge port
add bridge=bridge interface=ether1 priority=0x60
add bridge=bridge interface=ether2 priority=0x50
add bridge=bridge interface=ether3 priority=0x40
add bridge=bridge interface=ether4 priority=0x30
add bridge=bridge interface=ether5
```

- Configuration for SW2:

```
/interface bridge
add name=bridge priority=0x2000
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2
add bridge=bridge interface=ether3
```

- Configuration for SW3:

```
/interface bridge
add name=bridge priority=0x3000
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2
add bridge=bridge interface=ether3
```

- Configuration for SW4:

```
/interface bridge
add name=bridge priority=0x4000
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2 path-cost=20
add bridge=bridge interface=ether3
```

In this example, **SW1** is the root bridge since it has the lowest bridge priority. **SW2** and **SW3** have ether1,ether2 connected to the root bridge and ether3 is connected to **SW4**. When all switches are working properly, the traffic will be flowing from ServerA through SW1_ether2, through SW2, through SW4 to ServerB. In the case of **SW1** failure, the **SW2** becomes the root bridge because of the next lowest priority, indicated by the dotted line in the diagram. Below is a list of ports and their role for each switch:

- **root-port** - SW2_ether2, SW3_ether2, SW4_ether1
- **alternate-port** - SW2_ether1, SW3_ether1, SW4_ether2
- **designated-port** - SW1_ether1, SW1_ether2, SW1_ether3, SW1_ether4, SW1_ether5, SW2_ether3, SW2_ether3, SW4_ether3

Note: By the 802.1Q recommendations, you should use bridge priorities in steps of 4096. To set a recommended priority it is more convenient to use hexadecimal notation, for example, 0 is 0x0000, 4096 is 0x1000, 8192 is 0x2000 and so on (0..F).

Multiple Spanning Tree Protocol

Multiple Spanning Tree Protocol (MSTP) is used on a bridge interface to ensure loop-free topology across multiple VLANs, MSTP can also provide Layer2 redundancy and can be used as a load balancing technique for VLANs since it has the ability to have different paths across different VLANs. MSTP is operating very similarly to (R)STP and many concepts from (R)STP can be applied to MSTP and it is highly recommended to understand the principles behind (R)STP before using MSTP, but there are some differences that must be taken into account when designing an MSTP enabled network.

In case (R)STP is used, the BPDUs are sent across all physical interfaces in a bridge to determine loops and stop ports from being able to forward traffic if it causes a loop. In case there is a loop inside a certain VLAN, (R)STP might not be able to detect it. Some STP variants solve this problem by running an STP instance on every single VLAN (PVST), but this has been proven to be inefficient and some STP variants solve this problem by running a single STP instance across all VLANs (CST), but it lacks the possibility to do load balancing for each VLAN or VLAN group. MSTP tends to solve both problems by using MST instances that can define a group of VLANs (VLAN mapping) that can be used for load balancing and redundancy, this means that each VLAN group can have a different root bridge and a different path. Note that it is beneficial to group multiple VLANs in a single instance to reduce the amount of CPU cycles for each network topology change.

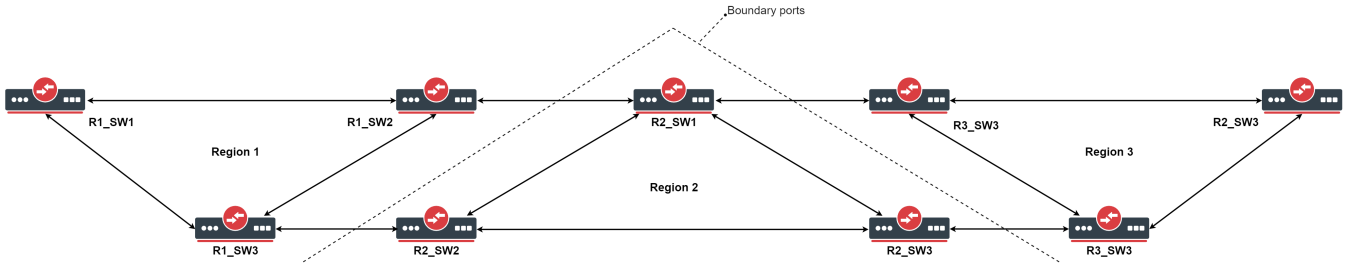
In RouterOS with MSTP enabled the bridge priority is the CIST's root bridge priority, as stated in the IEEE 802.1Q standard the bridge priority must be in steps of 4096, the 12 lowest bits are ignored. These are valid bridge priorities: 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672, 32768, 36864, 40960, 45056, 49152, 53248, 57344, 61440. When setting an invalid bridge priority, RouterOS will warn you about it and trunk the value to a valid value, but will save the original value in the configuration since invalid bridge priority values can still be used in (R)STP between devices running RouterOS, though it is recommended to use valid a bridge priority instead.

MSTP Regions

MSTP works in groups called regions, for each region there will be a regional root bridge and between regions, there will be a root bridge elected. MSTP will use Internal Spanning Tree (IST) to build the network topology inside a region and Common Spanning Tree (CST) outside a region to build the network topology between multiple regions, MSTP combines these two protocols into Common and Internal Spanning Tree (CIST), which holds information about topology inside a region and between regions. From CST's perspective, a region will seemingly be as a single virtual bridge, because of this MSTP is considered very scalable for large networks. In order for bridges to be in the same region, their configuration must match, BPDUs will not include VLAN mappings since they can be large, rather a computed hash is being transmitted. If a bridge receives a BPDU through a port and the configuration does not match, then MSTP will consider that port as a boundary port and that it can be used to reach other regions. Below is a list of parameters that need to match in order for MSTP to consider a BPDU from the same region:

- Region name
- Region revision
- VLAN mappings to MST Instance IDs (computed hash)

It is possible to create MSTP enabled network without regions, though to be able to do load balancing per VLAN group it is required for a bridge to receive a BPDU from a bridge that is connected to it with the same parameters mentioned above. In RouterOS the default region name is empty and region revision is 0, which are valid values, but you must make sure that they match in order to get multiple bridges in a single MSTP region. A region cannot exist if their bridges are scattered over the network, these bridges must be connected at least in one way, in which they can send and receive BPDUs without leaving the region, for example, if a bridge with different region related parameters is between two bridges that have the same region related parameters, then there will exist at least 3 different MSTP regions.



The downside of running every single bridge in a single MSTP region is the excess CPU cycles. In comparison, PVST(+) creates a Spanning Tree Instance for each VLAN ID that exists on the network, since there will be very limited paths that can exist in a network, then this approach creates a lot of overhead and unnecessary CPU cycles, this also means that this approach does not scale very well and can overload switches with not very powerful CPUs. MSTP solves this problem by dividing the network into MSTP regions, where each bridge inside this region will exchange and process information about VLANs that exist inside the same region, but will run a single instance of Spanning Tree Protocol in the background to maintain the network topology between regions. This approach has been proven to be much more effective and much more scalable, this means that regions should be used for larger networks to reduce CPU cycles.

In regions, you can define MST Instances, which are used to configure load balancing per VLAN group and to elect the regional root bridge. It is worth mentioning that in each region there exists a pre-defined MST Instance, in most documentations, this is called as **MSTI0**. This MST Instance is considered as the default MST Instance, there are certain parameters that apply to this special MST Instance. When traffic is passing through an MSTP enabled bridge, MSTP will look for an MST Instance that has a matching VLAN mapping, but if a VLAN mapping does not exist for a certain VLAN ID, then traffic will fall under **MSTI0**.

⚠ Since MSTP requires VLAN filtering on the bridge interface to be enabled, then make sure that you have allowed all required VLAN IDs in `/interface bridge vlan`, otherwise, the traffic will not be forwarded and it might seem as MSTP misconfigured, although this is a VLAN filtering misconfiguration.

Election process

The election process in MSTP can be divided into two sections, intra-region and inter-region. For MSTP to work properly there will always need to be a regional root, that is the root bridge inside a region, and a CIST root, that is the root bridge between regions. A regional root is the root bridge inside a region, regional root bridge will be needed to properly set up load balancing for VLAN groups inside a region. CIST root will be used to configure which ports will be alternate/backups ports (inactive) and which ports will be root ports (active).

⚠ Between regions, there is no load balancing per VLAN group, root port election process and port blocking between MSTP regions is done the same way as in (R)STP. If CIST has blocked a port that is inside an MSTP region to prevent traffic loops between MSTP regions, then this port can still be active for IST to do load balancing per VLAN group inside an MSTP region.

- The following parameters are involved to elect a regional root bridge or root ports inside a MSTP region:

Property	Description
priority (*integer: 0..65535 decimal format or 0x0000-0xffff hex format*; Default: **32768 / 0x8000**)	/interface bridge msti, MST Instance priority, used to elect a regional root inside a MSTP region.
internal-path-cost (*integer: 1..200000000*; Default:)	/interface bridge port, path cost to the regional root for unknown VLAN IDs (MSTI0), used on a root port inside a MSTP region.
priority (*integer: 0..240*; Default: **128**)	/interface bridge port mst-override, MST port priority for a defined MST Instance, used on a bridge port on the regional root bridge.
internal-path-cost (*integer: 1..200000000*; Default:)	/interface bridge port mst-override, MST port path cost for a defined MST Instance, used on a non-root bridge port inside a MSTP region.

- The following parameters are involved to elect a CIST root bridge or CIST root ports:

Property	Description
priority (*integer: 0..65535 decimal format or 0x0000-0xffff hex format*; Default: **32768 / 0x8000**)	/interface bridge, CIST bridge priority, used to elect a CIST root bridge.
priority (*integer: 0..240*; Default: **128**)	/interface bridge port, CIST port priority, used on a CIST root bridge to elect CIST root ports.
path-cost (*integer: 1..200000000*; Default:)	/interface bridge port, CIST port path cost, used on a CIST non-root bridge port to elect CIST root ports.

⚠ The sequence of parameters in which MSTP checks to elect root bridge/ports are the same as in (R)STP, you can read more about it at the (R)STP Election Process section.

MST Instance

Sub-menu: `/interface bridge msti`

This section is used to group multiple VLAN IDs to a single instance to create a different root bridge for each VLAN group inside an MSTP region.

Property	Description
bridge (*text*; Default:)	Bridge to which assign an MST instance.
identifier (*integer: 1..31*; Default:)	MST instance identifier.
priority (*integer: 0..65535 decimal format or 0x0000-0xffff hex format*; Default: **32768 / 0x8000**)	MST instance priority, used to determine the root bridge for a group of VLANs in an MSTP region.
vlan-mapping (*integer: 1..4094*; Default:)	The list of VLAN IDs to assign to MST instance. This setting accepts the VLAN ID range, as well as comma, separated values. E.g. `vlan-mapping=100-115,120,122,128-130`

MST Override

Sub-menu: `/interface bridge port mst-override`

This section is used to select the desired path for each VLAN mapping inside an MSTP region.

Property	Description
disabled (*yes* | *no*; Default: **no**)	Whether entry is disabled.
internal-path-cost (*integer: 1..200000000*; Default:)	Path cost for an MST instance's VLAN mapping, used on VLANs that are facing towards the root bridge to manipulate path selection, lower path cost is preferred.
identifier (*integer: 1..31*; Default:)	MST instance identifier.

priority (*integer: 0..240*; Default: **128**)	The priority an MST instance's VLAN, used on VLANs that are facing away from the root bridge to manipulate path selection, lower priority is preferred.
interface (*name*; Default:)	Name of the port on which use configured MST instance's VLAN mappings and defined path cost and priority.

Monitoring

Similarly to (R)STP, it is also possible to monitor MSTP status. By monitoring the bridge interface itself it possible to see the current CIST root bridge and the current regional root bridge for MSTI0, it is also possible to see the computed hash of MST Instance identifiers and VLAN mappings, this is useful when making sure that certain bridges are in the same MSTP region. Below you can find an example to monitoring an MSTP bridge:

```
/interface bridge monitor bridge
                    state: enabled
     current-mac-address: 6C:3B:6B:7B:F0:AA
             root-bridge: no
          root-bridge-id: 0x1000.64:D1:54:24:23:72
  regional-root-bridge-id: 0x4000.6C:3B:6B:7B:F0:AA
          root-path-cost: 10
               root-port: ether4
              port-count: 5
   designated-port-count: 3
       mst-config-digest: 74edbeefdbf82cf63a70cf60e43a56f3
```

In MSTP it is possible to monitor the MST Instance, this is useful to determine the current regional root bridge for a certain MST Instance and VLAN group, below you can find an example to monitor an MST Instance:

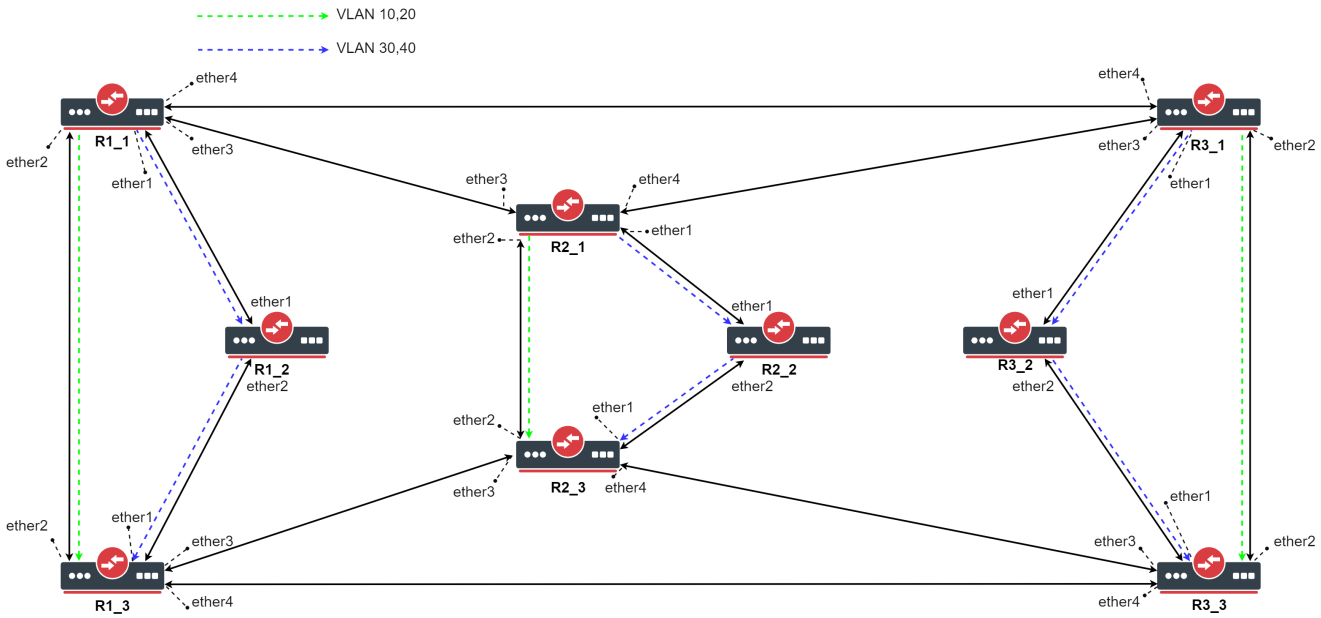
```
/interface bridge msti monitor 1
                    state: enabled
               identifier: 2
     current-mac-address: 6C:3B:6B:7B:F0:AA
             root-bridge: no
          root-bridge-id: 0.00:00:00:00:00:00
  regional-root-bridge-id: 0x1002.6C:3B:6B:7B:F9:08
          root-path-cost: 0
               root-port: ether2
              port-count: 5
   designated-port-count: 1
```

It is also possible to monitor a certain MST Override entry, this is useful to determine the port role for a certain MST Instance when configuring root ports and alternate/backup ports in an MSTP region, below you can find an example to monitor an MST Override entry:

```
/interface bridge port mst-override monitor 1
                     port: ether3
                   status: active
               identifier: 2
                     role: alternate-port
                 learning: no
               forwarding: no
   internal-root-path-cost: 15
         designated-bridge: 0x1002.6C:3B:6B:7B:F9:08
  designated-internal-cost: 0
    designated-port-number: 130
```

MSTP example

Let's say that we need to design topology and configure MSTP in a way that VLAN 10,20 will be forwarded in one path, but VLAN 30,40 will be forwarded in a different path, while all other VLAN IDs will be forwarded in one of those paths. This can easily be done by setting up MST Instances and assigning port path costs, below you can find a network topology that needs to do load balancing per VLAN group with 3 separate regions as an example:



The topology of an MSTP enabled network with load balancing per VLAN group

Start by adding each interface to a bridge, initially, you should create a (R)STP bridge without VLAN filtering enabled, this is to prevent losing access to the CPU. Each device in this example is named by the region that it is in (Rx) and a device number (_x). For larger networks configuring MSTP can be confusing because of the number of links and devices, we recommend using The Dude to monitor and design a network topology.

- Use the following commands on **R1_1**, **R1_3**, **R2_1**, **R2_3**, **R3_1**, **R3_3**:

```
/interface bridge
add name=bridge protocol-mode=rstp vlan-filtering=no
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2
add bridge=bridge interface=ether3
add bridge=bridge interface=ether4
```

- Use the following commands on **R1_2**, **R2_2**, **R3_2**:

```
/interface bridge
add name=bridge protocol-mode=rstp vlan-filtering=no
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=ether2
```

- Make sure you allow the required VLAN IDs on these devices, here we will consider that each device will receive tagged traffic that needs to be load balanced per VLAN group, use these commands on **R1_1**, **R1_3**, **R2_1**, **R2_3**, **R3_1**, **R3_3**:

```
/interface bridge vlan
add bridge=bridge tagged=ether1,ether2,ether3,ether4 vlan-ids=10,20,30,40
```

- Use the following commands on **R1_2**, **R2_2**, **R3_2**:

```
/interface bridge vlan
add bridge=bridge tagged=ether1,ether2 vlan-ids=10,20,30,40
```

⚠ Make sure you add all the needed VLAN IDs and ports to the bridge VLAN table, otherwise your device will not forward all required VLANs and /or you will lose access to the device.

We need to assign a region name for each bridge that we want to be in a single MSTP region, you can also specify the region revision, but it is optional, though they need to match. In this example, if all bridges will have the same region name, then they will all be in a single MSTP bridge. In this case, we want to separate a group of 3 bridges in a different MSTP region to do load balancing per VLAN group and to create diversity and scalability.

- Set appropriate region name (and region revision) for each bridge, use the following commands on each device (**change the region name!**):

```
/interface bridge
set bridge region-name=Rx region-revision=1
```

After we have created 3 different MSTP regions, we need to decide which device is going to be a regional root for each VLAN group. For consistency, we are going to set the first device (_1) in each region as the regional root for VLAN 10,20 and the third device (_3) in each region as the regional root for VLAN 30,40. This can be done by creating an MST Instance for each VLAN group and assigning a bridge priority to it. The MST Instance identifier is only relevant inside an MSTP region, outside an MSTP region these identifiers can be different and mapped to a different VLAN group.

- Use the following commands on **R1_1**, **R2_1**, **R3_1**:

```
/interface bridge msti
add bridge=bridge identifier=1 priority=0x1000 vlan-mapping=10,20
add bridge=bridge identifier=2 priority=0x3000 vlan-mapping=30,40
```

- Use the following commands on **R1_3**, **R2_3**, **R3_3**:

```
/interface bridge msti
add bridge=bridge identifier=1 priority=0x3000 vlan-mapping=10,20
add bridge=bridge identifier=2 priority=0x1000 vlan-mapping=30,40
```

- Use the following commands on **R1_2**, **R2_2**, **R3_2**:

```
/interface bridge msti
add bridge=bridge identifier=1 priority=0x2000 vlan-mapping=10,20
add bridge=bridge identifier=2 priority=0x2000 vlan-mapping=30,40
```

Now we need to override the port path-cost and/or port priority for each MST Instance. This can be done by adding a MST-Override entry for each port and each MST Instance. To achieve that for a certain MST Instance the traffic flow path is different, we simply need to make sure that the port path cost and/or priority is larger. We can either increase the port path cost or either decrease the port path cost to ports that are facing towards the regional root bridge. It doesn't matter if you increase or decrease all values, it is important that at the end one port's path cost is larger than the other's.

- Use the following commands on **R1_1**, **R2_1**, **R3_1**:

```
/interface bridge port mst-override
add identifier=2 interface=ether1 internal-path-cost=5
add identifier=2 interface=ether2 internal-path-cost=15
```

- Use the following commands on **R1_2**, **R2_2**, **R3_2**:

```
/interface bridge port mst-override
add identifier=1 interface=ether1 internal-path-cost=5
add identifier=2 interface=ether2 internal-path-cost=9
```

- Use the following commands on **R1_3**, **R2_3**, **R3_3**:

```
/interface bridge port mst-override
add identifier=1 interface=ether2 internal-path-cost=5
add identifier=1 interface=ether3 internal-path-cost=9
```

In this case for VLAN 10,20 to reach the third device from the first device, it would choose between ether1 and ether2, one port will be blocked and set as an alternate port, ether1 will have path cost as `5+9=14` and ether2 will have path cost as `10`, ether2 will be elected as the root port for MSTI1 on the third device. In case for VLAN 30,40 to reach the first device from the third device, ether1 will have path cost as `5+9=14` and ether2 will have path cost as `15`, ether1 will be elected as the root port for MSTI2 on the third device.

Now we can configure the root ports for **MSTI0**, in which will fall under all VLANs that are not assigned to a specific MST Instance, like in our example VLAN 10,20 and VLAN 30,40. To configure this special MST Instance, you will need to specify `internal-path-cost` to a bridge port. This value is only relevant to MSTP regions, it does not have any effect outside an MSTP region. In this example will choose that all unknown VLANs will be forwarded over the same path as VLAN 30,40, we will simply increase the path cost on one of the ports.

- Use the following commands on **R1_3**, **R2_3**, **R3_3**:

```
/interface bridge port
set [find where interface=ether3] internal-path-cost=25
```

At this point, a single region MSTP can be considered as configured and in general, MSTP is fully functional. It is highly recommended to configure the CIST part, but for testing purposes, it can be left with the default values. Before doing any tests, you need to enable MSTP on all bridges.

- Use the following commands on **all** devices:

```
/interface bridge
set bridge protocol-mode=mstp vlan-filtering=yes
```

When MSTP regions have been configured, you can check if they are properly configured by forwarding traffic, for example, send tagged traffic from the first device to the third device and change the VLAN ID for the tagged traffic to observe different paths based on VLAN ID. When this is working as expected, then you can continue to configure CIST related parameters to elect a CIST root bridge and CIST root ports. For consistency we will choose the first device in the first region to be the CIST root bridge and to ensure the consistency in case of failure we can set a higher priority to all other bridges.

- Use the following commands on **R1_1:**

```
/interface bridge
set bridge priority=0x1000
```

- Use the following commands on **R1_2**:

```
/interface bridge
set bridge priority=0x2000
```

- ...
- Use the following commands on **R3_3**:

```
/interface bridge
set bridge priority=0x9000
```

We also need to elect a root port on each bridge, for simplicity we will choose the port that is closest to **R1_1** as the root port and has the least hops. At this point the procedure to elect root ports is the same as the procedure in (R)STP.

- Use the following commands on **R3_3:**

```
/interface bridge port
set [find where interface=ether2] path-cost=30
set [find where interface=ether3] path-cost=40
set [find where interface=ether4] path-cost=20
```

- Use the following commands on **R1_3** and **R2_3:**

```
/interface bridge port
set [find where interface=ether2] path-cost=20
set [find where interface=ether3] path-cost=30
```

- Use the following commands on **R1_2**:

```
/interface bridge port
set [find where interface=ether1] path-cost=30
```

Wireless VLAN Trunk

Summary

A very common task is to forward only a certain set of VLANs over a Wireless Point-to-Point (PtP) link. Since RouterOS v6.41 this can be done using bridge VLAN filtering and should be used instead of any other methods (including bridging VLAN interfaces). Let's say we need to forward over a Wireless link to 2 different VLANs and all other VLAN IDs should be dropped. VLAN 10 is going to be our Internet traffic while VLAN 99 is going to be for our management traffic. Below you can find the network topology:

Configuration

Start by creating a new bridge on **AP** and **ST** and add **ether1** and **wlan1** ports to it:

```
/interface bridge
add name=bridge protocol-mode=none
/interface bridge port
add bridge=bridge interface=ether1
add bridge=bridge interface=wlan1
```

You can enable RSTP if it is required, but generally, RSTP is not required for PtP links since there should not be any way for a loop to occur.

For security reasons you should enable ingress-filtering since you are expecting only tagged traffic, then you can set the bridge to filter out all untagged traffic. Do the following on **AP** and **ST**:

```
/interface bridge port
set [find where interface=ether1 or interface=wlan1] frame-types=admit-only-vlan-tagged ingress-filtering=yes
```

Set up the bridge VLAN table. Since VLAN99 is going to be our management traffic, then we need to allow this VLAN ID to be able to access the bridge interface, otherwise, the traffic will be dropped as soon as you will try to access the device. VLAN10 does not need to access the bridge since it is only meant to be forwarded to the other end. To achieve such functionality add these entries to the bridge VLAN table on **AP** and **ST**:

```
/interface bridge vlan
add bridge=bridge tagged=ether1,wlan1 vlan-ids=10
add bridge=bridge tagged=ether1,wlan1,bridge vlan-ids=99
```

You can limit from which interfaces it will be allowed to access the device. For example, if you don't want the device to be accessible from `wlan1`, then you can remove the interface from the corresponding bridge VLAN entry.

For devices with hardware offloaded VLAN filtering and wireless interface support (e.g. RB4011 with RTL8367 switch chip, or LtAP with MT7621 switch chip), more attention needs to be paid. Packets going from HW offloaded ports to wireless can be filtered, if the VLAN access to the CPU is not allowed. It is possible to allow CPU access for a certain VLAN by adding the bridge interface as a VLAN member (similar to the VLAN99 example) or disabling HW offloading on bridge ports.

All devices (**R1**, **R2**, **AP,** and **ST**) need a VLAN interface created in order to be able to access the device through the specific VLAN ID. For **AP** and **ST** create the VLAN interface on top of the bridge interface and assign an IP address to it:

```
/interface vlan
add interface=bridge name=MGMT vlan-id=99
/ip address
add address=192.168.99.X/24 interface=MGMT
```

For **R1** and **R2** do the same, but the interface, on which you need to create the VLAN interface, will probably change, depending on your setup:

```
/interface vlan
add interface=ether1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.X/24 interface=MGMT
```

To allow more VLANs to be forwarded, you simply need to specify more VLAN IDs in the bridge VLAN table, you can specify multiple VLANs divided by coma or even VLAN ranges.

Setup the Wireless link on **AP**:

```
/interface wireless security-profiles
add authentication-types=wpa2-psk mode=dynamic-keys name=wlan_sec wpa2-pre-shared-key=use_a_long_password_here
/interface wireless
set wlan1 band=5ghz-a/n/ac channel-width=20/40/80mhz-Ceee disabled=no mode=bridge scan-list=5180 security-
profile=wlan_sec ssid=ptp_test
```

Setup the Wireless link on **ST**:

```
/interface wireless security-profiles
add authentication-types=wpa2-psk mode=dynamic-keys name=wlan_sec wpa2-pre-shared-key=use_a_long_password_here
/interface wireless
set wlan1 band=5ghz-a/n/ac channel-width=20/40/80mhz-Ceee disabled=no mode=station-bridge scan-list=5180
security-profile=wlan_sec ssid=ptp_test
```

For each type of setup, there are different requirements, for PtP links NV2 wireless protocol is commonly used. You can read more about NV2 on the NV2 Manual page.

When links are set up, you can enable bridge VLAN filtering on **AP** and **ST**:

```
/interface bridge
set bridge vlan-filtering=yes
```

Double-check the bridge VLAN table before enabling VLAN filtering. Misconfigured bridge VLAN table can lead to the device being inaccessible and a configuration reset might be required.

WMM and VLAN priority

How WMM works

WMM works by dividing traffic into 4 access categories: background, best effort, video, voice. QoS policy (different handling of access categories) is applied on transmitted packets, therefore the transmitting device is treating different packets differently, e.g. AP does not have control over how clients are transmitting packets, and clients do not have control over how AP transmits packets.

Mikrotik AP and client classifies packets based on the priority assigned to them, according to the table (as per WMM specification): 1,2 - background 0,3 - best effort 4,5 - video 6,7 - voice.

To be able to use multiple WMM access categories, not just best effort where all packets with default priority 0 go, priority must be set for those packets. By default, all packets (incoming and locally generated) inside the router have priority 0.

"Better" access category for packet does not necessarily mean that it will be sent over the air before all other packets with the "worse" access category. WMM works by executing the DCF method for medium access with different settings for each access category (EDCF), which basically means that "better" access category has a higher probability of getting access to medium - WMM enabled station can be considered to be 4 stations, one per access category, and the ones with "better" access category use settings that make them more likely to get chance to transmit (by using shorter backoff timeouts) when all are contending for medium. Details can be studied in 802.11e and WMM specifications.

> WMM support can be enabled using the `wmm-support` setting. It only applies to bands B and G. Other bands will have it enabled regardless of this setting

How VLAN priority works

The VLAN priority is a 3-bit field called Priority Code Point (PCP) within a VLAN-tagged header and values are between 0 and 7. It is used for implementing QoS on bridges and switches. MikroTik devices by default are sending VLAN packets (locally generated or encapsulated) with a priority of 0. RouterOS bridge forwards VLAN tagged packets unaltered, which means that received VLAN tagged packets with a certain VLAN priority will leave the bridge with the same VLAN priority. The only exception is when the bridge untags the packet, in this situation VLAN priority is not preserved due to the missing VLAN header.

More details can be studied in the IEEE 802.1p specification.

How to set priority

Priority of packets can be set using `action=set-priority` of IP firewall mangle rules or bridge filter/nat rules. Priority can be set to a specific value or taken from the ingress priority using the `from-ingress` setting. Ingress priority is the priority value that was detected on the incoming packet, if available. Currently, there are 2 sources of ingress priority - priority in the VLAN header and priority from the WMM packet received over a wireless interface. For all other packets ingress priority is 0.

Note that ingress priority value is not automatically copied to IP mangle `priority` value, the correct rule needs to be set up to do this.

There are basically 2 ways to control priority - assign priority with rules with particular matchers (protocol, addresses, etc.) or set it from ingress priority. Both options require setting up correct rules.

This essentially means that if it is not possible or wanted to classify packets by rules, the configuration of the network must be such that the router can extract ingress priority from incoming frames. Remember there are currently 2 sources for this - VLAN tag in packets and received WMM packets.

> Do not mix priority of queues with priority assigned to packets. Priorities of queues work separately and specify the "importance" of the queue and have meaning only within a particular queue setup. Think of packet priority as some kind of mark, that gets attached to the packet by rules. Also take into account that this mark currently is only used for outgoing packets when going over WMM enabled link, and in case VLAN tagged packet is sent out (no matter if that packet is tagged locally or bridged).

Set VLAN or WMM priority based on specific matchers

It is possible to change the VLAN and WMM priorities based on specific matchers in IP mangle or bridge filter/nat rules. In this example, all outgoing ICMP packets will be sent with a VLAN or WMM priority using the IP mangle rule:

```
/ip firewall mangle
add action=set-priority chain=output new-priority=2 protocol=icmp
```

Custom priority mapping

Sometimes certain VLAN or WMM priorities need to be changed or cleared to a default value. We can use the `ingress-priority` matcher in IP mangle or bridge firewall/nat rules to filter only the needed priorities and change them to a different value using the `new-priority` action setting. For example, forwarded VLAN tagged packets over a bridge with a priority of 5, need to be changed to 0.

```
/interface bridge filter
add action=set-priority chain=forward ingress-priority=5 new-priority=0
```

Translating WMM priority to VLAN priority inside a bridge

When a wireless packet is received with an already set WMM priority, the RouterOS bridge does not automatically translate it to a VLAN header. It means, that received wireless packets with WMM priority that gets VLAN tagged by the bridge will be forwarded with a VLAN priority of 0. However, we can use a bridge filter rule with `from-ingress` setting to keep the priority in VLAN packets. For example, we would like to forward wireless packets over ether2 with VLAN 10 header and keep the already set WMM priority (set by wireless client).

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=wlan2 pvid=10
/interface bridge vlan
add bridge=bridge1 tagged=ether2 vlan-ids=10

# translates WMM priority to VLAN priority
/interface bridge filter
add action=set-priority chain=forward new-priority=from-ingress out-interface=ether2
```

The same situation applies when wireless packets are VLAN tagged by the wireless interface using the `vlan-mode=use-tag` and `vlan-id` settings. You still need to use the same bridge filter rule to translate WMM priority to VLAN priority:

```
/interface wireless
set [ find default-name=wlan2 ] vlan-mode=use-tag vlan-id=10

/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=wlan2

 # translates WMM priority to VLAN priority
/interface bridge filter
add action=set-priority chain=forward new-priority=from-ingress out-interface=ether2
```

The same principles apply in the other direction. RouterOS does not automatically translate VLAN priority to WMM priority. The same rule `new-priority=from-ingress` can be used to translate VLAN priority to WMM priority.

RouterOS bridge forwards VLAN tagged packets unaltered, which means that received VLAN tagged packets with a certain VLAN priority will leave the bridge with the same VLAN priority. The only exception is when the bridge untags the packet, in this situation VLAN priority is not preserved due to the missing VLAN header.

Priority from DSCP

Another way of setting VLAN or WMM priority is by using the DSCP field in the IP header, this can only be done by the IP firewall mangle rule with `new-priority=from-dscp` or `new-priority=from-dscp-high-3-bits` settings and `set-priority` action property. Note that DSCP in IP header can have values 0-63, but priority only 0-7. When using the `new-priority=from-dscp` setting, the priority will be 3 low bits of the DSCP value, but when using `new-priority=from-dscp-high-3-bits` the priority will be 3 high bits of DSCP value.

Remember that DSCP can only be accessed on IP packets and DSCP value in IP header should be set somewhere (either by client devices or IP mangle rules).

It is best to set the DSCP value in the IP header of packets on some border router (e.g. main router used for connection to the Internet), based on traffic type e.g. set DSCP value for packets coming from the Internet belonging to SIP connections to 7, and 0 for the rest. This way packets must be marked only in one place. Then all APs on the network can set packet priority from DSCP value with just one rule.

Set VLAN or WMM priority from DSCP

In this example, the AP device will set WMM priority from DSCP when packets are routed through the wireless interface.

```
/ip firewall mangle
add action=set-priority chain=forward new-priority=from-dscp out-interface=wlan2
```

When packets are forwarded through a bridge, to change VLAN/WMM priority from DSCP you have to pass packets through IP firewall mangle rules, to do so set `use-ip-firewall=yes` under the bridge settings.

DSCP from Priority

Similarly, the DSCP value can be set if the received packet contains VLAN or WMM priority. This can be achieved with IP mangle rules with `new-dscp=from-priority` or `new-dscp=from-priority-to-high-3-bits` settings and `change-dscp` action property. Note that priority in VLAN or WMM packets can have values 0-7, but DSCP in IP headers are 0-63. When using the `new-dscp=from-priority` setting, the value of priority will set the 3 low bits of the DSCP, but when using `new-dscp=from-priority-to-high-3-bits` the value of priority will set the 3 high bits of the DSCP.

However, this setting cannot directly use ingress priority from received VLAN or WMM packets. You first need to set priority using IP mangle or bridge filter /nat rules (ingress priority can be used in this case), and only then apply the DSCP rule.

Set DSCP from VLAN or WMM priority

In this example, the AP device needs to set DSCP from WMM priority when packets are routed. First, add a rule to set priority, it will be needed for the DSCP rule to correctly change the DSCP value. This rule can take priority from ingress. Then add the DSCP rule to change its value.

```
/ip firewall mangle
add action=set-priority chain=prerouting in-interface=wlan2 new-priority=from-ingress
add action=change-dscp chain=prerouting in-interface=wlan2 new-dscp=from-priority
```

When packets are forwarded through a bridge, to change DSCP from VLAN/WMM you have to pass packets through IP firewall mangle rules, to do so set `use-ip-firewall=yes` under the bridge settings.

Combining priority setting and handling solutions

Complex networks and different situations can be handled by combining different approaches of carrying priority information to ensure QoS and optimize the use of resources, based on the "building blocks" described above. Several suggestions:

- The fewer number of filter rules in the whole network, the better (faster). Try classifying packets only when necessary, prefer to do that on fast routers as most probably connection tracking will be required.
- Use DSCP to carry priority information in IP packets forwarded in your network, this way you can use it when needed.
- Use VLANs where necessary, as they also carry priority information, make sure Ethernet bridges and switches in the way are not clearing priority information in the VLAN tag.
- Remember that QoS does not improve the throughput of links, it just treats different packets differently, and also that WMM traffic over the wireless link will discriminate regular traffic in the air.

See also

- Packet Flow in RouterOS
- IP mangle
- Bridge firewall

Firewall and Quality of Service

In This Section:

The firewall implements stateful (by utilizing connection tracking) and stateless packet filtering and thereby provides security functions that are used to manage data flow to, from, and through the router. Along with the Network Address Translation (NAT), it serves as a tool for preventing unauthorized access to directly attached networks and the router itself, as well as a filter for outgoing traffic.

Filter

Introduction

Firewall filters are used to allow or block specific packets forwarded to your local network, originated from your router, or destined to the router.

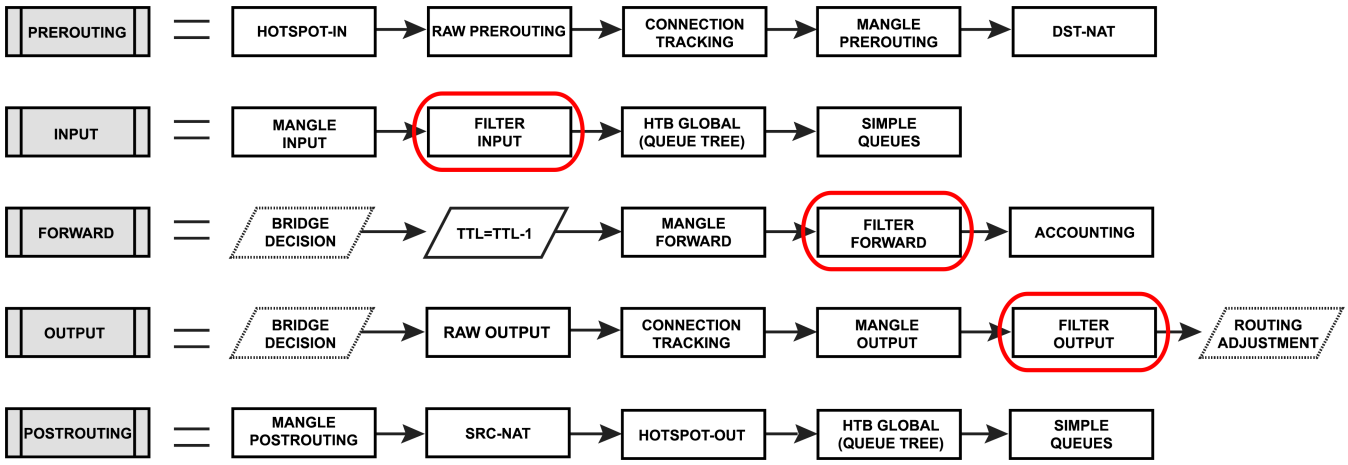
There are two methods on how to set up filtering:

- allow specific traffic and drop everything else
- drop only malicious traffic, everything else is allowed.

Both methods have pros and cons, for example, from a security point of view first method is much more secure, but requires administrator input whenever traffic for new service needs to be accepted. This strategy provides good control over the traffic and reduces the possibility of a breach because of service misconfiguration.

On the other hand, when securing a customer network it would be an administrative nightmare to accept all possible services that users may use. Therefore careful planning of the firewall is essential in advanced setups.

A firewall filter consists of three predefined chains that cannot be deleted:



- **input** - used to process packets entering the router through one of the interfaces with the destination IP address which is one of the router's addresses. Packets passing through the router are not processed against the rules of the input chain
- **forward** - used to process packets passing through the router
- **output** - used to process packets originated from the router and leaving it through one of the interfaces. Packets passing through the router are not processed against the rules of the output chain

When processing a chain, rules are taken from the chain in the order they are listed there from top to bottom. If a packet matches the criteria of the rule, then the specified action is performed on it, and no more rules are processed in that chain (the exception is the passthrough action). If a packet has not matched any rule within the built-in chain, then it is accepted. More detailed packet processing in RouterOS is described in the Packet Flow in RouterOS diagram.

IPv4

Properties

Property	Description
action (*action name*; Default: **accept**)	Action to take if a packet is matched by the rule: • accept - accept the packet. A packet is not passed to the next firewall rule. • add-dst-to-address-list - add destination address to address list specified by `address-list` parameter • add-src-to-address-list - add source address to address list specified by `address-list` parameter • drop - silently drop the packet • fasttrack-connection - process packets from a connection using FastPath by enabling FastTrack for the connection • jump - jump to the user-defined chain specified by the value of `jump-target` parameter • log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar as `passthrough` • passthrough - if a packet is matched by the rule, increase counter and go to next rule (useful for statistics) • reject - drop the packet and send an ICMP reject message; this action allows ICMP reply specification, such as: prohibit or unreachable admin/host/network/port • return - passes control back to the chain from where the jump took place • tarpit - captures and holds TCP connections (replies with SYN/ACK to the inbound TCP SYN packet)
address-list-timeout (*none-dynamic* | *none-static* | *time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till reboot • Value of none-static will leave the address in the address list forever and will be included in configuration export/backup
chain (*name*; Default:)	Specifies to which chain rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule
connection-bytes (*integer-integer*; Default:)	Matches packets only if a given amount of bytes has been transferred through the particular connection. 0 - means infinity, for example `connection-bytes=2000000-0` means that the rule matches if more than 2MB has been transferred through the relevant connection
connection-limit (*integer,netmask*; Default:)	Matches connections per address or address block after a given value is reached. Should be used together with connection-state=new and/or with tcp-flags=syn because matcher is very resource-intensive
connection-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular connection mark. If **no-mark** is set, the rule will match any unmarked connection
connection-nat-state (*srcnat* | *dstnat*; Default:)	Can match connections that are srcnatted, distracted, or both. Note that connection-state=related connections connection-nat-state is determined by the direction of the first packet. and if connection tracking needs to use dst-nat to deliver this connection to the same hosts as the main connection it will be in connection-nat-state=dstnat even if there are no dst-nat rules at all
connection-rate (*Integer 0..4294967295*; Default:)	Connection Rate is a firewall matcher that allows capturing traffic based on the present speed of the connection

connection-state (*established* | *invalid* | *new* | *related* | *untracked*; Default:)	Interprets the connection tracking analytics data for a particular packet: • established - a packet that belongs to an existing connection • invalid - a packet that does not have a determined state in connection tracking (usually - severe out-of-order packets, packets with wrong sequence/ack number, or in case of a resource over usage on the router), for this reason, an invalid packet will not participate in NAT (as only connection-state=new packets do), and will still contain original source IP address when routed. We strongly suggest dropping all *connection-state=invalid* packets in firewall filter forward and input chains • new - the packet has started a new connection, or otherwise associated with a connection that has not seen packets in both directions. • related - a packet that is related to, but not parts of an existing connection, such as ICMP errors or a packet that begins FTP data connection • an untracked - packet which was set to bypass connection tracking in firewall RAW tables.
connection-type (*ftp* | *h323* | *irc* | *pptp* | *quake3* | *sip* | *tftp*; Default:)	Matches packets from related connections based on information from their connection tracking helpers. A relevant connection helper must be enabled under the: /ip firewall service-port
content (*string*; Default:)	Match packets that contain specified text
dscp (*integer: 0..63*; Default:)	Matches DSCP IP header field.
dst-address (*IP/netmask* | *IP range*; Default:)	Matches packets which destination is equal to specified IP or falls into specified IP range.
dst-address-list (*name*; Default:)	Matches destination address of a packet against user-defined address-list.
dst-address-type (*unicast* | *local* | *broadcast* | *multicast*)	Matches destination address type: • unicast - IP address used for point to point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
dst-limit (*integer[/time],integer,dst-address* | *dst-port* | *src-address[/time]*; Default:)	Matches packets until a given rate is exceeded. Rate is defined as packets per time interval. As opposed to the limit matcher, every flow has its own limit. Flow is defined by a mode parameter. Parameters are written in the following format: `rate[/time],burst,mode[/expire]`. • **rate** - packet count per time interval per-flow to match • **time** - specifies the time interval in which the packet count rate per flow cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets per flow to match: this number gets recharged by one every `time/rate`, up to this number • **mode** - this parameter specifies what unique fields define flow (src-address, dst-address, src-and-dst-address, dst-address-and-port, addresses-and-dst-port) • **expire** - specifies interval after which flow with no packets will be allowed to be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges
fragment (*yes|no*; Default:)	Matches fragmented packets. The first (starting) fragment does not count. If connection tracking is enabled there will be no fragments as the system automatically assembles every packet
hotspot (*auth* | *from-client* | *http* | *local-dst* | *to-client*; Default:)	Matches packets received from HotSpot clients against various HotSpot matchers. • auth - matches authenticated HotSpot client packets • from-client - matches packets that are coming from the HotSpot client • http - matches HTTP requests sent to the HotSpot server • local-dst - matches packets that are destined to the HotSpot server • to-client - matches packets that are sent to the HotSpot client
hw-offload (*yes* | *no; Default:* **yes**)	Enables or disables connection offloading using L3HW. It applies specifically to rules with the "action=fasttrack-connection" and is applicable on devices that support Fasttrack offloading.
icmp-options (*integer:integer*; Default:)	Matches ICMP type: code fields

in-bridge-port (*name*; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings.
in-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-bridge-port
in-interface (*name*; Default:)	Interface the packet has entered the router
in-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-interface
ingress-priority (*integer: 0..63*; Default:)	Matches the priority of an ingress packet. Priority may be derived from VLAN, WMM, DSCP, or MPLS EXP bit. read more
ipsec-policy (*in* | *out, ipsec* | *none*; Default:)	Matches the policy used by IPsec. Value is written in the following format: **`direction, policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING, INPUT, and FORWARD chains • out - valid in the POSTROUTING, OUTPUT, and FORWARD chains • ipsec - matches if the packet is subject to IPsec processing; • none - matches packet that is not subject to IPsec processing (for example, IPSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then rule `ipsec-policy=in, ipsec` will match Gre packet, but a rule `ipsec-policy=in,none` will match the ESP packet.
ipv4-options (*any* | *loose-source-routing* | *no-record-route* | *no-router-alert* | *no-source-routing* | *no-timestamp* | *none* | *record-route* | *router-alert* | *strict-source-routing* | *timestamp*; Default:)	Matches IPv4 header options. • any - match packet with at least one of the ipv4 options • loose-source-routing - match packets with a loose source routing option. This option is used to route the internet datagram based on information supplied by the source • no-record-route - match packets with no record route option. This option is used to route the internet datagram based on information supplied by the source • no-router-alert - match packets with no router alter option • no-source-routing - match packets with no source routing option • no-timestamp - match packets with no timestamp option • record-route - match packets with record route option • router-alert - match packets with router alter option • strict-source-routing - match packets with strict source routing option • timestamp - match packets with a timestamp
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
layer7-protocol (*name*; Default:)	Layer7 filter name defined in layer7 protocol menu.
limit (*integer,time,integer*; Default:)	Matches packets up to a limited rate (packet rate or bit rate). A rule using this matcher will match until this limit is reached. Parameters are written in the following format: `rate[/time],burst:mode`. • **rate** - packet or bit count per time interval to match • **time** - specifies the time interval in which the packet or bit rate cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets or bits to match: this number gets recharged every 10ms so burst should be at least 1/100 of a rate per second • **mode** - packet or bit mode
log (*yes* | *no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
nth (*integer,integer*; Default:)	Matches every nth packet: *nth=2,1* rule will match every first packet of 2, hence, 50% of all the traffic that is matched by the rule
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings.

out-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-bridge-port
out-interface (; Default:)	Interface the packet is leaving the router
out-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-interface
packet-mark (*no-mark | string*; Default:)	Matches packets marked via mangle facility with particular packet mark. If **no-mark** is set, the rule will match any unmarked packet.
packet-size (*integer[-integer]:0..65535*; Default:)	Matches packets of specified size or size range in bytes.
per-connection-classifier (*ValuesToHash: Denominator/Remainder*; Default:)	PCC matcher allows dividing traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream. Read more >>
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
priority (*integer: 0..63*; Default:)	Matches the packet's priority after a new priority has been set. Priority may be derived from VLAN, WMM, DSCP, MPLS EXP bit, or from the priority that has been set using the set-priority action. Read more
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
psd (*integer,time,integer,integer*; Default:)	Attempts to detect TCP and UDP scans. Parameters are in the following format `WeightThreshold, DelayThreshold, LowPortWeight, HighPortWeight` • **WeightThreshold** - total weight of the latest TCP/UDP packets with different destination ports coming from the same host to be treated as port scan sequence • **DelayThreshold** - delay for the packets with different destination ports coming from the same host to be treated as possible port scan subsequence • **LowPortWeight** - the weight of the packets with privileged (<1024) destination port • **HighPortWeight** - the weight of the packet with non-privileged destination port
random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability
reject-with (*icmp-admin-prohibited | icmp-net-prohibited | icmp-protocol-unreachable | icmp-host-prohibited | icmp-network-unreachable | tcp-reset | icmp-host-unreachable | icmp-port-unreachable*; Default: icmp-network-unreachable)	Specifies ICMP error to be sent back if the packet is rejected. Applicable if `action=reject`
routing-table (*string*; Default:)	Matches packets which destination address is resolved in specific a routing table.
routing-mark (*string*; Default:)	Matches packets marked by mangle facility with particular routing mark
src-address (*Ip/Netmask, Ip range*; Default:)	Matches packets which source is equal to specified IP or falls into a specified IP range
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast | local | broadcast | multicast | blackhole | prohibit | unreachable ;* Default:)	mote{ta{tableMatches source address type: • unicast - IP address used for point to point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in the subnet • multicast - packet is forwarded to a defined group of devices
src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet

tcp-flags (*ack* / *cwr* / *ece* / *fin* / *psh* / *rst* / *syn* / *urg*; Default:)	Matches specified TCP flags • ack - acknowledging data • cwr - congestion window reduced • ece - ECN-echo flag (explicit congestion notification) • fin - close connection • psh - push function • rst - drop connection • syn - new connection • urg - urgent data
tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat* / *fri* / *thu* / *wed* / *tue* / *mon* / *sun*; Default:)	Allows to create a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
tls-host (*string*; Default:)	Allows matching HTTPS traffic based on TLS SNI hostname. Accepts GLOB syntax for wildcard matching. Note that the matcher will not be able to match hostname if the TLS handshake frame is fragmented into multiple TCP segments (packets). Watch our video about this value.
ttl (*integer: 0..255*; Default:)	Matches packets TTL value

Stats

To show additional *read-only* properties:

```
/ip firewall filter print stats
```

Property	Description
bytes (*integer*)	The total amount of bytes matched by the rule
packets (*integer*)	The total amount of packets matched by the rule

```
[admin@MikroTik] > ip firewall filter print stats
Flags: X - disabled, I - invalid, D - dynamic
 #    CHAIN
           ACTION                              BYTES         PACKETS
 0  D ;;; special dummy rule to show fasttrack counters
      forward
           passthrough              50 507 925 242      50 048 246
 1    ;;; defconf: drop invalid
      forward
           drop                            432 270           9 719
 2    ;;; defconf: drop invalid
      input
           drop                            125 943           2 434
 3    input
           accept                   20 090 211 549      20 009 864
 4    ;;; defconf: accept ICMP
      input
           accept                          634 926           7 648
 5    ;;; defconf: drop all not coming from LAN
      input
           drop                          4 288 079          83 428
 6    ;;; defconf: accept in ipsec policy
      forward
           accept                                0               0
7    ;;; defconf: accept out ipsec policy
      forward
           accept                                0               0
8    ;;; defconf: fasttrack
      forward
           fasttrack-connection     28 505 528 775      31 504 682
9    ;;; defconf: accept established,related, untracked
      forward
           accept                   28 505 528 775      31 504 682
10    ;;; defconf: drop all from WAN not DSTNATed
      forward
           drop                                  0               0
```

IPv6

```
/ipv6/firewall/filter
```

Properties

Property	Description
action (*accept* | *add-dst-to-address-list* | ...; Default: **accept**)	Action to take if a packet is matched by the rule: • accept - Accept the packet. It is not passed to the next firewall rule. • add-dst-to-address-list - Add the destination address to address list specified by `address-list` parameter • add-src-to-address-list - Add source address to address list specified by `address-list` parameter • drop -Silently drop the packet. • jump - Jump to the user-defined chain specified by the value of `jump-target` a parameter • log - Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar to `passthrough` • passthrough - Ignore this rule and go to the next one (useful for statistics). • reject - Drop the packet and send an ICMP reject message • return - Passes control back to the chain from where the jump took place.

address-list-timeout (*none-dynamic* / *none-static* / *time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till reboot • Value of none-static will leave the address in the address list forever and will be included in configuration export/backup
chain (*name*; Default:)	Specifies to which chain rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule
connection-bytes (*integer-integer*; Default:)	Matches packets only if a given amount of bytes has been transferred through the particular connection. 0 - means infinity, for example `connection-bytes=2000000-0` means that the rule matches if more than 2MB has been transferred through the relevant connection
connection-limit (*integer,netmask*; Default:)	Matches connections per address or address block after a given value is reached. Should be used together with connection-state=new and/or with tcp-flags=syn because matcher is very resource-intensive
connection-mark (*no-mark* / *string*; Default:)	Matches packets marked via mangle facility with particular connection mark. If **no-mark** is set, the rule will match any unmarked connection
connection-rate (*Integer 0..4294967295*; Default:)	Connection Rate is a firewall matcher that allows capturing traffic based on the present speed of the connection
connection-state (*established* / *invalid* / *new* / *related* / *untracked*; Default:)	Interprets the connection tracking analytics data for a particular packet: • established - a packet that belongs to an existing connection • invalid - a packet that does not have a determined state in connection tracking (usually - severe out-of-order packets, packets with wrong sequence/ack number, or in case of a resource over usage on the router), for this reason, an invalid packet will not participate in NAT (as only connection-state=new packets do), and will still contain original source IP address when routed. We strongly suggest dropping all *connection-state=invalid* packets in firewall filter forward and input chains • new - the packet has started a new connection, or is otherwise associated with a connection that has not seen packets in both directions. • related - a packet that is related to, but not parts of an existing connection, such as ICMP errors or a packet that begins FTP data connection • an untracked - packet which was set to bypass connection tracking in firewall RAW tables.
connection-type (*ftp* / *h323* / *irc* / *pptp* / *quake3* / *sip* / *tftp*; Default:)	Matches packets from related connections based on information from their connection tracking helpers. A relevant connection helper must be enabled under the: /ip firewall service-port
content (*string*; Default:)	Match packets that contain specified text
dscp (*integer: 0..63*; Default:)	Matches DSCP IP header field.
dst-address (*IP/netmask* / *IP range*; Default:)	Matches packets whose destination is equal to specified IP or falls into a specified IP range.
dst-address-list (*name*; Default:)	Matches destination address of a packet against user-defined address list.
dst-address-type (*unicast* / *local* / *broadcast* / *multicast*; Default:)	Matches destination address type: • unicast - IP address used for point-to-point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices

dst-limit (*integer[/time],integer,dst-address | dst-port | src-address[/time]*; Default:)	Matches packets until a given rate is exceeded. Rate is defined as packets per time interval. As opposed to the li mit matcher, every flow has its own limit. Flow is defined by a mode parameter. Parameters are written in the following format: `rate[/time],burst,mode[/expire]`. • **rate** - packet count per time interval per-flow to match • **time** - specifies the time interval in which the packet count rate per flow cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets per flow to match: this number gets recharged by one every `time`/`rate`, up to this number • **mode** - this parameter specifies what unique fields define flow (src-address, dst-address, src-and-dst-address, dst-address-and-port, addresses-and-dst-port) • **expire** - specifies interval after which flow with no packets will be allowed to be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges
icmp-options (*integer:integer*; Default:)	Matches ICMP type: code fields
in-bridge-port (*name*; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings.
in-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-bridge-port
in-interface (*name*; Default:)	Interface the packet has entered the router
in-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-interface
ingress-priority (*integer: 0..63*; Default:)	Matches the priority of an ingress packet. Priority may be derived from VLAN, WMM, DSCP, or MPLS EXP bit.
ipsec-policy (*in | out, ipsec | none*; Default:)	Matches the policy used by IPsec. Value is written in the following format: **`direction, policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING, INPUT, and FORWARD chains • out - valid in the POSTROUTING, OUTPUT, and FORWARD chains • ipsec - matches if the packet is subject to IPsec processing; • none - matches packet that is not subject to IPsec processing (for example, IPSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then rule `ipsec-policy=in,ipsec` will match Gre packet, but a rule `ipsec-policy=in,none` will match the ESP packet.
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
limit (*integer,time,integer*; Default:)	Matches packets up to a limited rate (packet rate or bit rate). A rule using this matcher will match until this limit is reached. Parameters are written in the following format: `rate[/time],burst:mode`. • **rate** - packet or bit count per time interval to match • **time** - specifies the time interval in which the packet or bit rate cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets or bits to match: this number gets recharged every 10ms so burst should be at least 1/100 of a rate per second • **mode** - packet or bit mode
log (*yes | no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
nth (*integer,integer*; Default:)	Matches every nth packet: *nth=2,1* rule will match every first packet of 2, hence, 50% of all the traffic that is matched by the rule
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings.
out-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-bridge-port

out-interface (; Default:)	Interface the packet is leaving the router
out-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-interface
packet-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular packet mark. If **no-mark** is set, the rule will match any unmarked packet.
packet-size (*integer[-integer]:0.. 65535*; Default:)	Matches packets of specified size or size range in bytes.
per-connection-classifier (*ValuesTo Hash:Denominator/Remainder*; Default:)	PCC matcher allows dividing traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream.
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
priority (*integer: 0..63*; Default:)	Matches the packet's priority after a new priority has been set. Priority may be derived from VLAN, WMM, DSCP, MPLS EXP bit, or from the priority that has been set using the set-priority action.
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability
reject-with (*icmp-address-unreachable* | *icmp-admin-prohibited* | *icmp-no-route* | *icmp-not-neighbour* | *icmp-port-unreachable* | *tcp-reset* ; Default: icmp-no-route)	Specifies ICMP error to be sent back if the packet is rejected. Applicable if `action=reject` • icmp-address-unreachable: sends ICMP address unreachable message • icmp-admin-prohibited: sends ICMP address prohibited message • icmp-no-route: sends ICMP address no-route message • icmp-not-neighbour: sends ICMP address not-member message • icmp-port-unreachable: sends ICMP port unreachable message • tcp-reset: sends ICMP resetting a TCP connection
routing-mark (*string*; Default:)	Matches packets marked by mangle facility with particular routing mark
src-address (*Ip/Netmask, Ip range*; Default:)	Matches packets whose source is equal to specified IP or falls into a specified IP range
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast* | *local* | *broadcast* | *multicast*; Default:)	Matches source address type: • unicast - IP address used for point-to-point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in the subnet • multicast - packet is forwarded to a defined group of devices
src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet
tcp-flags (*ack* | *cwr* | *ece* | *fin* | *psh* | *rst* | *syn* | *urg*; Default:)	Matches specified TCP flags • ack - acknowledging data • cwr - congestion window reduced • ece - ECN-echo flag (explicit congestion notification) • fin - close connection • psh - push function • rst - drop connection • syn - new connection • urg - urgent data

tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat | fri | thu | wed | tue | mon | sun*; Default:)	Allows to create a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
tls-host (*string*; Default:)	Allows matching HTTPS traffic based on TLS SNI hostname. Accepts GLOB syntax for wildcard matching. Note that the matcher will not be able to match the hostname if the TLS handshake frame is fragmented into multiple TCP segments (packets).

Stats

```
/ipv6/firewall/filter/print/stats
```

To show additional *read-only* properties:

Property	Description
bytes (*integer*)	The total amount of bytes matched by the rule
packets (*integer*)	The total amount of packets matched by the rule

Example

An example on how to properly secure your device you can find in our Building Your First Firewall article!

Kid Control

Summary

Sub-menu: `/ip kid-control`

"Kid control" is a parental control feature to limit internet connectivity for LAN devices.

Property Description

In this menu it is possible to create a profile for each Kid and restrict internet accessibility.

Property	Description
name (*string*)	Name of the Kids profile
mon,tue,wed,thu,fri,sat,sun (*time*)	Each day of week. Time of day is selected, when internet access should be allowed
disabled (*yes* | *no*)	Whether restrictions is enabled
rate-limit (*string*)	Maximum available data rate for flow
tur-mon,tur-tue,tur-wed,tur-thu,tur-fri,tur-sat,tur-sun (*time*)	Time unlimited rate. Time of day is selected, when internet access should be unlimited

Time unlimited rate parameters have higher priority than rate-limit parameter.

Devices

Sub-menu: `/ip kid-control device`

This sub-menu contains information if there is multiple connected devices to internet (phone, tablet, gaming console, tv etc.). Device is identified by MAC address that is retrieved from the ARP table. The appropriate IP address is taken from there.

Property	Description
name (*string*)	Name of the device
mac-address (*string*)	Devices mac-address
user (*string*)	To which profile append the device
reset-counters (*[id, name]*)	Reset bytes-up and bytes-down counters.

Application example

With following example we will restrict access for Peters mobile phone:

- Disabled internet access on Monday,Wednesday and Friday
- Allowed unlimited internet access on:
  - Tuesday
  - Thursday from 11:00-22:00
  - Sunday 15:00-22:00
- Limited bandwidth to 3Mbps for Peters mobile phone on Saturday from 18:30-21:00

```
[admin@MikroTik] > /ip kid-control add name=Peter mon="" tur-tue="00:00-24h" wed="" tur-thu="11:00-22:00"
fri="" sat="18:30-22:00" tur-sun="15h-21h" rate-limit=3M
[admin@MikroTik] > /ip kid-control device add name=Mobile-phone user=Peter mac-address=FF:FF:FF:ED:83:63
```

Internet access limitation is implemented by adding dynamic firewall filter rules or simple queue rules. Here are example firewall filter rules:

```
[admin@MikroTik] > /ip firewall filter print

1  D ;;; Mobile-phone, kid-control
      chain=forward action=reject src-address=192.168.88.254

2  D ;;; Mobile-phone, kid-control
      chain=forward action=reject dst-address=192.168.88.254
```

Dynamically created simple queue:

```
[admin@MikroTik] > /queue simple print
Flags: X - disabled, I - invalid, D - dynamic

 1  D ;;; Mobile-phone, kid-control
      name="queue1" target=192.168.88.254/32 parent=none packet-marks="" priority=8/8 queue=default-small
/default-small limit-at=3M/3M max-limit=3M/3M burst-limit=0/0
      burst-threshold=0/0 burst-time=0s/0s bucket-size=0.1/0.1
```

It is possible to monitor how much data use specific device:

```
[admin@MikroTik] > /ip kid-control device print stats

Flags: X - disabled, D - dynamic, B - blocked, L - limited, I - inactive
 #
NAME
IDLE-TIME    RATE-DOWN   RATE-UP   BYTES-DOWN     BYTES-UP
 1 BI Mobile-
phone
30s         0bps      0bps    3438.1KiB       8.9KiB
```

It is also possible to **pause** Internet access for the created kids, it will restrict all access until **resume** is used, which will continue with configured settings:

```
[admin@MikroTik] > /ip kid-control pause Peter
[admin@MikroTik] > /ip kid-control print
Flags: X - disabled, P - paused, B - blocked, L - rate-limited
 #
NAME
SUN      MON      TUE      WED      THU      FRI      SAT
 0 PB
Peter
15h-21h                           11h-22h          18:30h-22h
```

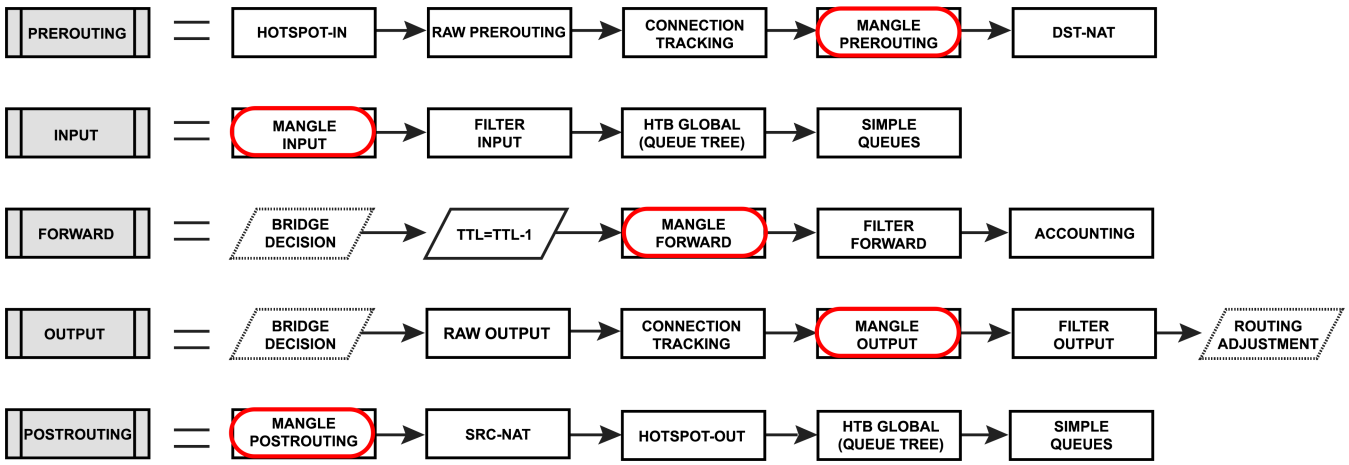
Mangle

Introduction

Mangle is a kind of 'marker' that marks packets for future processing with special marks. Many other facilities in RouterOS make use of these marks, e.g. queue trees, NAT, routing. They identify a packet based on its mark and process it accordingly. The mangle marks exist only within the router, they are not transmitted across the network.

Additionally, the mangle facility is used to modify some fields in the IP header, like TOS (DSCP) and TTL fields.

Firewall mangle rules consist of five predefined chains that cannot be deleted:



- The **PREROUTING** chain: Rules in this chain apply to packets as they just arrive on the network interface;
- The **INPUT** chain: Rules in this chain apply to packets just before they're given to a local process;
- The **OUTPUT** chain: The rules here apply to packets just after they've been produced by a process;
- The **FORWARD** chain: The rules here apply to any packets that are routed through the current host;
- The **POSTROUTING** chain: The rules in this chain apply to packets as they just leave the network interface;

Properties

Property	Description

action (*action name*; Default: **accept**)	Action to take if a packet is matched by the rule: • accept - accept the packet. A packet is not passed to the next firewall rule. • add-dst-to-address-list - add destination address to address list specified by `address-list` parameter • add-src-to-address-list - add source address to address list specified by `address-list` parameter • change-dscp - change Differentiated Services Code Point (DSCP) field value specified by the new-dscp parameter • change-mss - change Maximum Segment Size field value of the packet to a value specified by the new-mss parameter • change-ttl - change Time to Live field value of the packet to a value specified by the new-ttl parameter • clear-df - clear 'Do Not Fragment' Flag • fasttrack-connection - shows fasttrack counters, useful for statistics • jump - jump to the user-defined chain specified by the value of `jump-target` parameter • log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar as `passthrough` • mark-connection - place a mark specified by the new-connection-mark parameter on the entire connection that matches the rule • mark-packet - place a mark specified by the new-packet-mark parameter on a packet that matches the rule • mark-routing - place a mark specified by the new-routing-mark parameter on a packet. This kind of marks is used for policy routing purposes only. Do not apply any other routing marks besides "main" for the packets processed by FastTrack, since FastTrack can only work in the main routing table. • passthrough - if a packet is matched by the rule, increase counter and go to next rule (useful for statistics). • return - pass control back to the chain from where the jump took place • route - forces packets to a specific gateway IP by ignoring normal routing decision (prerouting chain only) • set-priority - set priority specified by the new-priority parameter on the packets sent out through a link that is capable of transporting priority (VLAN or WMM-enabled wireless interface). Read more • sniff-pc - send a packet to a remote RouterOS CALEA server. • sniff-tzsp - send a packet to a remote TZSP compatible system (such as Wireshark). Set remote target with `sniff-target` and `sniff-target-port` parameters (Wireshark recommends port 37008) • strip-ipv4-options - strip IPv4 option fields from IP header, the action does not actually remove IPv4 options but rather replaces all option octets with NOP, further matcher with ipv4-options=any will still match the packet.
address-list (*string*; Default:)	Name of the address list to be used. Applicable if action is `add-dst-to-address-list` or `add-src-to-address-list`
address-list-timeout (*none-dynamic* | *none-static* | *time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till reboot • Value of none-static will leave the address in the address list forever and will be included in configuration export/backup
chain (*name*; Default:)	Specifies to which chain the rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule.
connection-bytes (*integer-integer*; Default:)	Matches packets only if a given amount of bytes has been transferred through the particular connection. 0 - means infinity, for example `connection-bytes=2000000-0` means that the rule matches if more than 2MB (upload and download) has been transferred through the relevant connection
connection-limit (*integer,netmask*; Default:)	Matches connections per address or address block after a given value is reached
connection-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular connection mark. If **no-mark** is set, the rule will match any unmarked connection.

connection-nat-state (*srcnat* | *dstnat*; Default:)	Can match connections that are srcnatted, dstnatted, or both. Note that connection-state=related connections connection-nat-state is determined by the direction of the first packet. and if connection tracking needs to use dst-nat to deliver this connection to the same hosts as the main connection it will be in connection-nat-state=dstnat even if there are no dst-nat rules at all.
connection-rate (*Integer 0..4294967295* ; Default:)	Connection Rate is a firewall matcher that allows the capture of traffic based on the present speed of the connection.
connection-state (*established* | *invalid* | *new* | *related*; Default:)	Interprets the connection tracking analytics data for a particular packet: • established - a packet that belongs to an existing connection • invalid - a packet that does not have a determined state in connection tracking (usually - severe out-of-order packets, packets with wrong sequence/ack number, or in case of a resource over usage on a router), for this reason, invalid packet will not participate in NAT (as only connection-state=new packets do), and will still contain original source IP address when routed. We strongly suggest dropping all connection-state=invalid packets in firewall filter forward and input chains • new - the packet has started a new connection, or otherwise associated with a connection that has not seen packets in both directions • related - a packet that is related to, but not parts of an existing connection, such as ICMP errors or a packet that begins FTP data connection • untracked - packet which was set to bypass connection tracking in Firewall RAW tables.
connection-type (*ftp* | *h323* | *irc* | *pptp* | *quake3* | *sip* | *tftp*; Default:)	Matches packets from related connections based on information from their connection tracking helpers. A relevant connection helper must be enabled under: /ip firewall service-port
content (*string*; Default:)	Match packets that contain specified text
dscp (*integer: 0..63*; Default:)	Matches DSCP IP header field
dst-address (*IP/netmask* | *IP range*; Default:)	Matches packets where the destination is equal to specified IP or falls into a specified IP range
dst-address-list (*name*; Default:)	Matches destination address of a packet against user-defined address list
dst-address-type (*unicast* | *local* | *broadcast* | *multicast*; Default:)	Matches destination address type: • unicast - IP address used for point to point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
dst-limit (*integer[/time],integer,dst-address* | *dst-port* | *src-address[/time]*; Default:)	Matches packets until a given PPS limit is exceeded. As opposed to the limit matcher, every destination IP address/destination port has its own limit. Parameters are written in the following format: `count[/time],burst,mode[/expire]`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional) • **burst** - number of packets that are not counted by packet rate • **mode** - the classifier for packet rate limiting • **expire** - specifies interval after which recorded ip address /port will be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges
fragment (*yes*|*no*; Default:)	Matches fragmented packets. The first (starting) fragment does not count. If connection tracking is enabled there will be no fragments as the system automatically assembles every packet
hotspot (*auth* | *from-client* | *http* | *local-dst* | *to-client*; Default:)	Matches packets received from HotSpot clients against various HotSpot matches. • auth - matches authenticated HotSpot client packets • from-client - matches packets that are coming from the HotSpot client • http - matches HTTP requests sent to the HotSpot server • local-dst - matches packets that are destined to the HotSpot server • to-client - matches packets that are sent to the HotSpot client

icmp-options (*integer:integer*; Default:)	Matches ICMP "type:code" fields
in-bridge-port (*name*; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge
in-interface (*name*; Default:)	Interface the packet has entered the router
ingress-priority (*integer: 0..63*; Default:)	Matches ingress the priority of the packet. Priority may be derived from VLAN, WMM, or MPLS EXP bit. `Read more`
ipsec-policy (*in | out, ipsec | none*; Default:)	Matches the policy used by IpSec. Value is written in the following format: **`direction, policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING, INPUT, and FORWARD chains • out - valid in the POSTROUTING, OUTPUT, and FORWARD chains • ipsec - matches if the packet is subject to IpSec processing; • none - matches packet that is not subject to IpSec processing (for example, IpSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then rule `ipsec-policy=in, ipsec` will match Gre packet, but a rule `ipsec-policy=in,none` will match the ESP packet.
ipv4-options (*any | loose-source-routing | no-record-route | no-router-alert | no-source-routing | no-timestamp | none | record-route | router-alert | strict-source-routing | timestamp*; Default:)	Matches IPv4 header options. • any - match packet with at least one of the ipv4 options • loose-source-routing - match packets with a loose source routing option. This option is used to route the internet datagram based on information supplied by the source • no-record-route - match packets with no record route option. This option is used to route the internet datagram based on information supplied by the source • no-router-alert - match packets with no router alter option • no-source-routing - match packets with no source routing option • no-timestamp - match packets with no timestamp option • record-route - match packets with record route option • router-alert - match packets with router alter option • strict-source-routing - match packets with strict source routing option • timestamp - match packets with a timestamp
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
layer7-protocol (*name*; Default:)	Layer7 filter name defined in layer7 protocol menu.
limit (*integer,time,integer*; Default:)	Matches packets until a given PPS limit is exceeded. Parameters are written in the following format: `count[/time],burst`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional, 1s will be used if not specified) • **burst** - number of packets that are not counted by packet rate
log (*yes | no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
new-dscp (*integer: 0..63*; Default:)	Sets a new DSCP value for a packet
new-mss (*integer*; Default:)	Sets a new MSS for a packet. `Clamp-to-pmtu feature sets (DF) bit in the IP header to dynamically discover the PMTU of a path. Host sends all datagrams on that path with the DF bit set until receives ICMP Destination Unreachable messages with a code meaning "fragmentation needed and DF set". Upon receipt of such a message, the source host reduces its assumed PMTU for the path.`

new-packet-mark (*string*; Default:)	Sets a new packet-mark value
new-priority (*integer | from-dscp | from-dscp-high-3-bits | from-ingress*; Default:)	Sets a new priority for a packet. This can be the VLAN, WMM, DSCP or MPLS EXP priority Read more. This property can also be used to set an internal priority.
new-routing-mark (*string*; Default:)	Sets a new routing-mark value (in RouterOS v7 routing mark must be created before as a new Routing table)
new-ttl (*decrement | increment | set: integer*; Default:)	Sets a new Time to live value
nth (*integer,integer*; Default:)	Matches every nth packet: *nth=2,1* rule will match every first packet of 2, hence, 50% of all the traffic that is matched by the rule
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge
out-interface (; Default:)	Interface the packet is leaving the router
packet-mark (*no-mark | string*; Default:)	Matches packets marked via mangle facility with particular packet mark. If **no-mark** is set, the rule will match any unmarked packet
packet-size (*integer[-integer]:0..65535*; Default:)	Matches packets of specified size or size range in bytes
passthrough (*yes|no*; Default: **yes**)	whether to let the packet to pass further (like action passthrough) into the firewall or not (property only valid some actions)
per-connection-classifier (*ValuesToHash:Denominator/Remainder*; Default:)	PCC matcher allows division of traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
psd (*integer,time,integer,integer*; Default:)	Attempts to detect TCP and UDP scans. Parameters are in the following format `WeightThreshold, DelayThreshold, LowPortWeight, HighPortWeight` • **WeightThreshold** - total weight of the latest TCP/UDP packets with different destination ports coming from the same host to be treated as port scan sequence • **DelayThreshold** - delay for the packets with different destination ports coming from the same host to be treated as possible port scan subsequence • **LowPortWeight** - the weight of the packets with privileged (<1024) destination port • **HighPortWeight** - the weight of the packet with a non-privileged destination port
random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability.
routing-mark (*string*; Default:)	Matches packets marked by mangle facility with particular routing mark
route-dst (*IP, Default:*)	Matches packets with a specific gateway
priority (*integer: 0..63*; Default:)	Matches the packet's priority after a new priority has been set. Priority may be derived from VLAN, WMM, DSCP, MPLS EXP bit, or from the internal priority that has been set using the set-priority action
src-address (*IP/Netmask, IP range*; Default:)	Matches packets where the source is equal to specified IP or falls into a specified IP range.
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast | local | broadcast | multicast*; Default:)	Matches source address type: • unicast - IP address used for point-to-point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices

src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP.
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet
tcp-flags (*ack* / *cwr* / *ece* / *fin* / *psh* / *rst* / *syn* / *urg*; Default:)	Matches specified TCP flags - ack - acknowledging data - cwr - congestion window reduced - ece - ECN-echo flag (explicit congestion notification) - fin - close connection - psh - push function - rst - drop connection - syn - new connection - urg - urgent data
tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat* / *fri* / *thu* / *wed* / *tue* / *mon* / *sun*; Default:)	Allows creation of a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
tls-host (*string*; Default:)	Allows matching traffic based on TLS hostname. Accepts GLOB syntax for wildcard matching. Note that the matcher will not be able to match the hostname if the TLS handshake frame is fragmented into multiple TCP segments (packets).
ttl (*equal* / *greater-than* / *less-than* / *not-equal : integer(0..255)*; Default:)	Matches packets TTL value.

Stats

To show additional *read-only* properties:

Property	Description
bytes (*integer*)	The total amount of bytes matched by the rule
packets (*integer*)	The total amount of packets matched by the rule

To print out stats:

```
[admin@MikroTik] > ip firewall mangle print stats all
Flags: X - disabled, I - invalid, D - dynamic
# CHAIN ACTION BYTES PACKETS
0 D ;;; special dummy rule to show fasttrack counters
prerouting passthrough 18 176 176 30 562
1 D ;;; special dummy rule to show fasttrack counters
forward passthrough 18 176 176 30 562
2 D ;;; special dummy rule to show fasttrack counters
postrouting passthrough 18 176 176 30 562
3 forward change-mss 18 512 356
```

Configuration example

Change MSS

It is a known fact that VPN links have a smaller packet size due to encapsulation overhead. A large packet with MSS that exceeds the MSS of the VPN link should be fragmented prior to sending it via that kind of connection. However, if the packet has a *Don't Fragment* flag set, it cannot be fragmented and should be discarded. On links that have broken path MTU discovery (PMTUD), it may lead to a number of problems, including problems with FTP and HTTP data transfer and e-mail services.

In the case of a link with broken PMTUD, a decrease of the MSS of the packets coming through the VPN link resolves the problem. The following example demonstrates how to decrease the MSS value via mangle:

```
/ip firewall mangle add out-interface=pppoe-out protocol=tcp tcp-flags=syn action=change-mss new-mss=1300
chain=forward tcp-mss=1301-65535
```

NAT

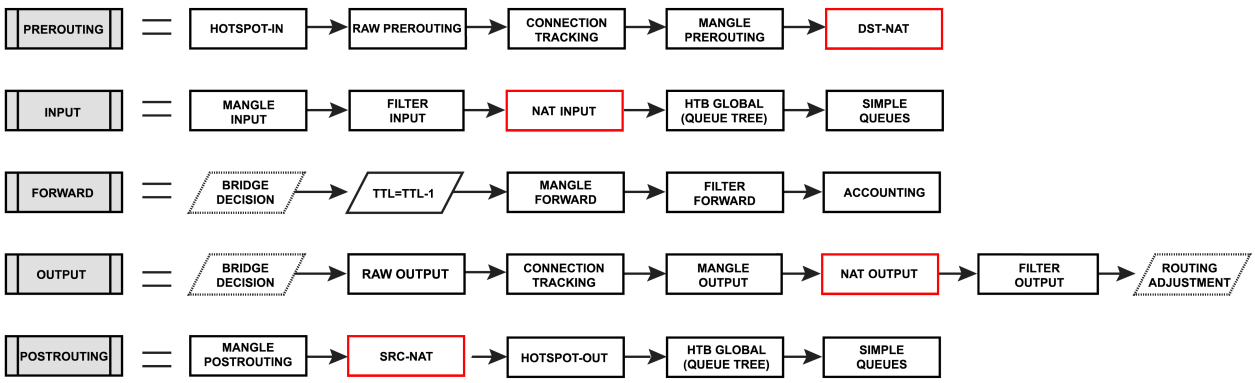
Introduction

Network Address Translation is an Internet standard that allows hosts on local area networks to use one set of IP addresses for internal communications and another set of IP addresses for external communications. A LAN that uses NAT is ascribed as a *natted* network. For NAT to function, there should be a NAT gateway in each *natted* network. The NAT gateway (NAT router) performs IP address rewriting on the way packet travel from/to LAN.

Nat matches only the first packet of the connection, connection tracking remembers the action and performs on all other packets belonging to the same connection.

> Whenever NAT rules are changed or added, the connection tracking table should be cleared otherwise NAT rules may seem to be not functioning correctly until connection entry expires.

Types of NAT:

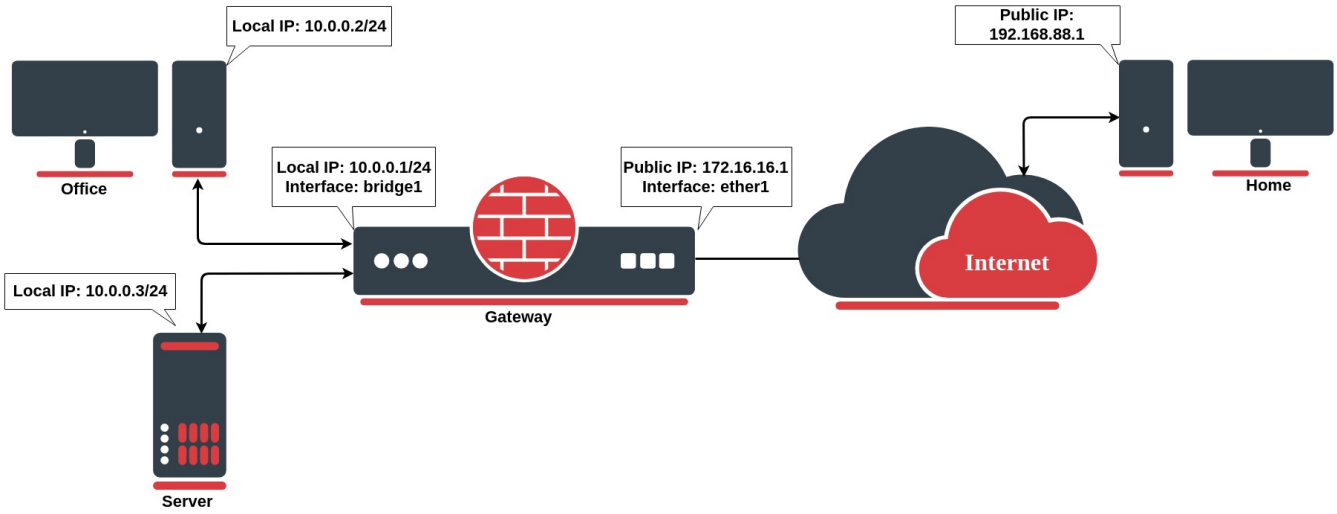


- **source NAT or srcnat.** This type of NAT is performed on packets that are originated from a natted network. A NAT router replaces the private source address of an IP packet with a new public IP address as it travels through the router. A reverse operation is applied to the reply packets traveling in the other direction.
- **destination NAT or dstnat.** This type of NAT is performed on packets that are destined for the natted network. It is most commonly used to make hosts on a private network to be accessible from the Internet. A NAT router performing dstnat replaces the destination IP address of an IP packet as it travels through the router towards a private network.

Since RouterOS v7 the firewall NAT has two new *INPUT* and *OUTPUT* chains which are traversed for packets delivered to and sent from applications running on the local machine:

- **input** - used to process packets entering the router through one of the interfaces with the destination IP address which is one of the router's addresses. Packets passing through the router are not processed against the rules of the input chain.
- **output** - used to process packets that originated from the router and leave it through one of the interfaces. Packets passing through the router are not processed against the rules of the output chain.

Destination NAT



Network address translation works by modifying network address information in the packets IP header. Let`s take a look at the common setup where a network administrator wants to access an office server from the internet.

We want to allow connections from the internet to the office server whose local IP is 10.0.0.3. In this case, we have to configure a destination address translation rule on the office gateway router:

```
/ip firewall nat add chain=dstnat action=dst-nat dst-address=172.16.16.1 dst-port=22 to-addresses=10.0.0.3
protocol=tcp
```

The rule above translates: when an incoming connection requests TCP port 22 with destination address 172.16.16.1, use the *dst-nat* action and depart packets to the device with local IP address 10.0.0.3 and port 22.

To allow access only from the PC at home, we can improve our *dst-nat* rule with *"src-address=192.168.88.1"* which is a Home`s PC public (this example) IP address. It is also considered to be more secure!

Source NAT

If you want to hide your local devices behind your public IP address received from ISP, you should configure the source network address translation (masquerading) feature of the MikroTik router.
Let`s assume you want to hide both office computer and server behind the public IP 172.16.16.1, the rule will look like the following one:

```
/ip firewall nat add chain=srcnat src-address=10.0.0.0/24 action=src-nat to-addresses=172.16.16.1 out-
interface=WAN
```

Now your ISP will see all the requests coming with IP 172.16.16.1 and they will not see your LAN network IP addresses.

Masquerade

Firewall NAT *action=masquerade* is a unique subversion of *action=srcnat,* it was designed for specific use in situations when public IP can randomly change, for example, DHCP server change assigned IP or PPPoE tunnel after disconnect gets different IP, in short - **when public IP is dynamic**.

```
/ip firewall nat add chain=srcnat src-address=10.0.0.0/24 action=masquerade out-interface=WAN
```

Every time when interface disconnects and/or its IP address changes, the router will clear all masqueraded connection tracking entries related to the interface, this way improving system recovery time after public IP change. If *srcnat* is used instead of *masquerade,* connection tracking entries remain and connections can simply resume after a link failure.

Unfortunately, this can lead to some issues with unstable links when the connection gets routed over different links after the primary link goes down. In such a scenario following things can happen:

- on disconnect, all related connection tracking entries are purged;
- next packet from every purged (previously masqueraded) connection will come into the firewall as *new*, and, if a primary interface is not back, a packet will be routed out via an alternative route (if you have any) thus creating a new masqueraded connection;
- the primary link comes back, routing is restored over the primary link, so packets that belong to existing connections are sent over the primary interface without being masqueraded, that way leaking local IPs to a public network.

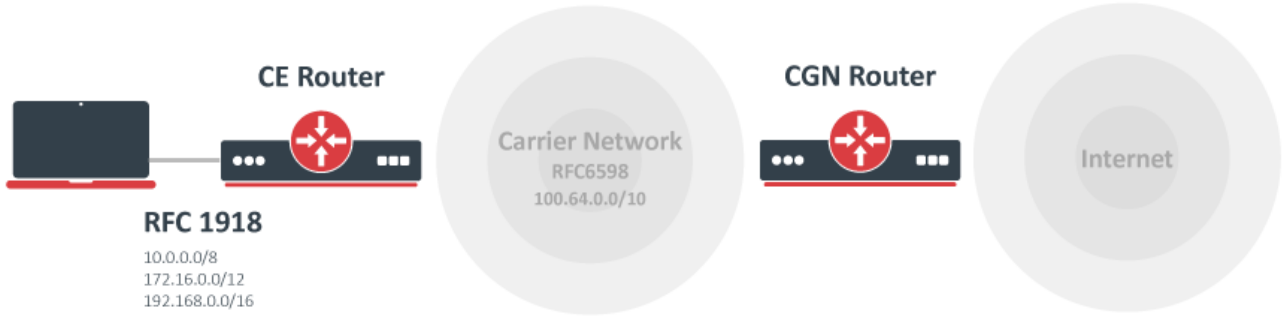
To work around this situation **blackhole** route can be created as an alternative to the route that might disappear on disconnect.

Hosts behind a NAT-enabled router do not have true end-to-end connectivity. Therefore some Internet protocols might not work in scenarios with NAT. Services that require the initiation of TCP connection from outside the private network or stateless protocols such as UDP, can be disrupted.

To overcome these limitations RouterOS includes a number of so-called NAT helpers, that enable NAT traversal for various protocols. When *action=srcnat* is used instead, connection tracking entries remain and connections can simply resume.

Though Source NAT and masquerading perform the same fundamental function: mapping one address space into another one, the details differ slightly. Most noticeably, masquerading chooses the source IP address for the outbound packet from the IP bound to the interface through which the packet will exit.

CGNAT (NAT444)



To combat IPv4 address exhaustion, a new RFC 6598 was deployed. The idea is to use shared 100.64.0.0/10 address space inside the carrier's network and perform NAT on the carrier's edge router to a single public IP or public IP range.

Because of the nature of such setup, it is also called NAT444, as opposed to a NAT44 network for a 'normal' NAT environment, three different IPv4 address spaces are involved.

CGNAT configuration on RouterOS does not differ from any other regular source NAT configuration:

```
/ip firewall nat
 add chain=src-nat action=srcnat src-address=100.64.0.0/10 to-address=2.2.2.2 out-interface=<public_if>
```

Where:

- 2.2.2.2 - public IP address,
- public_if - interface on providers edge router connected to the internet

The advantage of NAT444 is obvious, fewer public IPv4 addresses are used. But this technique comes with major drawbacks:

- The service provider router performing CGNAT needs to maintain a state table for all the address translations: this requires a lot of memory and CPU resources.
- Console gaming problems. Some games fail when two subscribers using the same outside public IPv4 address try to connect to each other.
- Tracking users for legal reasons means extra logging, as multiple households go behind one public address.

- Anything requiring incoming connections is broken. While this already was the case with regular NAT, end-users could usually still set up port forwarding on their NAT router. CGNAT makes this impossible. This means no web servers can be hosted here, and IP Phones cannot receive incoming calls by default either.
- Some web servers only allow a maximum number of connections from the same public IP address, as a means to counter DoS attacks like SYN floods. Using CGNAT this limit is reached more often and some services may be of poor quality.
- 6to4 requires globally reachable addresses and will not work in networks that employ addresses with a limited topological span.

Packets with Shared Address Space source or destination addresses MUST NOT be forwarded across Service Provider boundaries. Service Providers MUST filter such packets on ingress links. In RouterOS this can be easily done with firewall filters on edge routers:

```
/ip firewall filter
 add chain=input src-address=100.64.0.0/10 action=drop in-interface=<public_if>
 add chain=output dst-address=100.64.0.0/10 action=drop out-interface=<public_if>
 add chain=forward src-address=100.64.0.0/10 action=drop in-interface=<public_if>
 add chain=forward src-address=100.64.0.0/10 action=drop out-interface=<public_if>
 add chain=forward dst-address=100.64.0.0/10 action=drop out-interface=<public_if>
```

Service providers may be required to log of MAPed addresses, in a large CGN deployed network that may be a problem. Fortunately, RFC 7422 suggests a way to manage CGN translations in such a way as to significantly reduce the amount of logging required while providing traceability for abuse response.

RFC states that instead of logging each connection, CGNs could deterministically map customer private addresses (received on the customer-facing interface of the CGN, a.k.a., internal side) to public addresses extended with port ranges.

That means that separate NAT rules have to be added to achieve individual mappings such as the ones seen in the below example:

Inside IP	Outside IP/Port range
100.64.0.1	2.2.2.2:5000-5199
100.64.0.2	2.2.2.2:5200-5399
100.64.0.3	2.2.2.2:5400-5599
100.64.0.4	2.2.2.2:5600-5799
100.64.0.5	2.2.2.2:5800-5999

Instead of writing the rules by hand, it is suggested to use a script instead. The following example could be adapted to any requirements of your setup.

```
{
######## Adjustable values #########
:local StartingAddress 100.64.0.1
:local ClientCount 5
:local AddressesPerClient 2
:local PublicAddress 2.2.2.2
:local StartingPort 5000
:local PortsPerAddress 200
####################################

# All client chain jump
/ip firewall nat add chain=srcnat action=jump jump-target=clients \
    src-address="$StartingAddress-$($StartingAddress + ($ClientCount * $AddressesPerClient) - 1)"

:local currentPort $StartingPort

:for c from=1 to=$ClientCount do={
    # Specific client chain jumps
    :if ($AddressesPerClient > 1) do={
      /ip firewall nat add chain=clients action=jump jump-target="client-$c" \
      src-address="$($StartingAddress + ($AddressesPerClient * ($c - 1)))-$($StartingAddress +
($AddressesPerClient * $c -1))"
    } else={
      /ip firewall nat add chain=clients action=jump jump-target="client-$c" \
      src-address="$($StartingAddress + ($AddressesPerClient * ($c - 1)))"
    }
```

```
    # Translation rules
    :for a from=1 to=$AddressesPerClient do={
      /ip firewall nat add chain="client-$c" action=src-nat protocol=tcp \
      src-address="$($StartingAddress + (($c -1) * $AddressesPerClient) + $a - 1)" to-address=$PublicAddress to-
ports="$currentPort-$($currentPort + $PortsPerAddress - 1)"
      /ip firewall nat add chain="client-$c" action=src-nat protocol=udp \
      src-address="$($StartingAddress + (($c -1) * $AddressesPerClient) + $a - 1)" to-address=$PublicAddress to-
ports="$currentPort-$($currentPort + $PortsPerAddress - 1)"
      :set currentPort ($currentPort + $PortsPerAddress)
    }
}
}
```

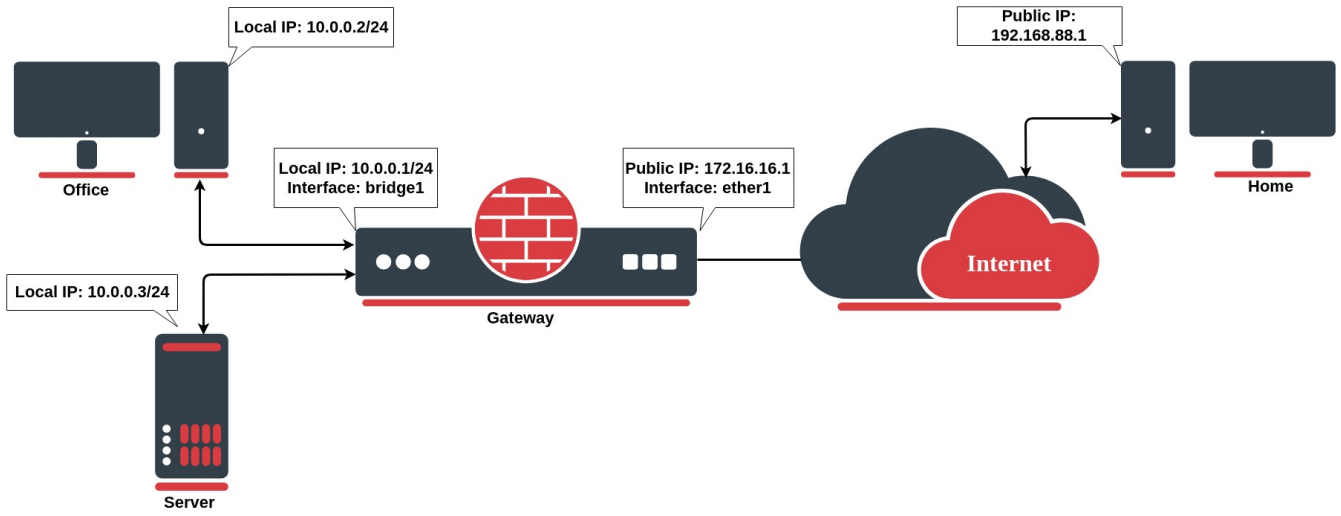
The six local values can be adjusted and the script can be either simply pasted in the terminal or it can be stored in the system script section, in case the configuration needs to be regenerated later.

After execution you should get a set of rules:

```
[admin@MikroTik] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
 0    chain=srcnat action=jump jump-target=clients
      src-address=100.64.0.1-100.64.0.10

 1    chain=clients action=jump jump-target=client-1
      src-address=100.64.0.1-100.64.0.2

 2    chain=client-1 action=src-nat to-addresses=2.2.2.2 to-ports=5000-5199
      protocol=tcp src-address=100.64.0.1

 3    chain=client-1 action=src-nat to-addresses=2.2.2.2 to-ports=5000-5199
      protocol=udp src-address=100.64.0.1

 4    chain=client-1 action=src-nat to-addresses=2.2.2.2 to-ports=5200-5399
      protocol=tcp src-address=100.64.0.2

 5    chain=client-1 action=src-nat to-addresses=2.2.2.2 to-ports=5200-5399
      protocol=udp src-address=100.64.0.2

 6    chain=clients action=jump jump-target=client-2
      src-address=100.64.0.3-100.64.0.4

 7    chain=client-2 action=src-nat to-addresses=2.2.2.2 to-ports=5400-5599
      protocol=tcp src-address=100.64.0.3

 8    chain=client-2 action=src-nat to-addresses=2.2.2.2 to-ports=5400-5599
      protocol=udp src-address=100.64.0.3

 9    chain=client-2 action=src-nat to-addresses=2.2.2.2 to-ports=5600-5799
      protocol=tcp src-address=100.64.0.4

10    chain=client-2 action=src-nat to-addresses=2.2.2.2 to-ports=5600-5799
      protocol=udp src-address=100.64.0.4

11    chain=clients action=jump jump-target=client-3
      src-address=100.64.0.5-100.64.0.6

12    chain=client-3 action=src-nat to-addresses=2.2.2.2 to-ports=5800-5999
      protocol=tcp src-address=100.64.0.5

13    chain=client-3 action=src-nat to-addresses=2.2.2.2 to-ports=5800-5999
      protocol=udp src-address=100.64.0.5

14    chain=client-3 action=src-nat to-addresses=2.2.2.2 to-ports=6000-6199
      protocol=tcp src-address=100.64.0.6

15    chain=client-3 action=src-nat to-addresses=2.2.2.2 to-ports=6000-6199
      protocol=udp src-address=100.64.0.6

[...]
```

Hairpin NAT

Hairpin network address translation (*NAT Loopback*) is where the device on the LAN is able to access another machine on the LAN via the public IP address of the gateway router.

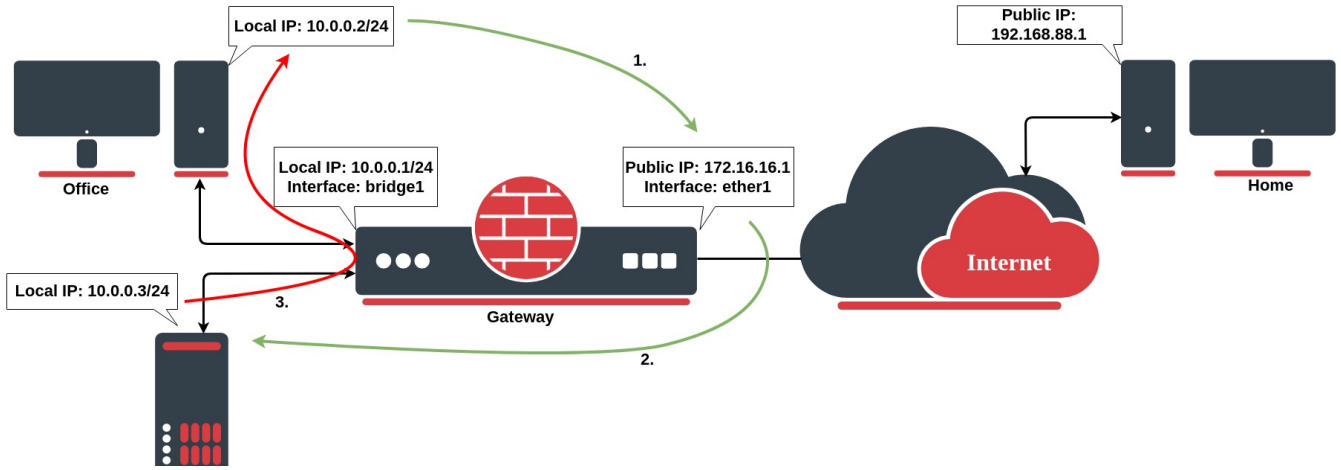


In the above example the gateway router has the following *dst-nat* configuration rule:

```
/ip firewall nat add chain=dstnat action=dst-nat dst-address=172.16.16.1 dst-port=443 to-addresses=10.0.0.3 to-ports=443 protocol=tcp
```

When a user from the PC at home establishes a connection to the webserver, the router performs DST NAT as configured:

1. the client sends a packet with a source IP address of 192.168.88.1 to a destination IP address of 172.16.16.1 on port 443 to request some web resources;
2. the router destination NAT`s the packet to 10.0.0.3 and replaces the destination IP address in the packet accordingly. The source IP address stays the same: 192.168.88.1;
3. the server replies to the client's request and the reply packet have a source IP address of 10.0.0.3 and a destination IP address of 192.168.88.1.
4. the router determines that the packet is part of a previous connection and undoes the destination NAT, and puts the original destination IP address into the source IP address field. The destination IP address is 192.168.88.1, and the source IP address is 172.16.16.1;
5. The client receives the reply packet it expects, and the connection is established;



But, there will be a **problem**, when a client on the same network as the web server requests a connection to the web server's **public** IP address:

1. the client sends a packet with a source IP address of 10.0.0.2 to a destination IP address of 172.16.16.1 on port 443 to request some web resources;
2. the router destination NATs the packet to 10.0.0.3 and replaces the destination IP address in the packet accordingly. The source IP address stays the same: 10.0.0.2;
3. the server replies to the client's request. However, the source IP address of the request is on the same subnet as the web server. The web server does not send the reply back to the router but sends it back directly to 10.0.0.2 with a source IP address in the reply of 10.0.0.3;
4. The client receives the reply packet, but it discards it because it expects a packet back from 172.16.16.1, and not from 10.0.0.3;

To resolve this issue, we will configure a new *src-nat* rule (the hairpin NAT rule) as follows:

```
/ip firewall nat
add action=masquerade chain=srcnat dst-address=10.0.0.3 out-interface=LAN protocol=tcp src-address=10.0.0.0/24
```

After configuring the rule above:

1. the client sends a packet with a source IP address of 10.0.0.2 to a destination IP address of 172.16.16.1 on port 443 to request some web resources;
2. the router destination NATs the packet to 10.0.0.3 and replaces the destination IP address in the packet accordingly. It also source NATs the packet and replaces the source IP address in the packet with the IP address on its LAN interface. The destination IP address is 10.0.0.3, and the source IP address is 10.0.0.1;
3. the web server replies to the request and sends the reply with a source IP address of 10.0.0.3 back to the router's LAN interface IP address of 10.0.0.1;
4. the router determines that the packet is part of a previous connection and undoes both the source and destination NAT, and puts the original destination IP address of 10.0.0.3 into the source IP address field, and the original source IP address of 172.16.16.1 into the destination IP address field

Endpoint-Independent NAT

Endpoint-independent NAT creates mapping in the source NAT and uses the same mapping for all subsequent packets with the same source IP and port. This mapping is created with the following rule:

```
/ip firewall nat
add action=endpoint-independent-nat chain=srcnat out-interface=WAN protocol=udp
```

This mapping allows running source-independent filtering, which allows forwarding packets from any source from WAN to mapped internal IP and port. Following rule enables filtering:

```
/ip firewall nat
add action=endpoint-independent-nat chain=dstnat in-interface=WAN protocol=udp
```

⚠ Endpoint-independent NAT works only with UDP protocol.

Additionally, endpoint-independent-nat can take a few other parameters:

- **randomize-port** - randomize to which public port connections will be mapped.

More info https://www.ietf.org/rfc/rfc5128.txt section 2.2.3 and 2.2.5

IPv4

Properties

Property	Description
action (*action name*; Default: **accept**)	Action to take if a packet is matched by the rule: • **accept** - accept the packet. A packet is not passed to the next NAT rule. • **add-dst-to-address-list** - add the destination address to the address list specified by `address-list` parameter • **add-src-to-address-list** - add the source address to the address list specified by a `ddress-list` parameter • **dst-nat** - replaces the destination address and/or port of an IP packet with values specified by `to-addresses` and `to-ports` parameters • **jump** - jump to the user-defined chain specified by the value of `jump-target` pa rameter • **log** - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar as `passt hrough` • **masquerade** - replaces the source port of an IP packet with one specified by `to-ports` parameter and replace the source address of an IP packet to the IP determined by the routing facility. • **netmap** - creates a static 1:1 mapping of one set of IP addresses to another one. Often used to distribute public IP addresses to hosts on private networks • **passthrough** - if a packet is matched by the rule, increase the counter and go to the next rule (useful for statistics). • **redirect** - replaces the destination port of an IP packet with one specified by `to-ports` parameter and destination address to one of the router's local addresses • **return** - passes control back to the chain from where the jump took place • **same** - gives a particular client the same source/destination IP address from a supplied range for each connection. This is most frequently used for services that expect the same client address for multiple connections from the same client • **src-nat** - replaces the source address of an IP packet with values specified by `to -addresses` and `to-ports` parameters • **endpoint-independent-nat** - uses endpoint-independent mapping and filtering. Works only with UDP protocol.
address-list (*string*; Default:)	Name of the address list to be used. Applicable if action is `add-dst-to-address-list` or `add-src-to-address-list`
address-list-timeout (*none-dynamic* | *none-static* | *time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till the reboot • Value of none-static will leave the address in the address list forever and will be included in the configuration export/backup
chain (*name*; Default:)	Specifies to which chain rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule
connection-bytes (*integer-integer*; Default:)	Matches packet only if a given amount of bytes has been transferred through the particular connection. 0 - means infinity, for example `connection-bytes=2000000-0` means that the rule matches if more than 2MB has been transferred through the relevant connection
connection-limit (*integer,netmask*; Default:)	Matches connections per address or address block after a given value is reached
connection-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular connection mark. If a **no-mark** is set, the rule will match any unmarked connection

connection-rate (*Integer 0..4294967295*; Default:)	Connection Rate is a firewall matcher that allows capturing traffic based on the present speed of the connection
connection-type (*ftp* | *h323* | *irc* | *pptp* | *quake3* | *sip* | *tftp*; Default:)	Matches packets from related connections based on information from their connection tracking helpers. A relevant connection helper must be enabled under /ip firewall service-port
content (*string*; Default:)	Match packets that contain specified text
dscp (*integer: 0..63*; Default:)	Matches DSCP IP header field.
dst-address (*IP/netmask* | *IP range*; Default:)	Matches packets whose destination is equal to specified IP or falls into a specified IP range.
dst-address-list (*name*; Default:)	Matches the destination address of a packet against a user-defined address list
dst-address-type (*unicast* | *local* | *broadcast* | *multicast*; Default:)	Matches destination address type: • unicast - IP address used for point-to-point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
dst-limit (*integer[/time],integer,dst-address* | *dst-port* | *src-address[/time]*; Default:)	Matches packets until a given PPS limit is exceeded. As opposed to the limit matcher, every destination IP address/destination port has its own limit. Parameters are written in the following format: `count[/time],burst,mode[/expire]`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional) • **burst** - number of packets that are not counted by packet rate • **mode** - the classifier for packet rate limiting • **expire** - specifies interval after which recorded IP address /port will be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges in format *Range[,Port]*, for example, *dst-port=123-345,456-678*
fragment (*yes*|*no*; Default:)	Matches fragmented packets. The first (starting) fragment does not count. If connection tracking is enabled there will be no fragments as the system automatically assembles every packet
hotspot (*auth* | *from-client* | *http* | *local-dst* | *to-client*; Default:)	Matches packets received from HotSpot clients against various HotSpot matchers. • auth - matches authenticated HotSpot client packets • from-client - matches packets that are coming from the HotSpot client • http - matches HTTP requests sent to the HotSpot server • local-dst - matches packets that are destined to the HotSpot server • to-client - matches packets that are sent to the HotSpot client
icmp-options (*integer:integer*; Default:)	Matches ICMP type: code fields
in-bridge-port (*name*; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge
in-interface (*name*; Default:)	Interface the packet has entered the router
ingress-priority (*integer: 0..63*; Default:)	Matches ingress the priority of the packet. Priority may be derived from VLAN, WMM or MPLS EXP bit.
ipsec-policy (*in* | *out, ipsec* | *none*; Default:)	Matches the policy used by IPSec. Value is written in the following format: **`direction, policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING, INPUT, and FORWARD chains • out - valid in the POSTROUTING, OUTPUT, and FORWARD chains • ipsec - matches if the packet is subject to IPsec processing;

	• none - matches packet that is not subject to IPsec processing (for example, IPSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then rule `ipsec-policy=in,ipsec` will match Gre packet, but the rule `ipsec-policy=in,none` will match the ESP packet.
ipv4-options (*any* | *loose-source-routing* | *no-record-route* | *no-router-alert* | *no-source-routing* | *no-timestamp* | *none* | *record-route* | *router-alert* | *strict-source-routing* | *timestamp*; Default:)	Matches IPv4 header options. • any - match packet with at least one of the ipv4 options • loose-source-routing - match packets with a loose source routing option. This option is used to route the internet datagram based on information supplied by the source • no-record-route - match packets with no record route option. This option is used to route the internet datagram based on information supplied by the source • no-router-alert - match packets with no router alter option • no-source-routing - match packets with no source routing option • no-timestamp - match packets with no timestamp option • record-route - match packets with record route option • router-alert - match packets with router alter option • strict-source-routing - match packets with a strict source routing option • timestamp - match packets with a timestamp
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
layer7-protocol (*name*; Default:)	Layer7 filter name defined in layer7 protocol menu.
limit (*integer,time,integer*; Default:)	Matches packets until a given PPS limit is exceeded. Parameters are written in the following format: `count[/time],burst`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional, 1s will be used if not specified) • **burst** - number of packets that are not counted by packet rate
log (*yes* | *no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge
out-interface (; Default:)	Interface the packet is leaving the router
packet-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular packet mark. If **no-mark** is set, the rule will match any unmarked packet
packet-size (*integer[-integer]:0..65535*; Default:)	Matches packets of specified size or size range in bytes
per-connection-classifier (*ValuesToHash:Denominator/Remainder*; Default:)	PCC matcher allows dividing traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
psd (*integer,time,integer,integer*; Default:)	Attempts to detect TCP and UDP scans. Parameters are in the following format `WeightThreshold, DelayThreshold, LowPortWeight, HighPortWeight` • **WeightThreshold** - total weight of the latest TCP/UDP packets with different destination ports coming from the same host to be treated as port scan sequence • **DelayThreshold** - delay for the packets with different destination ports coming from the same host to be treated as possible port scan subsequence • **LowPortWeight** - the weight of the packets with privileged (<1024) destination port • **HighPortWeight** - the weight of the packet with a non-privileged destination port

random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability
routing-mark (*string*; Default:)	Matches packets marked by mangle facility with particular routing mark
same-not-by-dst (*yes* | *no*; Default:)	Specifies whether to take into account or not destination IP address when selecting a new source IP address. Applicable if `action=same`
src-address (*Ip/Netmaks, Ip range*; Default:)	Matches packets whose source is equal to specified IP or falls into a specified IP range.
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast* | *local* | *broadcast* | *multicast*; Default:)	Matches source address type: • unicast - IP address used for point-to-point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP.
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet
tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat* | *fri* | *thu* | *wed* | *tue* | *mon* | *sun*; Default:)	Allows to create a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
to-addresses (*IP address[-IP address]*; Default: **0.0.0.0**)	Replace the original address with the specified one. Applicable if action is dst-nat, netmap, same, src-nat
to-ports (*integer[-integer]: 0..65535*; Default:)	Replace the original port with the specified one. Applicable if action is dst-nat, redirect, masquerade, netmap, same, src-nat
ttl (*integer: 0..255*; Default:)	Matches packets TTL value

Stats

Property	Description
bytes (*integer*)	The total amount of bytes matched by the rule
packets (*integer*)	The total amount of packets matched by the rule

To show additional *read-only* properties:

```
[admin@MikroTik] > ip firewall nat print stats all
Flags: X - disabled, I - invalid, D - dynamic
# CHAIN ACTION BYTES PACKETS
0 srcnat masquerade 265 659 987
```

IPv6

NAT66 is supported since RouterOS v7.1.

```
ipv6/firewall/nat/
```

Properties

Property	Description
action (*action name*; Default: **accept**)	Action to take if a packet is matched by the rule: • accept - accept the packet. A packet is not passed to the next NAT rule. • add-dst-to-address-list - add the destination address to the address list specified by `address-list` parameter • add-src-to-address-list - add the source address to the address list specified by `address-list` parameter • dst-nat - replaces destination address and/or port of an IP packet to values specified by `to-addresses` and `to-ports` parameters • jump - jump to the user-defined chain specified by the value of `jump-target` parameter • log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar as `passthrough` • masquerade - replaces the source port of an IP packet with one specified by `to-ports` parameter and replace the source address of an IP packet to IP determined by the routing facility. • netmap - creates a static 1:1 mapping of one set of IP addresses to another one. Often used to distribute public IP addresses to hosts on private networks • passthrough - if a packet is matched by the rule, increase the counter and go to the next rule (useful for statistics). • redirect - replaces the destination port of an IP packet with one specified by `to-ports` parameter and destination address to one of the router's local addresses • return - passes control back to the chain from where the jump took place • src-nat - replaces the source address of an IP packet with values specified by `to-addresses` and `to-ports` parameters
address-list (*string*; Default:)	Name of the address list to be used. Applicable if action is `add-dst-to-address-list` or `add-src-to-address-list`
address-list-timeout (*none-dynamic | none-static | time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till reboot • Value of none-static will leave the address in the address list forever and will be included in configuration export/backup
chain (*name*; Default:)	Specifies to which chain rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule
connection-bytes (*integer-integer*; Default:)	Matches packets only if a given amount of bytes has been transferred through the particular connection. 0 - means infinity, for example `connection-bytes=2000000-0` means that the rule matches if more than 2MB has been transferred through the relevant connection
connection-limit (*integer,netmask*; Default:)	Matches connections per address or address block after a given value is reached
connection-mark (*no-mark | string*; Default:)	Matches packets marked via mangle facility with particular connection mark. If **no-mark** is set, the rule will match any unmarked connection
connection-rate (*Integer 0..4294967295*; Default:)	Connection Rate is a firewall matcher that allows capturing traffic based on the present speed of the connection
connection-state (*established | invalid | new | related | untracked*; Default:)	Interprets the connection tracking analytics data for a particular packet: • established - a packet that belongs to an existing connection • invalid - a packet that does not have a determined state in connection tracking (usually - severe out-of-order packets, packets with wrong sequence/ack number, or in case of a resource over usage on the router), for this reason, an invalid packet will not participate in NAT (as only connection-state=new packets do), and will still contain original source IP address when routed. We strongly suggest dropping all *connection-state=invalid* packets in firewall filter forward and input chains • new - the packet has started a new connection, or is otherwise associated with a connection that has not seen packets in both directions.

	• related - a packet that is related to, but not parts of an existing connection, such as ICMP errors or a packet that begins FTP data connection • an untracked - packet which was set to bypass connection tracking in firewall RAW tables.
connection-type (*ftp | h323 | irc | pptp | quake3 | sip | tftp*; Default:)	Matches packets from related connections based on information from their connection tracking helpers. A relevant connection helper must be enabled under /ip firewall service-port
content (*string*; Default:)	Match packets that contain specified text
dscp (*integer: 0..63* ; Default:)	Matches DSCP IP header field.
dst-address (*IP /netmask | IP range* ; Default:)	Matches packets whose destination is equal to specified IP or falls into a specified IP range.
dst-address-list (*name*; Default:)	Matches destination address of a packet against user-defined address list
dst-address-type (*unicast | local | broadcast | multicast*; Default:)	Matches destination address type: • unicast - IP address used for point-to-point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
dst-limit (*integer[/time],integer,dst-address | dst-port | src-address[/time]*; Default:)	Matches packets until a given PPS limit is exceeded. As opposed to the limit matcher, every destination IP address /destination port has its own limit. Parameters are written in the following format: `count[/time],burst,mode[/expire]`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional) • **burst** - number of packets that are not counted by packet rate • **mode** - the classifier for packet rate limiting • **expire** - specifies interval after which recorded IP address /port will be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges in format *Range[,Port]*, for example, *dst-port=123-345,456-678*
icmp-options (*integer:integer*; Default:)	Matches ICMP type: code fields
in-bridge-port (*name* ; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge
in-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-bridge-port
in-interface (*name*; Default:)	Interface the packet has entered the router
in-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-interface
ingress-priority (*integer: 0..63*; Default:)	Matches ingress the priority of the packet. Priority may be derived from VLAN, WMM or MPLS EXP bit.
ipsec-policy (*in | out, ipsec | none*; Default:)	Matches the policy used by IPSec. Value is written in the following format: **`direction, policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING, INPUT, and FORWARD chains • out - valid in the POSTROUTING, OUTPUT, and FORWARD chains

	• ipsec - matches if the packet is subject to IPsec processing; • none - matches packet that is not subject to IPsec processing (for example, IPSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then rule `ipsec-policy=in,ipsec` will match Gre packet, but the rule `ipsec-policy=in,none` will match the ESP packet.
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
layer7-protocol (*name*; Default:)	Layer7 filter name defined in layer7 protocol menu.
limit (*integer,time, integer*; Default:)	Matches packets until a given PPS limit is exceeded. Parameters are written in the following format: `count[/time],burst`. • **count** - maximum average packet rate measured in packets per `time` interval • **time** - specifies the time interval in which the packet rate is measured (optional, 1s will be used if not specified) • **burst** - number of packets that are not counted by packet rate
log (*yes* | *no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge
out-bridge-port-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-bridge-port
out-interface (; Default:)	Interface the packet is leaving the router
out-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-interface
packet-mark (*no-mark* | *string*; Default:)	Matches packets marked via mangle facility with particular packet mark. If **no-mark** is set, the rule will match any unmarked packet
packet-size (*integer[-integer]:0..65535* ; Default:)	Matches packets of specified size or size range in bytes
per-connection-classifier (*ValuesToHash: Denominator /Remainder*; Default:)	PCC matcher allows dividing traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
priority (*integer: 0..63*; Default:)	Matches the packet's priority after a new priority has been set. Priority may be derived from VLAN, WMM, DSCP, MPLS EXP bit, or from the priority that has been set using the set-priority action.
random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability
routing-mark (*string* ; Default:)	Matches packets marked by mangle facility with particular routing mark

src-address (*Ip /Netmaks, Ip range*; Default:)	Matches packets whose source is equal to specified IP or falls into a specified IP range.
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast | local | broadcast | multicast*; Default:)	Matches source address type: • unicast - IP address used for point-to-point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP.
tcp-flags (*ack | cwr | ece | fin | psh | rst | syn | urg*; Default:)	Matches specified TCP flags • ack - acknowledging data • cwr - congestion window reduced • ece - ECN-echo flag (explicit congestion notification) • fin - close connection • psh - push function • rst - drop connection • syn - new connection • urg - urgent data
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet
tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat | fri | thu | wed | tue | mon | sun*; Default:)	Allows to create a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
to-addresses (*IP address[-IP address]*; Default: **0.0.0.0**)	Replace the original address with the specified one. Applicable if action is dst-nat, netmap, same, src-nat
to-ports (*integer[-integer]: 0..65535*; Default:)	Replace the original port with the specified one. Applicable if action is dst-nat, redirect, masquerade, netmap, same, src-nat

Stats

Property	Description
bytes (*integer*)	The total amount of bytes matched by the rule
packets (*integer*)	The total amount of packets matched by the rule

To show additional *read-only* properties:

```
ipv6/firewall/nat/print stats
```

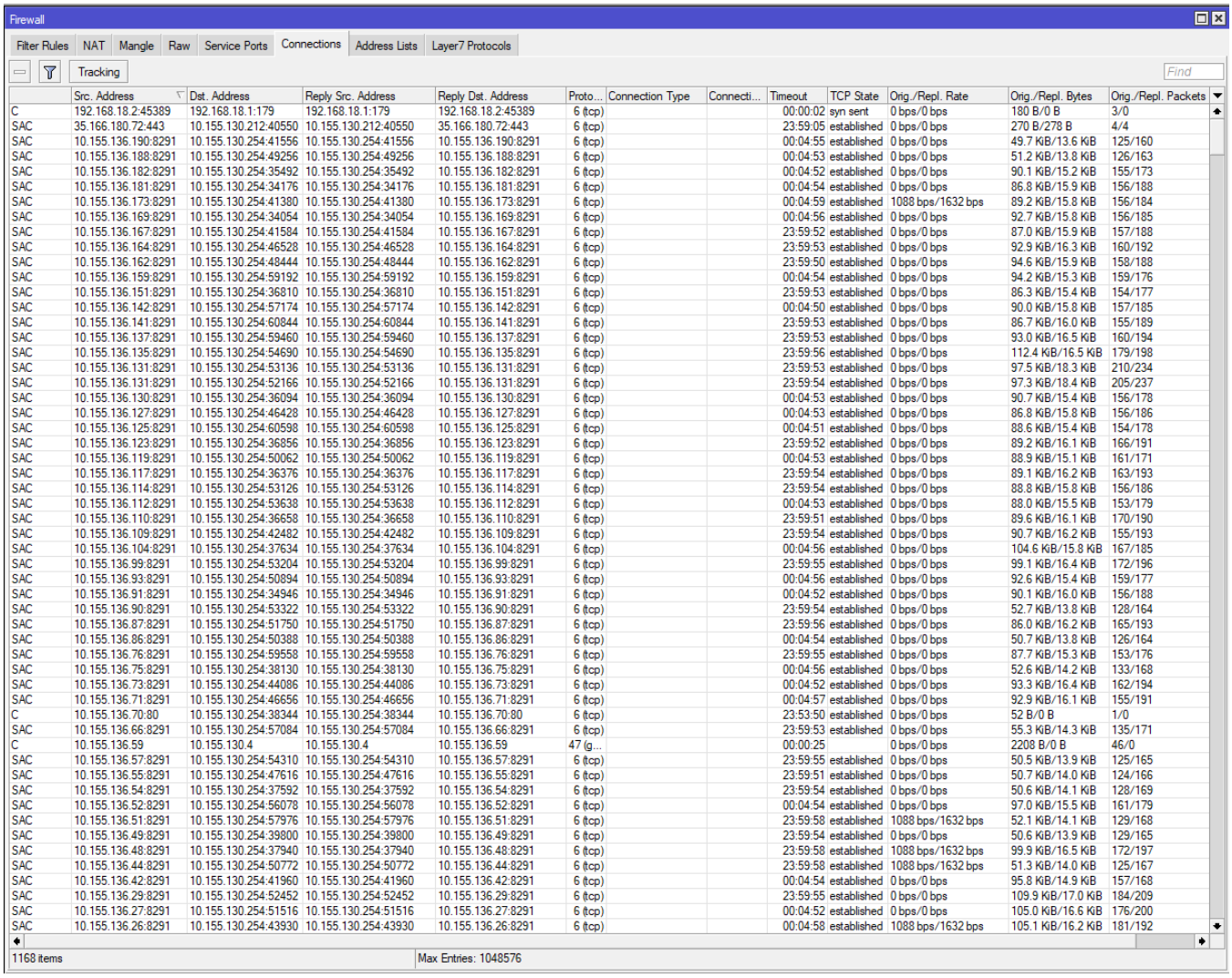
Connection tracking

Introduction

```
/ip firewall connection
```

There are several ways to see what connections are making their way through the router.

In the Winbox Firewall window, you can switch to the Connections tab, to see current connections to/from/through your router. It looks like this:



Src. Address	Dst. Address	Reply Src. Address	Reply Dst. Address	Proto...	Connection Type	Connecti...	Timeout	TCP State	Orig./Repl. Rate	Orig./Repl. Bytes	Orig./Repl. Packets
C	192.168.18.2:45389	192.168.18.1:179	192.168.18.1:179	192.168.18.2:45389	6 (tcp)		00:00:02	syn sent	0 bps/0 bps	180 B/0 B	3/0
SAC	35.166.180.72:443	10.155.130.212:40550	10.155.130.212:40550	35.166.180.72:443	6 (tcp)		23:59:05	established	0 bps/0 bps	270 B/278 B	4/4
SAC	10.155.136.190:8291	10.155.130.254:41556	10.155.130.254:41556	10.155.136.190:8291	6 (tcp)		00:04:55	established	0 bps/0 bps	49.7 KiB/13.6 KiB	125/160
SAC	10.155.136.188:8291	10.155.130.254:49256	10.155.130.254:49256	10.155.136.188:8291	6 (tcp)		00:04:53	established	0 bps/0 bps	51.2 KiB/13.8 KiB	126/163
SAC	10.155.136.182:8291	10.155.130.254:35492	10.155.130.254:35492	10.155.136.182:8291	6 (tcp)		00:04:52	established	0 bps/0 bps	90.1 KiB/15.2 KiB	155/173
SAC	10.155.136.181:8291	10.155.130.254:34176	10.155.130.254:34176	10.155.136.181:8291	6 (tcp)		00:04:54	established	0 bps/0 bps	86.8 KiB/15.9 KiB	156/188
SAC	10.155.136.173:8291	10.155.130.254:41380	10.155.130.254:41380	10.155.136.173:8291	6 (tcp)		00:04:59	established	1088 bps/1632 bps	89.2 KiB/15.8 KiB	156/184
SAC	10.155.136.169:8291	10.155.130.254:34054	10.155.130.254:34054	10.155.136.169:8291	6 (tcp)		00:04:56	established	0 bps/0 bps	92.7 KiB/15.8 KiB	156/185
SAC	10.155.136.167:8291	10.155.130.254:41584	10.155.130.254:41584	10.155.136.167:8291	6 (tcp)		23:59:52	established	0 bps/0 bps	87.0 KiB/15.9 KiB	157/188
SAC	10.155.136.164:8291	10.155.130.254:46528	10.155.130.254:46528	10.155.136.164:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	92.9 KiB/16.3 KiB	160/192
SAC	10.155.136.162:8291	10.155.130.254:48444	10.155.130.254:48444	10.155.136.162:8291	6 (tcp)		23:59:50	established	0 bps/0 bps	94.6 KiB/15.9 KiB	158/188
SAC	10.155.136.159:8291	10.155.130.254:59192	10.155.130.254:59192	10.155.136.159:8291	6 (tcp)		00:04:54	established	0 bps/0 bps	94.2 KiB/15.3 KiB	159/176
SAC	10.155.136.151:8291	10.155.130.254:36810	10.155.130.254:36810	10.155.136.151:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	86.3 KiB/15.4 KiB	154/177
SAC	10.155.136.142:8291	10.155.130.254:57174	10.155.130.254:57174	10.155.136.142:8291	6 (tcp)		00:04:50	established	0 bps/0 bps	90.0 KiB/15.8 KiB	157/185
SAC	10.155.136.141:8291	10.155.130.254:60844	10.155.130.254:60844	10.155.136.141:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	86.7 KiB/16.0 KiB	155/189
SAC	10.155.136.137:8291	10.155.130.254:59460	10.155.130.254:59460	10.155.136.137:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	93.0 KiB/16.5 KiB	160/194
SAC	10.155.136.135:8291	10.155.130.254:54690	10.155.130.254:54690	10.155.136.135:8291	6 (tcp)		23:59:56	established	0 bps/0 bps	112.4 KiB/16.5 KiB	179/198
SAC	10.155.136.131:8291	10.155.130.254:53136	10.155.130.254:53136	10.155.136.131:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	97.5 KiB/18.3 KiB	210/234
SAC	10.155.136.131:8291	10.155.130.254:52166	10.155.130.254:52166	10.155.136.131:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	97.3 KiB/18.4 KiB	205/237
SAC	10.155.136.130:8291	10.155.130.254:36094	10.155.130.254:36094	10.155.136.130:8291	6 (tcp)		00:04:53	established	0 bps/0 bps	90.7 KiB/15.4 KiB	156/178
SAC	10.155.136.127:8291	10.155.130.254:46428	10.155.130.254:46428	10.155.136.127:8291	6 (tcp)		00:04:53	established	0 bps/0 bps	86.8 KiB/15.8 KiB	156/186
SAC	10.155.136.125:8291	10.155.130.254:60598	10.155.130.254:60598	10.155.136.125:8291	6 (tcp)		00:04:51	established	0 bps/0 bps	88.6 KiB/15.4 KiB	154/178
SAC	10.155.136.123:8291	10.155.130.254:36856	10.155.130.254:36856	10.155.136.123:8291	6 (tcp)		23:59:52	established	0 bps/0 bps	89.2 KiB/16.1 KiB	166/191
SAC	10.155.136.119:8291	10.155.130.254:50062	10.155.130.254:50062	10.155.136.119:8291	6 (tcp)		00:04:53	established	0 bps/0 bps	88.9 KiB/15.1 KiB	161/171
SAC	10.155.136.117:8291	10.155.130.254:36376	10.155.130.254:36376	10.155.136.117:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	89.1 KiB/16.2 KiB	163/193
SAC	10.155.136.114:8291	10.155.130.254:53126	10.155.130.254:53126	10.155.136.114:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	88.8 KiB/15.8 KiB	156/186
SAC	10.155.136.112:8291	10.155.130.254:53638	10.155.130.254:53638	10.155.136.112:8291	6 (tcp)		00:04:53	established	0 bps/0 bps	88.0 KiB/15.5 KiB	153/179
SAC	10.155.136.110:8291	10.155.130.254:36658	10.155.130.254:36658	10.155.136.110:8291	6 (tcp)		23:59:51	established	0 bps/0 bps	89.6 KiB/16.1 KiB	170/190
SAC	10.155.136.109:8291	10.155.130.254:42482	10.155.130.254:42482	10.155.136.109:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	90.7 KiB/16.2 KiB	155/193
SAC	10.155.136.104:8291	10.155.130.254:37634	10.155.130.254:37634	10.155.136.104:8291	6 (tcp)		00:04:56	established	0 bps/0 bps	104.6 KiB/15.8 KiB	167/185
SAC	10.155.136.99:8291	10.155.130.254:53204	10.155.130.254:53204	10.155.136.99:8291	6 (tcp)		23:59:55	established	0 bps/0 bps	99.1 KiB/16.4 KiB	172/196
SAC	10.155.136.93:8291	10.155.130.254:50894	10.155.130.254:50894	10.155.136.93:8291	6 (tcp)		00:04:56	established	0 bps/0 bps	92.6 KiB/15.4 KiB	159/177
SAC	10.155.136.91:8291	10.155.130.254:34946	10.155.130.254:34946	10.155.136.91:8291	6 (tcp)		00:04:52	established	0 bps/0 bps	90.1 KiB/16.0 KiB	156/188
SAC	10.155.136.90:8291	10.155.130.254:53322	10.155.130.254:53322	10.155.136.90:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	52.7 KiB/13.8 KiB	128/164
SAC	10.155.136.87:8291	10.155.130.254:51750	10.155.130.254:51750	10.155.136.87:8291	6 (tcp)		23:59:56	established	0 bps/0 bps	86.0 KiB/16.2 KiB	165/193
SAC	10.155.136.86:8291	10.155.130.254:50388	10.155.130.254:50388	10.155.136.86:8291	6 (tcp)		00:04:54	established	0 bps/0 bps	50.7 KiB/13.8 KiB	126/164
SAC	10.155.136.76:8291	10.155.130.254:59558	10.155.130.254:59558	10.155.136.76:8291	6 (tcp)		23:59:55	established	0 bps/0 bps	87.7 KiB/15.3 KiB	153/176
SAC	10.155.136.75:8291	10.155.130.254:38130	10.155.130.254:38130	10.155.136.75:8291	6 (tcp)		00:04:56	established	0 bps/0 bps	52.6 KiB/14.2 KiB	133/168
SAC	10.155.136.73:8291	10.155.130.254:44086	10.155.130.254:44086	10.155.136.73:8291	6 (tcp)		00:04:52	established	0 bps/0 bps	93.3 KiB/16.4 KiB	162/194
SAC	10.155.136.71:8291	10.155.130.254:46656	10.155.130.254:46656	10.155.136.71:8291	6 (tcp)		00:04:57	established	0 bps/0 bps	92.9 KiB/16.1 KiB	155/191
C	10.155.136.70:80	10.155.130.254:38344	10.155.130.254:38344	10.155.136.70:80	6 (tcp)		23:53:50	established	0 bps/0 bps	52 B/0 B	1/0
SAC	10.155.136.66:8291	10.155.130.254:57084	10.155.130.254:57084	10.155.136.66:8291	6 (tcp)		23:59:53	established	0 bps/0 bps	55.3 KiB/14.3 KiB	135/171
C	10.155.136.59	10.155.130.4	10.155.130.4	10.155.136.59	47 (g...		00:00:25		0 bps/0 bps	2208 B/0 B	46/0
SAC	10.155.136.57:8291	10.155.130.254:54310	10.155.130.254:54310	10.155.136.57:8291	6 (tcp)		23:59:55	established	0 bps/0 bps	50.5 KiB/13.9 KiB	125/165
SAC	10.155.136.55:8291	10.155.130.254:47616	10.155.130.254:47616	10.155.136.55:8291	6 (tcp)		23:59:51	established	0 bps/0 bps	50.7 KiB/14.0 KiB	124/166
SAC	10.155.136.54:8291	10.155.130.254:37592	10.155.130.254:37592	10.155.136.54:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	50.6 KiB/14.1 KiB	128/169
SAC	10.155.136.52:8291	10.155.130.254:56078	10.155.130.254:56078	10.155.136.52:8291	6 (tcp)		00:04:54	established	0 bps/0 bps	97.0 KiB/15.5 KiB	161/179
SAC	10.155.136.51:8291	10.155.130.254:57976	10.155.130.254:57976	10.155.136.51:8291	6 (tcp)		23:59:58	established	1088 bps/1632 bps	52.1 KiB/14.1 KiB	129/168
SAC	10.155.136.49:8291	10.155.130.254:39800	10.155.130.254:39800	10.155.136.49:8291	6 (tcp)		23:59:54	established	0 bps/0 bps	50.6 KiB/13.9 KiB	129/165
SAC	10.155.136.48:8291	10.155.130.254:37940	10.155.130.254:37940	10.155.136.48:8291	6 (tcp)		23:59:58	established	1088 bps/1632 bps	99.9 KiB/16.5 KiB	172/197
SAC	10.155.136.44:8291	10.155.130.254:50772	10.155.130.254:50772	10.155.136.44:8291	6 (tcp)		23:59:58	established	1088 bps/1632 bps	51.3 KiB/14.0 KiB	125/167
SAC	10.155.136.42:8291	10.155.130.254:41960	10.155.130.254:41960	10.155.136.42:8291	6 (tcp)		00:04:54	established	0 bps/0 bps	95.8 KiB/14.9 KiB	157/168
SAC	10.155.136.29:8291	10.155.130.254:52452	10.155.130.254:52452	10.155.136.29:8291	6 (tcp)		23:59:55	established	0 bps/0 bps	109.9 KiB/17.0 KiB	184/209
SAC	10.155.136.27:8291	10.155.130.254:51516	10.155.130.254:51516	10.155.136.27:8291	6 (tcp)		00:04:52	established	0 bps/0 bps	105.0 KiB/16.6 KiB	176/200
SAC	10.155.136.26:8291	10.155.130.254:43930	10.155.130.254:43930	10.155.136.26:8291	6 (tcp)		00:04:58	established	1088 bps/1632 bps	105.1 KiB/16.2 KiB	181/192

Properties

All properties in the connection list are read-only

Property	Description

assured (*yes* | *no*)	Indicates that this connection is assured and that it will not be erased if the maximum possible tracked connection count is reached.
confirmed (*yes* | *no*)	Connection is confirmed and a packet is sent out from the device.
connection-mark (*string*)	Connection mark that was set by mangle rule.
connection-type (*pptp* | *ftp*)	Type of connection, the property is empty if connection tracking is unable to determine a predefined connection type.
dst-address (*ip[:port]*)	Destination address and port (if a protocol is port-based).
dstnat (*yes* | *no*)	A connection has gone through DST-NAT (for example, port forwarding).
dying (*yes* | *no*)	The connection is dying due to connection timeout.
expected (*yes* | *no*)	Connection is set up using connection helpers (pre-defined service rules).
fasttrack (*yes* | *no*)	Whether the connection is FastTracked.
gre-key (*integer*)	Contents of the GRE Key field.
gre-protocol (*string*)	Protocol of the encapsulated payload.
gre-version (*string*)	A version of the GRE protocol was used in the connection.
icmp-code (*string*)	ICMP Code Field
icmp-id (*integer*)	Contains the ICMP ID
icmp-type (*integer*)	ICMP Type Number
orig-bytes (*integer*)	Amount of bytes sent out from the source address using the specific connection.
orig-fasttrack-bytes (*integer*)	Amount of FastTracked bytes sent out from the source address using the specific connection.
orig-fasttrack-packets (*integer*)	Amount of FastTracked packets sent out from the source address using the specific connection.
orig-packets (*integer*)	Amount of packets sent out from the source address using the specific connection.
orig-rate (*integer*)	The data rate at which packets are sent out from the source address using the specific connection.
protocol (*string*)	IP protocol type
repl-bytes (*integer*)	Amount of bytes received from the destination address using the specific connection.
repl-fasttrack-bytes (*string*)	Amount of FastTracked bytes received from the destination address using the specific connection.
repl-fasttrack-packets (*integer*)	Amount of FastTracked packets received from the destination address using the specific connection.
repl-packets (*integer*)	Amount of packets received from the destination address using the specific connection.
repl-rate (*string*)	The data rate at which packets are received from the destination address using the specific connection.
reply-dst-address (*ip[:port]*)	Destination address (and port) expected of return packets. Usually the same as "src-address: port"
reply-src-address (*ip[:port]*)	Source address (and port) expected of return packets. Usually the same as "dst-address: port"
seen-reply (*yes* | *no*)	The destination address has replied to the source address.
src-address (*ip[:port]*)	The source address and port (if a protocol is port-based).
srcnat (*yes* | *no*)	Connection is going through SRC-NAT, including packets that were masqueraded through NAT.

tcp-state (*string*)	The current state of TCP connection : • "established" • "time-wait" • "close" • "syn-sent" • "syn-received"
timeout (*time*)	Time after connection will be removed from the connection list.

Connection tracking settings

```
/ip firewall connection tracking
```

Properties

Property	Description
enabled (*yes* | *no* | *auto*; Default: **auto**)	Allows to disable or enable connection tracking. Disabling connection tracking will cause several firewall features to stop working. See the list of affected features. Starting from v6.0rc2 default value is auto. This means that connection tracing is disabled until at least one firewall rule is added.
loose-tcp-tracking (*yes*; Default: **yes**)	Disable picking up already established connections
tcp-syn-sent-timeout (*time*; Default: **5s**)	TCP SYN timeout.
tcp-syn-received-timeout (*time*; Default: **5s**)	TCP SYN timeout.
tcp-established-timeout (*time*; Default: **1d**)	Time when established TCP connection times out.
tcp-fin-wait-timeout (*time*; Default: **10s**)	
tcp-close-wait-timeout (*time*; Default: **10s**)	
tcp-last-ack-timeout (*time*; Default: **10s**)	
tcp-time-wait-timeout (*time*; Default: **10s**)	
tcp-close-timeout (*time*; Default: **10s**)	
udp-timeout (*time* ; Default: **30s**)	Specifies the timeout for UDP connections that have seen packets in one direction
udp-stream-timeout (*time*; Default: **3m**)	Specifies the timeout of UDP connections that has seen packets in both directions

icmp-timeout (*time*; Default: **10s**)	ICMP connection timeout
generic-timeout (*time*; Default: **10m**)	Timeout for all other connection entries

Read-only properties

Property	Description
max-entries (*integer*)	Max amount of entries that the connection tracking table can hold. This value depends on the installed amount of RAM. Note that the system does not create a maximum-size connection tracking table when it starts, it may increase if the situation demands it and the system still has free ram, but size will not exceed 1048576
total-entries (*integer*)	Amount of connections that currently connection table holds.

Features affected by connection tracking

- NAT
- firewall:
  - connection-bytes
  - connection-mark
  - connection-type
  - connection-state
  - connection-limit
  - connection-rate
  - layer7-protocol
  - new-connection-mark
  - tarpit

Queues

Overview

A queue is a collection of data packets collectively waiting to be transmitted by a network device using a pre-defined structure methodology. Queuing works almost on the same methodology used at banks or supermarkets, where the customer is treated according to its arrival.

Queues are used to:

- limit data rate for certain IP addresses, subnets, protocols, ports, etc.;
- limit peer-to-peer traffic;
- packet prioritization;
- configure traffic bursts for traffic acceleration;
- apply different time-based limits;
- share available traffic among users equally, or depending on the load of the channel

Queue implementation in MikroTik RouterOS is based on Hierarchical Token Bucket (HTB). HTB allows to the creation of a hierarchical queue structure and determines relations between queues. These hierarchical structures can be attached at two different places, the Packet Flow diagram illustrate both *inp ut* and *postrouting* chains.

There are two different ways how to configure queues in RouterOS:

- **/queue simple** menu - designed to ease configuration of simple, every day queuing tasks (such as single client upload/download limitation, p2p traffic limitation, etc.).
- **/queue tree** menu - for implementing advanced queuing tasks (such as global prioritization policy, user group limitations). Requires marked packet flows from **/ip firewall mangle** facility.

RouterOS provides a possibility to configure queue in 8 levels - the first level is an interface queue from "/queue interface" menu and the other 7 are lower-level queues that can be created in Queue Simple and/or Queue Tree.

Rate limitation principles

Rate limiting is used to control the rate of traffic flow sent or received on a network interface. Traffic which rate that is less than or equal to the specified rate is sent, whereas traffic that exceeds the rate is dropped or delayed.

Rate limiting can be performed in two ways:

1. discard all packets that exceed rate limit – ***rate-limiting (dropper or shaper)*** *(100% rate limiter when queue-size=0)*
2. delay packets that exceed specific rate limit in the queue and transmit its when it is possible – ***rate equalizing (scheduler)*** (100% rate equalizing when *queue-size=unlimited*)

Next figure explains the difference between *rate limiting* and rate *equalizing*:

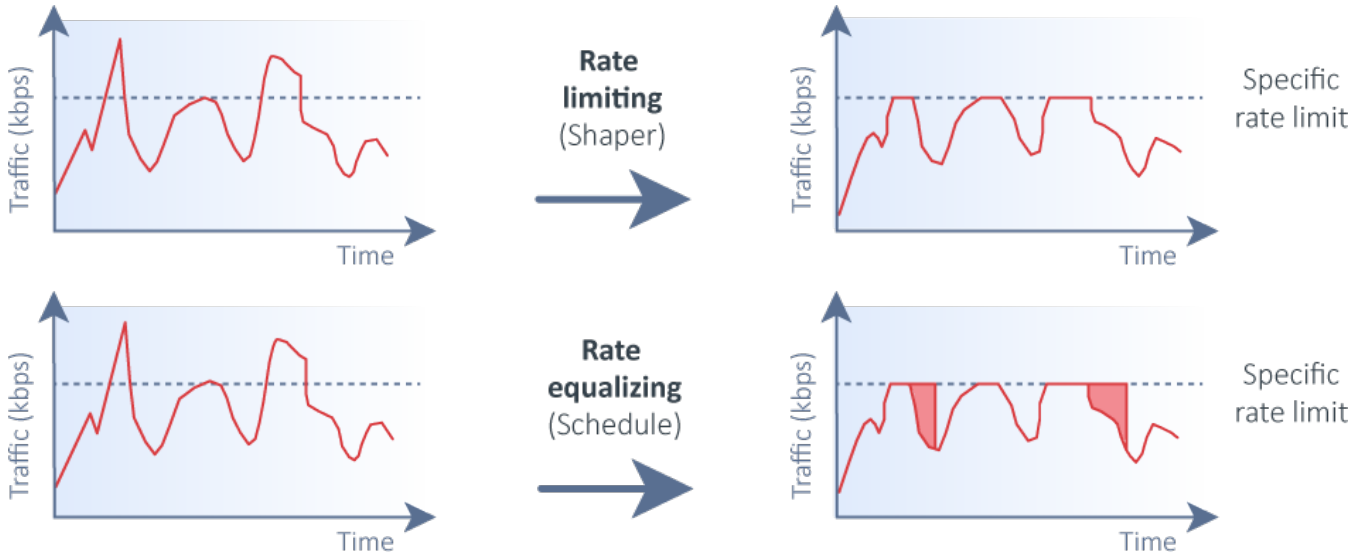


Figure 8.1. Principles of rate limiting and equalizing

As you can see in the first case all traffic exceeds a specific rate and is dropped. In another case, traffic exceeds a specific rate and is delayed in the queue and transmitted later when it is possible, but note that the packet can be delayed only until the queue is not full. If there is no more space in the queue buffer, packets are dropped.

For each queue we can define two rate limits:

- **CIR** (Committed Information Rate) – (**limit-at** in RouterOS) worst-case scenario, the flow will get this amount of traffic rate regardless of other traffic flows. At any given time, the bandwidth should not fall below this committed rate.
- **MIR** (Maximum Information Rate) – (**max-limit** in RouterOS) best-case scenario, the maximum available data rate for flow, if there is free any part of the bandwidth.

Simple Queue

```
/queue simple
```

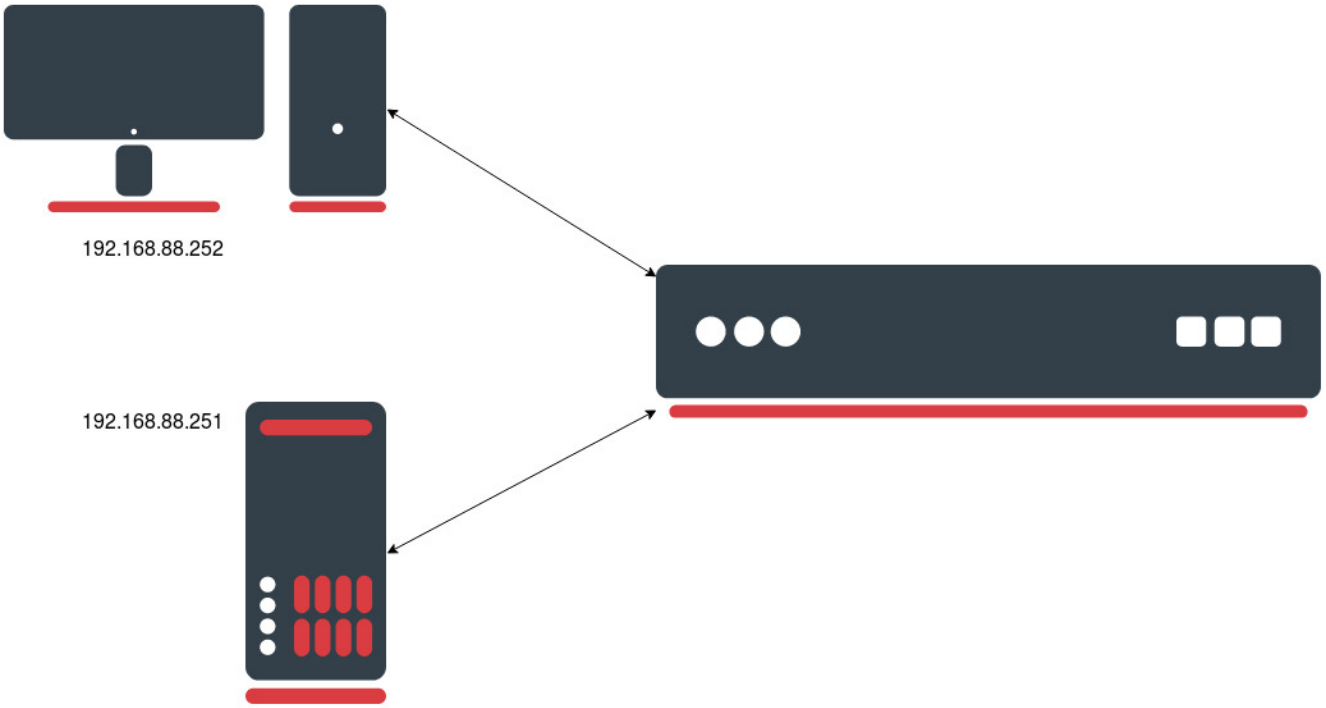
A simple queue is a plain way how to limit traffic for a particular target. Also, you can use simple queues to build advanced QoS applications. They have useful integrated features:

- peer-to-peer traffic queuing;
- applying queue rules on chosen time intervals;
- prioritization;
- using multiple packet marks from */ip firewall mangle*
- traffic shaping (scheduling) of bidirectional traffic (one limit for the total of upload + download)

⚠ Simple queues have a strict order - each packet must go through every queue until it reaches one queue which conditions fit packet parameters or until the end of the queues list is reached. For example, In the case of 1000 queues, a packet for the last queue will need to proceed through 999 queues before it will reach the destination.

Configuration example

In the following example, we have one SOHO device with two connected units PC and Server.



We have a 15 Mbps connection available from ISP in this case. We want to be sure the server receives enough traffic, so we will configure a simple queue with a *limit-at* parameter to guarantee a server to receive 5Mbps:

```
/queue simple
add limit-at=5M/5M max-limit=15M/15M name=queue1 target=192.168.88.251/32
```

That is all. The server will get 5 Mbps of traffic rate regardless of other traffic flows. If you are using the default configuration, be sure the FastTrack rule is disabled for this particular traffic, otherwise, it will bypass Simple Queues and they will not work.

Queue Tree

```
/queue tree
```

The queue tree creates only a one-directional queue in one of the HTBs. It is also the only way how to add a queue on a separate interface. This way it is possible to ease mangle configuration - you don't need separate marks for download and upload - only the upload will get to the Public interface and only the download will get to a Private interface. The main difference from Simple Queues is that the Queue tree is not ordered - all traffic passes it together.

Configuration example

In the following example, we will mark all the packets coming from preconfigured *in-interface-list=LAN* and will limit the traffic with a queue tree based on these packet marks.

Let`s create a firewall address-list:

```
[admin@MikroTik] > /ip firewall address-list
add address=www.youtube.com list=Youtube
[admin@MikroTik] > ip firewall address-list print
Flags: X - disabled, D - dynamic
 #   LIST
ADDRESS                                                                  CREATION-TIME
TIMEOUT
 0   Youtube                                                    www.youtube.
com                                                               oct/17/2019 14:47:11
 1 D ;;; www.youtube.com
     Youtube
216.58.211.14                                                            oct/17/2019 14:47:11
 2 D ;;; www.youtube.com
     Youtube
216.58.207.238                                                           oct/17/2019 14:47:11
 3 D ;;; www.youtube.com
     Youtube
216.58.207.206                                                           oct/17/2019 14:47:11
 4 D ;;; www.youtube.com
     Youtube
172.217.21.174                                                           oct/17/2019 14:47:11
 5 D ;;; www.youtube.com
     Youtube
216.58.211.142                                                           oct/17/2019 14:47:11
 6 D ;;; www.youtube.com
     Youtube
172.217.22.174                                                           oct/17/2019 14:47:21
 7 D ;;; www.youtube.com
     Youtube
172.217.21.142                                                           oct/17/2019 14:52:21
```

Mark packets with firewall mangle facility:

```
[admin@MikroTik] > /ip firewall mangle
add action=mark-packet chain=forward dst-address-list=Youtube in-interface-list=LAN new-packet-mark=pmark-
Youtube passthrough=yes
```

Configure the queue tree based on previously marked packets:

```
[admin@MikroTik] /queue tree
add max-limit=5M name=Limiting-Youtube packet-mark=pmark-Youtube parent=global
```

Check Queue tree stats to be sure traffic is matched:

```
[admin@MikroTik] > queue tree print stats
Flags: X - disabled, I - invalid
 0   name="Limiting-Youtube" parent=global packet-mark=pmark-Youtube rate=0 packet-rate=0 queued-bytes=0 queued-
packets=0 bytes=67887 packets=355 dropped=0
```

Queue Types

```
/queue type
```

This sub-menu list by default created queue types and allows to add of new user-specific ones.

By default RouterOS creates the following pre-defined queue types:

```
[admin@MikroTik] > /queue type print
Flags: * - default
 0 * name="default" kind=pfifo pfifo-limit=50

 1 * name="ethernet-default" kind=pfifo pfifo-limit=50

 2 * name="wireless-default" kind=sfq sfq-perturb=5 sfq-allot=1514

 3 * name="synchronous-default" kind=red red-limit=60 red-min-threshold=10 red-max-threshold=50 red-burst=20
red-avg-packet=1000

 4 * name="hotspot-default" kind=sfq sfq-perturb=5 sfq-allot=1514

 5 * name="pcq-upload-default" kind=pcq pcq-rate=0 pcq-limit=50KiB pcq-classifier=src-address pcq-total-
limit=2000KiB pcq-burst-rate=0 pcq-burst-threshold=0 pcq-burst-time=10s pcq-src-address-mask=32
     pcq-dst-address-mask=32 pcq-src-address6-mask=128 pcq-dst-address6-mask=128

 6 * name="pcq-download-default" kind=pcq pcq-rate=0 pcq-limit=50KiB pcq-classifier=dst-address pcq-total-
limit=2000KiB pcq-burst-rate=0 pcq-burst-threshold=0 pcq-burst-time=10s pcq-src-address-mask=32
     pcq-dst-address-mask=32 pcq-src-address6-mask=128 pcq-dst-address6-mask=128

 7 * name="only-hardware-queue" kind=none

 8 * name="multi-queue-ethernet-default" kind=mq-pfifo mq-pfifo-limit=50

 9 * name="default-small" kind=pfifo pfifo-limit=10
```

All RouterBOARDS have default queue type "**only-hardware-queue**" with "kind=none". "only-hardware-queue" leaves interface with only hardware transmit descriptor ring buffer which acts as a queue in itself. Usually, at least 100 packets can be queued for transmit in transmit descriptor ring buffer. Transmit descriptor ring buffer size and the number of packets that can be queued in it varies for different types of ethernet MACs. Having no software queue is especially beneficial on SMP systems because it removes the requirement to synchronize access to it from different CPUs/cores which is resource-intensive. Having the possibility to set "only-hardware-queue" requires support in an ethernet driver so it is available only for some ethernet interfaces mostly found on RouterBOARDs.

A **"multi-queue-ethernet-default"** can be beneficial on SMP systems with ethernet interfaces that have support for multiple transmit queues and have a Linux driver support for multiple transmit queues. By having one software queue for each hardware queue there might be less time spent on synchronizing access to them.

> ⚠ Improvement from only-hardware-queue and multi-queue-ethernet-default is present only when there is no "/queue tree" entry with a particular interface as a parent.

Kinds

Queue kinds are packet processing algorithms. Kind describe which packet will be transmitted next in the line. RouterOS supports the following Queueing kinds:

- FIFO (BFIFO, PFIFO, MQ PFIFO)
- RED
- SFQ
- PCQ

FIFO

These kinds are based on the FIFO algorithm (First-In-First-Out). The difference between **PFIFO** and **BFIFO** is that one is measured in packets and the other one in bytes. These queues use **pfifo-limit** and **bfifo-limit** parameters.

Every packet that cannot be enqueued (if the queue is full), is dropped. Large queue sizes can increase latency but utilize the channel better.

MQ-PFIFO is *pfifo* with support for multiple transmit queues. This queue is beneficial on SMP systems with ethernet interfaces that have support for multiple transmit queues and have a Linux driver support for multiple transmit queues (mostly on x86 platforms). This kind uses the **mq-pfifo-limit** parameter.

RED

Random Early Drop is a queuing mechanism that tries to avoid network congestion by controlling the average queue size. The average queue size is compared to two thresholds: a minimum (min_{th}) and maximum (max_{th}) threshold. If the average queue size (avg_q) is less than the minimum threshold, no packets are dropped. When the average queue size is greater than the maximum threshold, all incoming packets are dropped. But if the average queue size is between the minimum and maximum thresholds packets are randomly dropped with probability P_d where probability is exact a function of the average queue size: $P_d = P_{max}(avg_q – min_{th})/ (max_{th} - min_{th})$. If the average queue grows, the probability of dropping incoming packets grows too. P_{max} - ratio, which can adjust the packet discarding probability abruptness, (the simplest case P_{max} can be equal to one. The 8.2 diagram shows the packet drop probability in the RED algorithm.

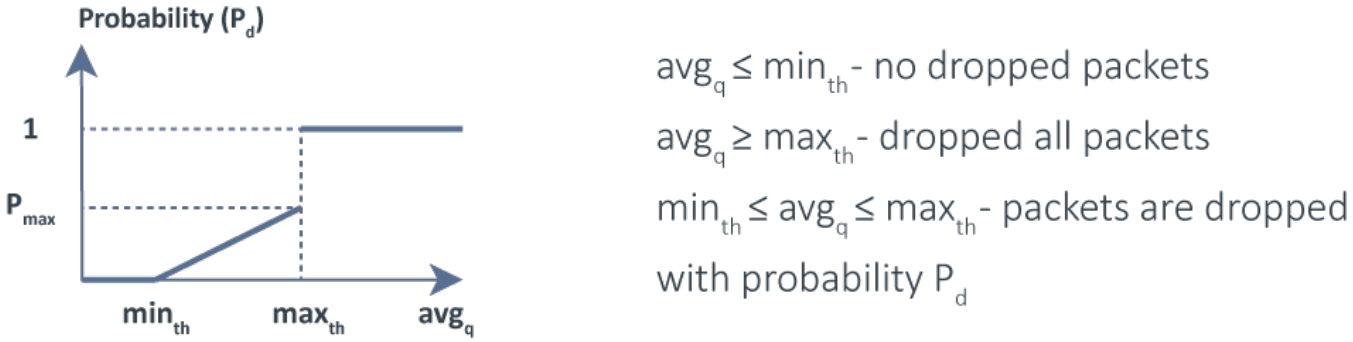


Figure 8.2. RED operation

SFQ

Stochastic Fairness Queuing (SFQ) is ensured by hashing and round-robin algorithms. SFQ is called "Stochastic" because it does not really allocate a queue for each flow, it has an algorithm that divides traffic over a limited number of queues (1024) using a hashing algorithm.

Traffic flow may be uniquely identified by 4 options (*src-address, dst-address, src-port,* and *dst-port*), so these parameters are used by the SFQ hashing algorithm to classify packets into one of 1024 possible sub-streams. Then round-robin algorithm will start to distribute available bandwidth to all sub-streams, on each round giving **sfq-allot** bytes of traffic. The whole SFQ queue can contain 128 packets and there are 1024 sub-streams available. The 8.3 diagram shows the SFQ operation:

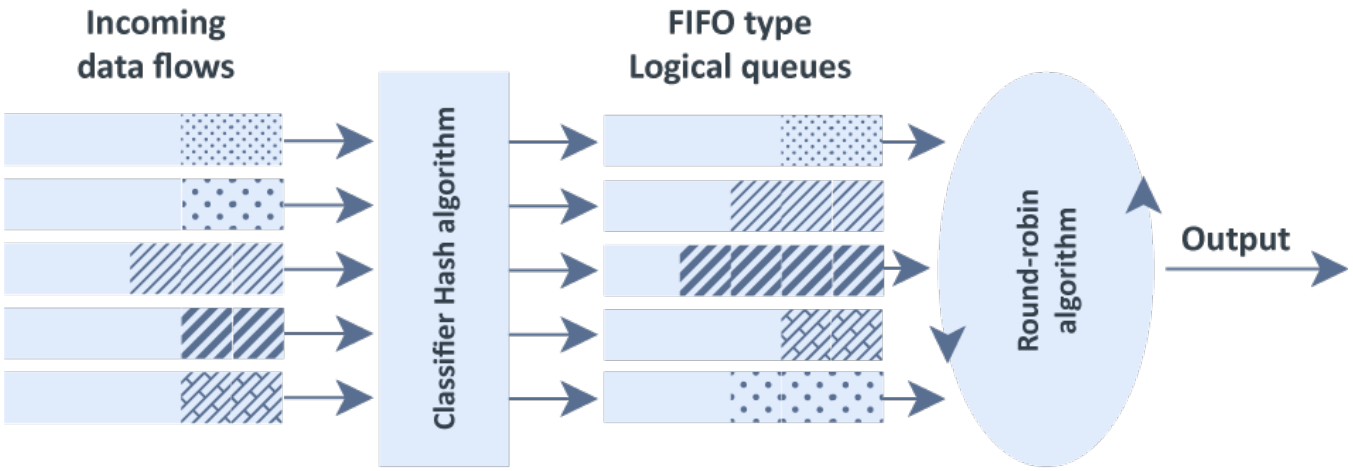


Figure 8.3. SFQ operation

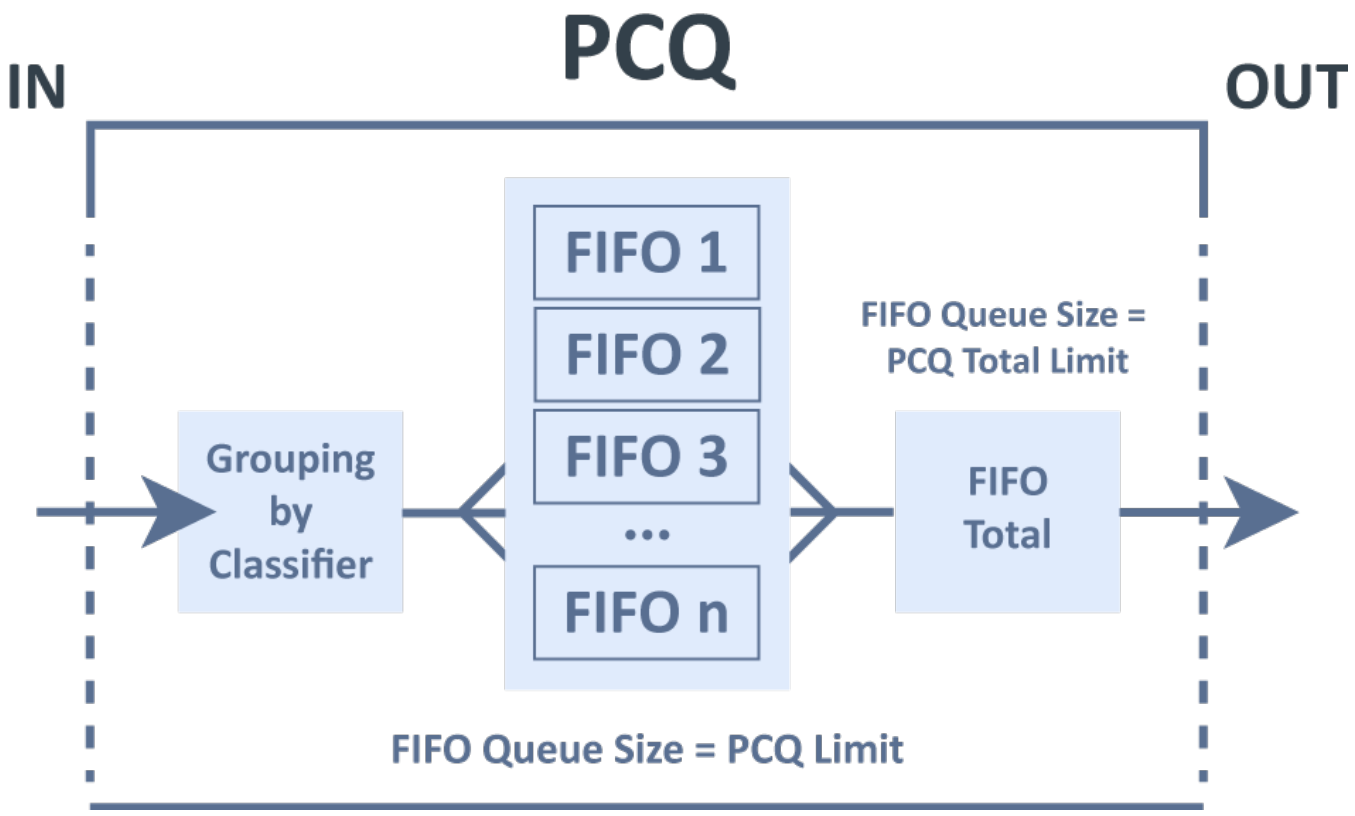
PCQ

PCQ algorithm is very simple - at first, it uses selected classifiers to distinguish one sub-stream from another, then applies individual FIFO queue size and limitation on every sub-stream, then groups all sub-streams together and applies global queue size and limitation.

PCQ parameters:

- **pcq-classifier** (dst-address | dst-port | src-address | src-port; default: "") : selection of sub-stream identifiers
- **pcq-rate** (number): maximal available data rate of each sub-steam
- **pcq-limit** (number): queue size of single sub-stream (in KiB)
- **pcq-total-limit** (number): maximum amount of queued data in all sub-streams (in KiB)

It is possible to assign a speed limitation to sub-streams with the **pcq-rate** option. If "pcq-rate=0" sub-streams will divide available traffic equally.



For example, instead of having 100 queues with 1000kbps limitation for download, we can have one PCQ queue with 100 sub-streams

PCQ has burst implementation identical to Simple Queues and Queue Tree:

- **pcq-burst-rate** (number): maximal upload/download data rate which can be reached while the burst for substream is allowed
- **pcq-burst-threshold** (number): this is the value of burst on/off switch
- **pcq-burst-time** (time): a period of time (in seconds) over which the average data rate is calculated. (This is NOT the time of actual burst)

PCQ also allows using different size IPv4 and IPv6 networks as sub-stream identifiers. Before it was locked to a single IP address. This is done mainly for IPv6 as customers from an ISP point of view will be represented by /64 network, but devices in customers network will be /128. PCQ can be used for both of these scenarios and more. PCQ parameters:

- **pcq-dst-address-mask** (number): the size of the IPv4 network that will be used as a dst-address sub-stream identifier
- **pcq-src-address-mask** (number): the size of the IPv4 network that will be used as an src-address sub-stream identifier
- **pcq-dst-address6-mask** (number): the size of the IPV6 network that will be used as a dst-address sub-stream identifier
- **pcq-src-address6-mask** (number): the size of the IPV6 network that will be used as an src-address sub-stream identifier

⚠ The following queue kinds CoDel, FQ-Codel, and CAKE available since RouterOS version 7.1beta3.

CoDel

CoDel (Controlled-Delay Active Queue Management) algorithm uses the local minimum queue as a measure of the persistent queue, similarly, it uses a minimum delay parameter as a measure of the standing queue delay. Queue size is calculated using packet residence time in the queue.

Properties

Property	Description
codel-ce-threshold (*default*:)	Marks packets above a configured threshold with ECN.
codel-ecn (*default*: **no**)	An option is used to mark packets instead of dropping them.
codel-interval (*default*: **100ms**)	Interval should be set on the order of the worst-case RTT through the bottleneck giving endpoints sufficient time to react.
codel-limit (*default*: **1000**)	Queue limit, when the limit is reached, incoming packets are dropped.
codel-target (*default*: **5ms**)	Represents an acceptable minimum persistent queue delay.

FQ-Codel

CoDel - Fair Queuing (FQ) with Controlled Delay (CoDel) uses a randomly determined model to classify incoming packets into different flows and is used to provide a fair share of the bandwidth to all the flows using the queue. Each flow is managed using CoDel queuing discipline which internally uses a FIFO algorithm.

Properties

Property	Description
fq-codel-ce-threshold (*default*:)	Marks packets above a configured threshold with ECN.
fq-codel-ecn (*default*: **yes**)	An option is used to mark packets instead of dropping them.
fq-codel-flows (default: **1024**)	A number of flows into which the incoming packets are classified.
fq-codel-interval (*default*: **100ms**)	Interval should be set on the order of the worst-case RTT through the bottleneck giving endpoints sufficient time to react.
fq-codel-limit (*default*: **10240**)	Queue limit, when the limit is reached, incoming packets are dropped.
fq-codel-memlimit (default: **32.0MiB**)	A total number of bytes that can be queued in this FQ-CoDel instance. Will be enforced from the *fq-codel-limit* parameter.
fq-codel-quantum (*default*: **1514**)	A number of bytes used as 'deficit' in the fair queuing algorithm. Default (1514 bytes) corresponds to the Ethernet MTU plus the hardware header length of 14 bytes.
fq-codel-target (*default*: **5ms**)	Represents an acceptable minimum persistent queue delay.

CAKE

CAKE - Common Applications Kept Enhanced (CAKE) implemented as a *queue discipline* (qdisc) for the Linux kernel uses COBALT (AQM algorithm combining Codel and BLUE) and a variant of DRR++ for flow isolation. In other words, Cake's fundamental design goal is user-friendliness. All settings are optional; the default settings are chosen to be practical in most common deployments. In most cases, the configuration requires only a bandwidth parameter to get useful results,

Properties

Property	Description
cake-ack-filter *(default:* **none**)	
cake-atm *(default:*)	Compensates for ATM cell framing, which is normally found on ADSL links.
cake-autorate-ingress *(yes/no, default:*)	Automatic capacity estimation based on traffic arriving at this qdisc. This is most likely to be useful with cellular links, which tend to change quality randomly. The Bandwidth Limit parameter can be used in conjunction to specify an initial estimate. The shaper will periodically be set to a bandwidth slightly below the estimated rate. This estimator cannot estimate the bandwidth of links downstream of itself.

cake-bandwidth *(default:*)	Sets the shaper bandwidth.
cake-diffserv *(default:* **diffserv3**)	CAKE can divide traffic into "tins" based on the Diffserv field: • **diffserv4** Provides a general-purpose Diffserv implementation with four tins: Bulk (CS1), 6.25% threshold, generally low priority. Best Effort (general), 100% threshold. Video (AF4x, AF3x, CS3, AF2x, CS2, TOS4, TOS1), 50% threshold. Voice (CS7, CS6, EF, VA, CS5, CS4), 25% threshold. • **diffserv3** (default) Provides a simple, general-purpose Diffserv implementation with three tins: Bulk (CS1), 6.25% threshold, generally low priority. Best Effort (general), 100% threshold. Voice (CS7, CS6, EF, VA, TOS4), 25% threshold, reduced Codel interval.
cake-flowmode *(dsthost/dual-dsthost/dual-srchost/flowblind /flows/hosts/srchost/triple-isolate, default:* **triple-isolate**)	• **flowblind** - Disables flow isolation; all traffic passes through a single queue for each tin. • **srchost** - Flows are defined only by source address. • **dsthost** Flows are defined only by destination address. • **hosts** - Flows are defined by source-destination host pairs. This is host isolation, rather than flow isolation. • **flows** - Flows are defined by the entire 5-tuple of source address, a destination address, transport protocol, source port,and destination port. This is the type of flow isolation performed by SFQ and fq_codel. • **dual-srchost** Flows are defined by the 5-tuple, and fairness is applied first over source addresses, then over individual flows. Good for use on egress traffic from a LAN to the internet, where it'll prevent anyone LAN host from monopolizing the uplink, regardless of the number of flows they use. • **dual-dsthost** Flows are defined by the 5-tuple, and fairness is applied first over destination addresses, then over individual flows. Good for use on ingress traffic to a LAN from the internet, where it'll prevent anyone LAN host from monopolizing the downlink, regardless of the number of flows they use. • **triple-isolate** - Flows are defined by the 5-tuple, and fairness is applied over source *and* destination addresses intelligently (ie. not merely by host-pairs), and also over individual flows. • **nat** Instructs Cake to perform a NAT lookup before applying flow- isolation rules, to determine the true addresses and port numbers of the packet, to improve fairness between hosts "inside" the NAT. This has no practical effect in "flowblind" or "flows" modes, or if NAT is performed on a different host. • **nonat** (default) The cake will not perform a NAT lookup. Flow isolation will be performed using the addresses and port numbers directly visible to the interface Cake is attached to.
cake-memlimit *(default:*)	Limit the memory consumed by Cake to LIMIT bytes. By default, the limit is calculated based on the bandwidth and RTT settings.
cake-mpu *(-64 ... 256, default:*)	Rounds each packet (including overhead) up to a minimum length BYTES.
cake-nat *(default:* **no)**	Instructs Cake to perform a NAT lookup before applying a flow-isolation rule.
cake-overhead *(-64 ... 256, default:*)	Adds BYTES to the size of each packet. BYTES may be negative.
cake-overhead-scheme *(default:*)	
cake-rtt *(default:* **100ms**)	Manually specify an RTT. Default 100ms is suitable for most Internet traffic.
cake-rtt-scheme *(datacentre /internet/interplanetary/lan /metro/none/oceanic/regional /satellite, default:*)	• **datacentre** - For extremely high-performance 10GigE+ networks only. Equivalent to **RTT 100us.** • **lan** - For pure Ethernet (not Wi-Fi) networks, at home or in the office. Don't use this when shaping for an Internet access link. Equivalent to **RTT 1ms.** • **metro** - For traffic mostly within a single city. Equivalent to **RTT 10ms. regional** For traffic mostly within a European-sized country. Equivalent to **RTT 30ms.** • **internet** (default) This is suitable for most Internet traffic. Equivalent to **RTT 100ms.** • **oceanic** - For Internet traffic with generally above-average latency, such as that suffered by Australasian residents. Equivalent to **RTT 300ms.** • **satellite** - For traffic via geostationary satellites. Equivalent to **RTT 1000ms.** • **interplanetary** - So named because Jupiter is about 1 light-hour from Earth. Use this to (almost) completely disable AQM actions. Equivalent to **RTT 3600s.**
cake-wash *(default:* **no**)	Apply the wash option to clear all extra DiffServ (but not ECN bits), after priority queuing has taken place.

Interface Queue

```
/queue interface
```

Before sending data over an interface, it is processed by the queue. This sub-menu lists all available interfaces in RouterOS and allows to change queue type for a particular interface. The list is generated automatically.

```
[admin@MikroTik] > queue interface print
Columns: INTERFACE, QUEUE, ACTIVE-QUEUE
# INTERFACE QUEUE ACTIVE-QUEUE
0 ether1 only-hardware-queue only-hardware-queue
1 ether2 only-hardware-queue only-hardware-queue
2 ether3 only-hardware-queue only-hardware-queue
3 ether4 only-hardware-queue only-hardware-queue
4 ether5 only-hardware-queue only-hardware-queue
5 ether6 only-hardware-queue only-hardware-queue
6 ether7 only-hardware-queue only-hardware-queue
7 ether8 only-hardware-queue only-hardware-queue
8 ether9 only-hardware-queue only-hardware-queue
9 ether10 only-hardware-queue only-hardware-queue
10 sfp-sfpplus1 only-hardware-queue only-hardware-queue
11 wlan1 wireless-default wireless-default
12 wlan2 wireless-default wireless-default
```

Queue load visualization in GUI

In Winbox and Webfig, a green, yellow, or red icon visualizes each Simple and Tree queue usage based on max-limit.

	0% - 50% of max-limit used
	50% - 75% of max-limit used
	75% - 100% of max-limit used

HTB (Hierarchical Token Bucket)

Introduction

HTB (Hierarchical Token Bucket) is a classful queuing method that is useful for handling different kinds of traffic. This article will concentrate on the "Token Bucket" part of **Hierarchical Token Bucket**(HTB) - an algorithm inside the single queue and configuration examples.

Token Bucket algorithm (Red part of the diagram)

The Token Bucket algorithm is based on an analogy to a bucket where tokens, represented in bytes, are added at a specific rate. The bucket itself has a specified capacity.

If the bucket fills to capacity, newly arriving tokens are dropped

Bucket capacity = bucket-size * max-limit

- **bucket size** (0..10, Default:0.1) - queue option was added in RouterOS v6.35, before that it was hard-coded to a value of "0.1".

Before allowing any packet to pass through the queue, the queue bucket is inspected to see if it already contains sufficient tokens at that moment.

If yes, the appropriate number of tokens are removed ("cashed in") and the packet is permitted to pass through the queue.

If not, the packets stay at the start of the packet waiting queue until the appropriate amount of tokens is available.

In the case of a multi-level queue structure, tokens used in a child queue are also 'charged' to their parent queues. In other words - child queues 'borrow' tokens from their parent queues.

Packet queue (Blue part of the diagram)

The size of this packet queue, the sequence, how packets are added to this queue, and when packets are discarded is determined by:

- **queue-type** - Queue
- **queue-size** - Queue Size

Token rate selection (Black part of the diagram)

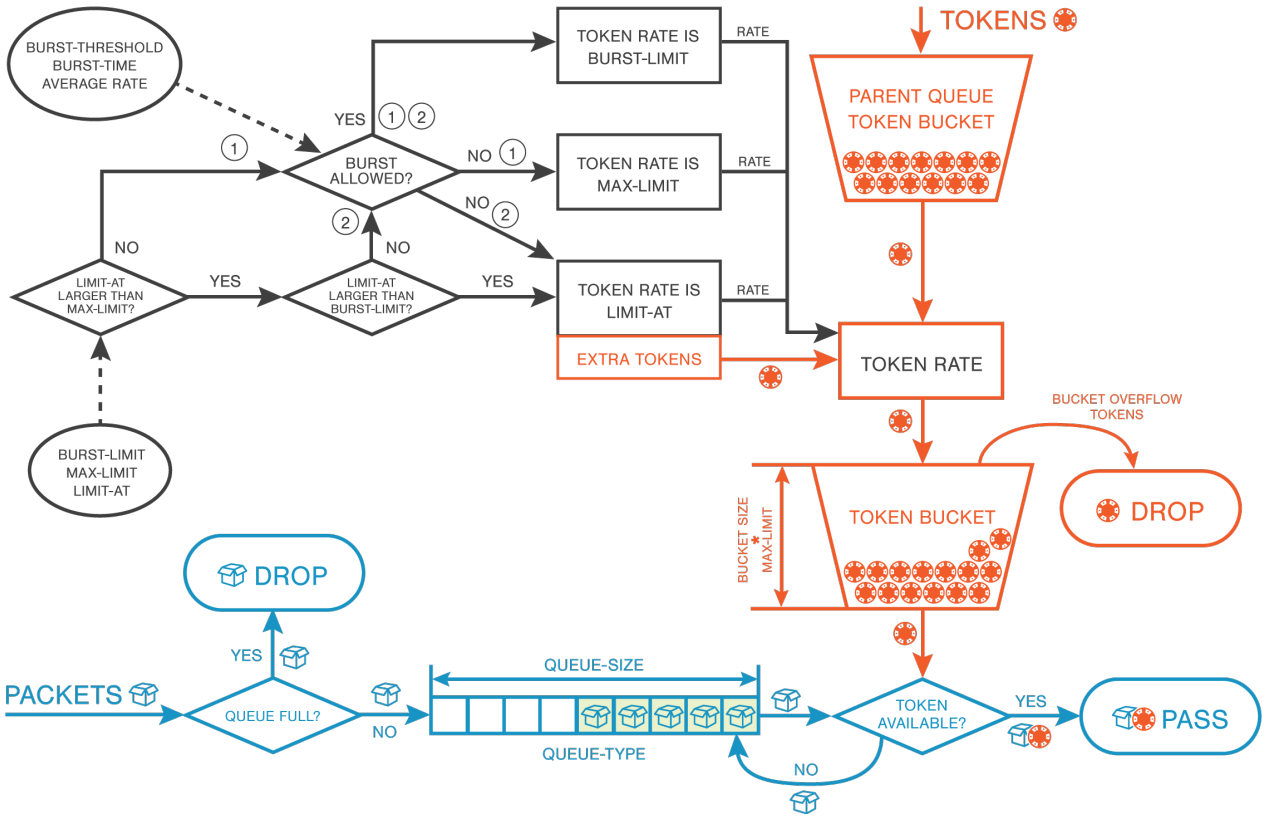
The maximal token rate at any given time is equal to the highest activity of these values:

- **limit-at** (*NUMBER/NUMBER*): guaranteed upload/download data rate to a target
- **max-limit** (*NUMBER/NUMBER*): maximal upload/download data rate that is allowed for a target
- **burst-limit** (*NUMBER/NUMBER*): maximal upload/download data rate that is allowed for a target while the 'burst' is active

burst-limit is active only when 'burst' is in the allowed state - more info here: Queue Burst

In a case where **limit-at** is the highest value, extra tokens need to be issued to compensate for all missing tokens that were not borrowed from its parent queue.

The Diagram



Bucket Size in action

Let's have a simple setup where all traffic from and to one IP address is marked with a packet-mark:

```
/ip firewall mangle
add chain=forward action=mark-connection connection-mark=no-mark src-address=192.168.88.101 new-connection-
mark=pc1_conn
add chain=forward action=mark-connection connection-mark=no-mark dst-address=192.168.88.101 new-connection-
mark=pc1_conn
add chain=forward action=mark-packet connection-mark=pc1_conn new-packet-mark=pc1_traffic
```

Default Queue Bucket

```
/queue tree
add name=download parent=Local packet-mark=PC1-traffic max-limit=10M
add name=upload parent=Public packet-mark=PC1-traffic max-limit=10M
```

In this case bucket-size=0.1, so bucket-capacity= 0.1 x 10M = 1M

If the bucket is full (that is, the client was not using the full capacity of the queue for some time), the next 1Mb of traffic can pass through the queue at an unrestricted speed.

Large Queue Bucket

```
/queue tree
add name=download parent=Local packet-mark=PC1-traffic max-limit=10M bucket-size=10
add name=upload parent=Public packet-mark=PC1-traffic max-limit=10M bucket-size=10
```

Let's try to apply the same logic to a situation when bucket size is at its maximal value:

In this case bucket-size=10, so bucket-capacity= 10 x 10M = 100M

If the bucket is full (that is, the client was not using the full capacity of the queue for some time), the next 100Mb of traffic can pass through the queue at an unrestricted speed.

So you can have:

- 20Mbps transfer speed for 10s
- 60Mbps transfer burst for 2s
- 1Gbps transfer burst for approximately 100ms

You can therefore see that the bucket permits a type of 'burstiness' of the traffic that passes through the queue. The behavior is similar to the normal burst feature but lacks the upper limit of the burst. This setback can be avoided if we utilize bucket size in the queue structure:

Large Child Queue Bucket, Small Parent Queue Bucket

```
/queue tree
add name=download_parent parent=Local max-limit=20M
add name=download parent=download_parent packet-mark=PC1-traffic max-limit=10M bucket-size=10
add name=upload_parent parent=Public max-limit=20M
add name=upload parent=upload_parent packet-mark=PC1-traffic max-limit=10M bucket-size=10
```

In this case:

- parent queue bucket-size=0.1, bucket-capacity= 0.1 x 20M = 2M
- child queue bucket-size=10, bucket-capacity= 10 x 10M = 100M

The parent will run out of tokens much faster than the child queue and as its child queue always borrows tokens from the parent queue the whole system is restricted to token-rate of the parent queue - in this case to max-limit=20M. This rate will be sustained until the child queue runs out of tokens and will be restricted to its token rate of 10Mbps.

In this way, we can have a burst at 20Mbps for up to 10 seconds.

Configuration

We have to follow three basic steps to create HTB:

- **Match and mark traffic** – classify traffic for further use. Consists of one or more matching parameters to select packets for the specific class;
- **Create rules (policy) to mark traffic** – put specific traffic classes into specific queues and define the actions that are taken for each class;
- **Attach a policy for specific interface(-s)** – append policy for all interfaces (global-in, global-out, or global-total), for a specific interface, or for a specific parent queue;

HTB allows to create of a hierarchical queue structure and determines relations between queues, like "parent-child" or "child-child".

As soon as the queue has at least one child it becomes an **inner** queue, all queues without children - are **leaf** queues. **Leaf** queues make actual traffic consumption, **Inner** queues are responsible only for traffic distribution. All **leaf** queues are treated on an equal basis.

In RouterOS, it is necessary to specify the **parent** option to assign a queue as a child to another queue.

Dual Limitation

Each queue in HTB has two rate limits:

- **CIR** (Committed Information Rate) – (**limit-at** in RouterOS) worst case scenario, the flow will get this amount of traffic no matter what (assuming we can actually send so much data);
- **MIR** (Maximal Information Rate) – (**max-limit** in RouterOS) best case scenario, a rate that flow can get up to if their queue's parent has spare bandwidth;

In other words, at first **limit-at** (**CIR**) of all queues will be satisfied, only then child queues will try to borrow the necessary data rate from their parents in order to reach their **max-limit** (**MIR**).

> ⚠ **CIR** will be assigned to the corresponding queue no matter what. (even if max-limit of the parent is exceeded)

That is why, to ensure optimal (as designed) usage of the dual limitation feature, we suggest sticking to these rules:

- The Sum of committed rates of all children must be less or equal to the amount of traffic that is available to parents;

CIR(parent)* ≥ CIR(child1) +...+ CIR(childN)*in case if parent is main parent CIR(parent)=MIR(parent)

- The maximal rate of any child must be less or equal to the maximal rate of the parent

MIR (parent) ≥ MIR(child1) & MIR (parent) ≥ MIR(child2) & ... & MIR (parent) ≥ MIR(childN)

Queue colors in Winbox:

- 0% - 50% available traffic used - green
- 51% - 75% available traffic used - yellow
- 76% - 100% available traffic used - red

Priority

We already know that **limit-at** (**CIR**) to all queues will be given out no matter what.

Priority is responsible for the distribution of remaining parent queues traffic to child queues so that they are able to reach **max-limit**

The queue with higher priority will reach its **max-limit** before the queue with lower priority. 8 is the lowest priority, and 1 is the highest.

Make a note that priority only works:

- for **leaf** queues - priority in the **inner** queue has no meaning.
- if **max-limit** is specified (not 0)

Examples

In this section, we will analyze HTB in action. To do that we will take one HTB structure and will try to cover all the possible situations and features, by changing the amount of incoming traffic that HTB has to recycle. and changing some options.

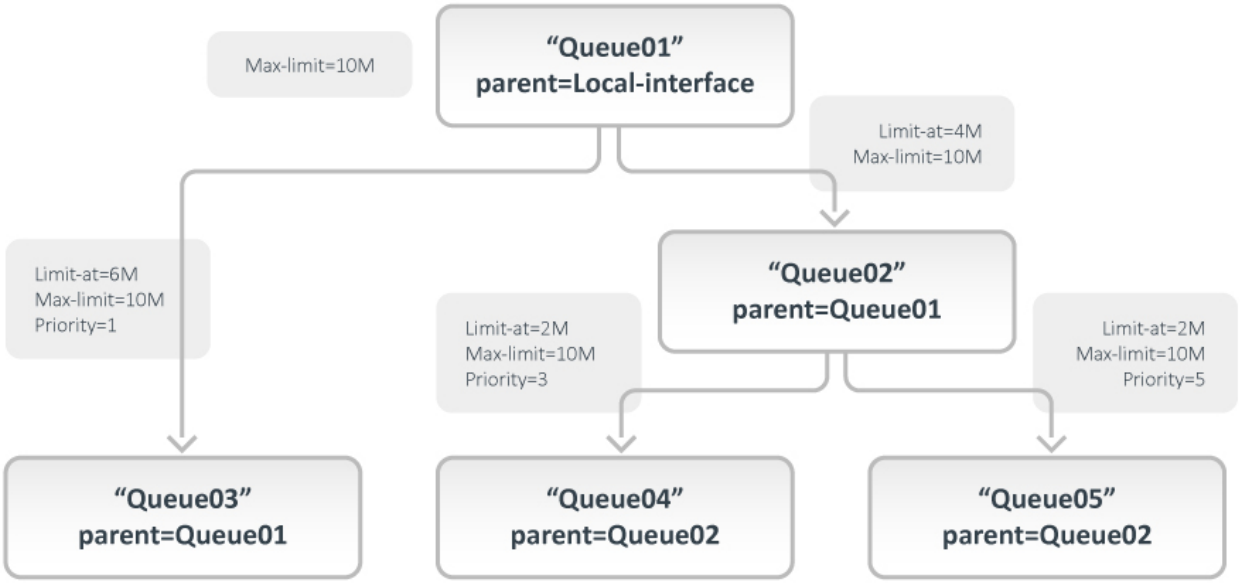
Structure

Our HTB structure will consist of 5 queues:

- **Queue01** inner queue with two children - **Queue02** and **Queue03**
- **Queue02** inner queue with two children - **Queue04** and **Queue05**
- **Queue03** leaf queue
- **Queue04** leaf queue
- **Queue05** leaf queue

Queue03, **Queue04,** and **Queue05** are clients who require 10Mbps all the time Outgoing interface is able to handle 10Mbps of traffic.

Example 1: Usual case

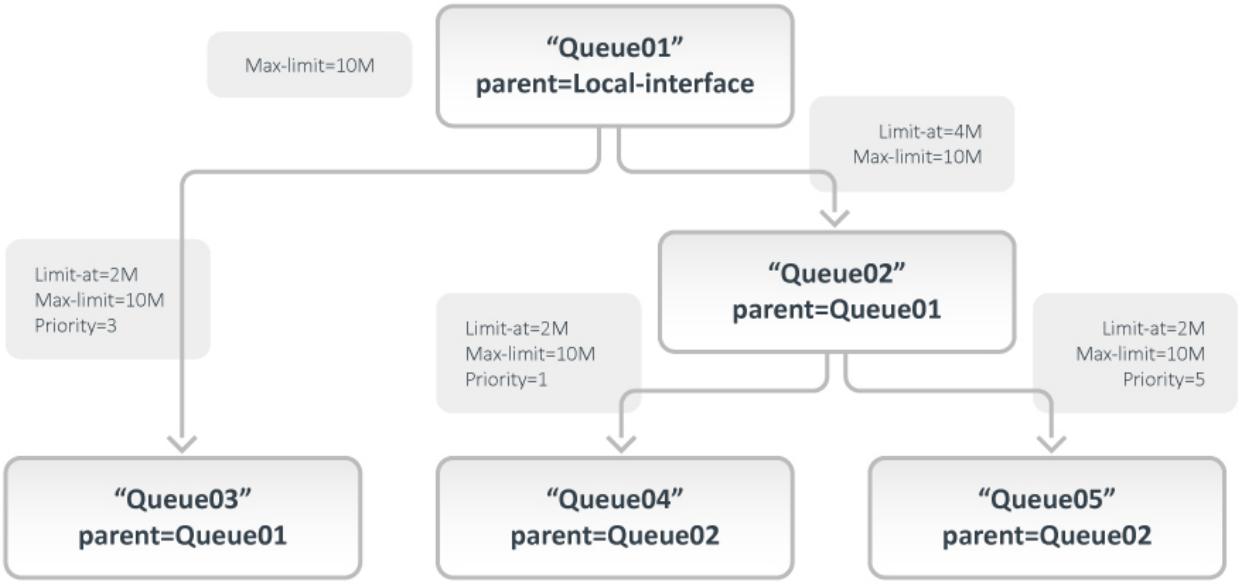


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=6Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 1

- **Queue03** will receive 6Mbps
- **Queue04** will receive 2Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** HTB was built in a way, that, by satisfying all **limit-at**s, the main queue no longer has throughput to distribute.

Example 2: Usual case with max-limit

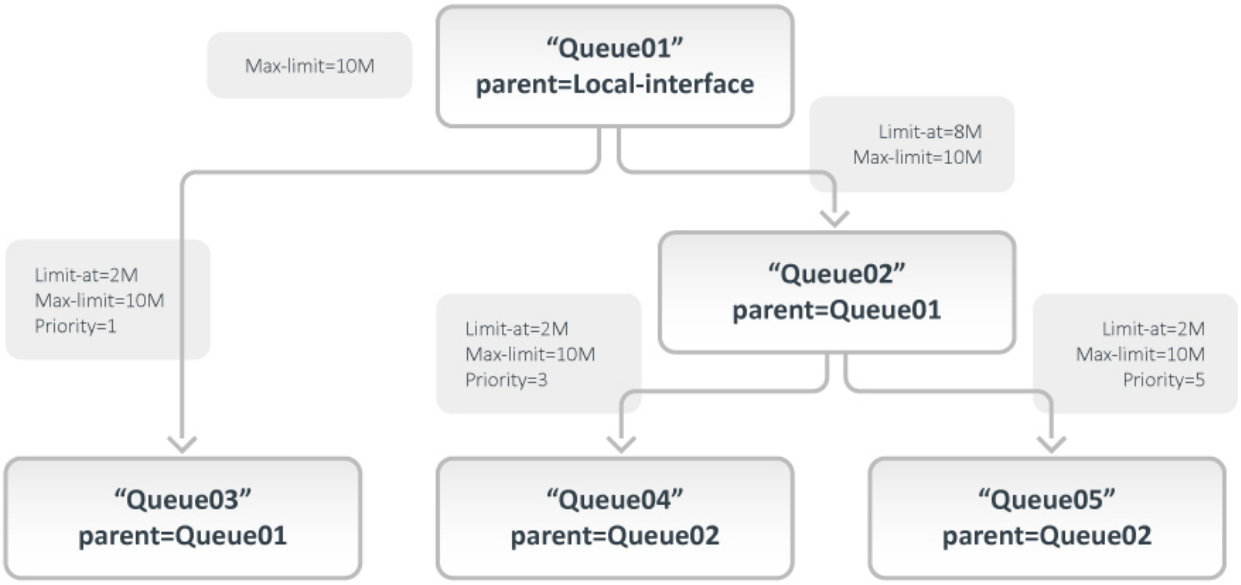


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=1
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 2

- **Queue03** will receive 2Mbps
- **Queue04** will receive 6Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** After satisfying all **limit-at**s HTB will give throughput to queue with the highest priority.

Example 3: Inner queue limit-at

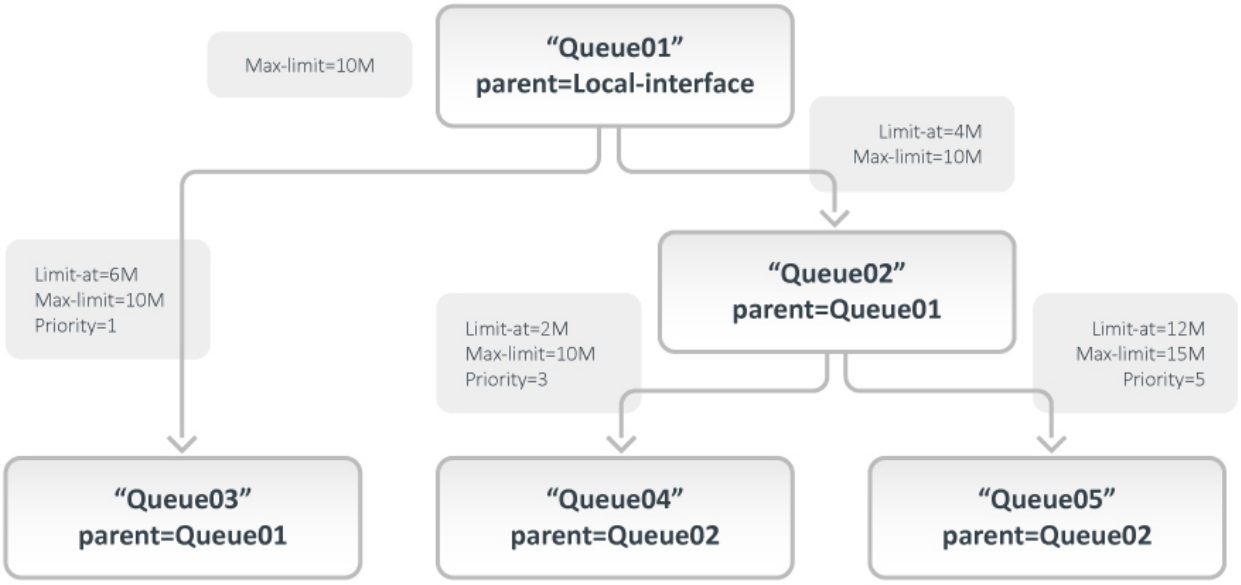


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=8Mbps max-limit=10Mbps
- **Queue03** limit-at=2Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 3

- **Queue03** will receive 2Mbps
- **Queue04** will receive 6Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** After satisfying all **limit-at**s HTB will give throughput to queue with the highest priority. But in this case, **inner** queue **Queue02** had **limit-at** specified, by doing so, it reserved 8Mbps of throughput for queues **Queue04** and **Queue05**. Of these two **Queue04** have the highest priority, that is why it gets additional throughput.

Example 4: Leaf queue limit-at



- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=6Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=12Mbps max-limit=15Mbps priority=5

Result of Example 4

- **Queue03** will receive ~3Mbps
- **Queue04** will receive ~1Mbps
- **Queue05** will receive ~6Mbps
- **Clarification:** Only by satisfying all **limit-at**s HTB was forced to allocate 20Mbps - 6Mbps to **Queue03**, 2Mbps to **Queue04**, and 12Mbps to **Queue05** , but our output interface is able to handle 10Mbps. As the output interface queue is usually FIFO throughput allocation will keep the ratio 6:2:12 or 3:1:6

Queue size

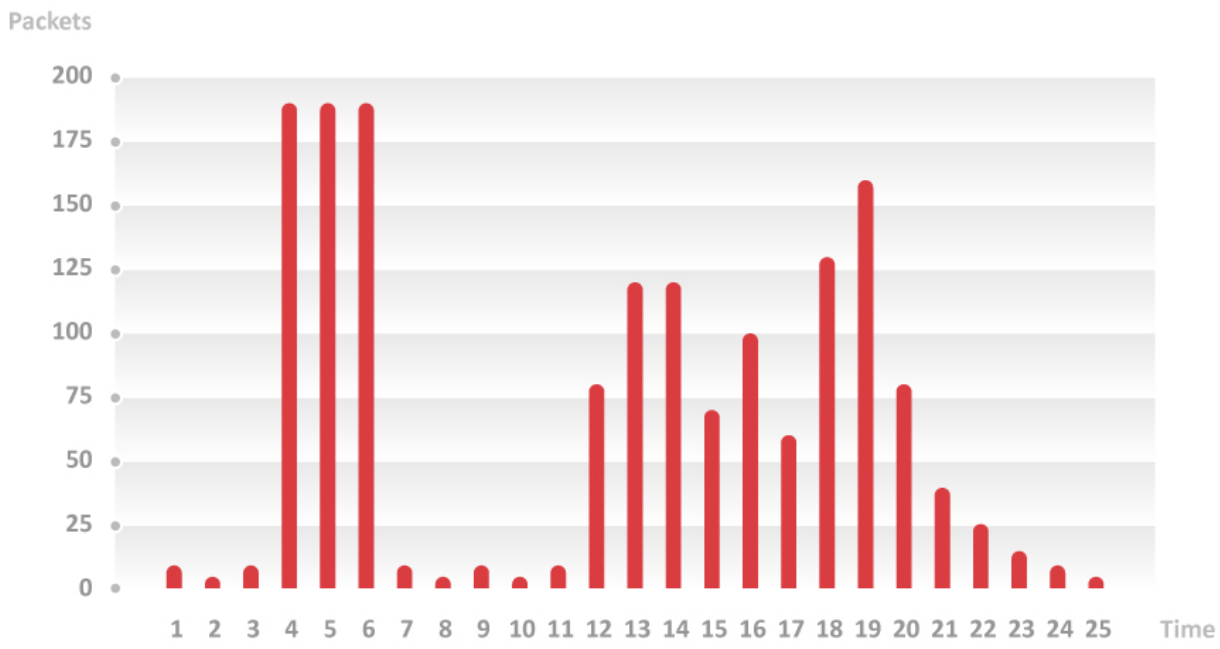
Example

This example was created to highlight the queue size impact on traffic that was queued by a specific queue.

In Mikrotik RouterOS, queue size can be specified in the "/queue type" menu. Each queue type has a different option for specifying queue size (pfifo-limit, bfifo-limit, pcq-limit, pcq-total-limit, red-limit), but all principles are the same - queue size is the main option that decides should the package be dropped or scheduled for a later time.

In a real-time environment, this process is happening continuously without any stops, steps, or other interruptions, but in order to show it as an example, we will divide it into steps, where it is possible to know exactly how many packets will be received/transited in every step.

We will not go into specific details of TCP and dropped packet retransmission - consider these packets as simple UDP streams.

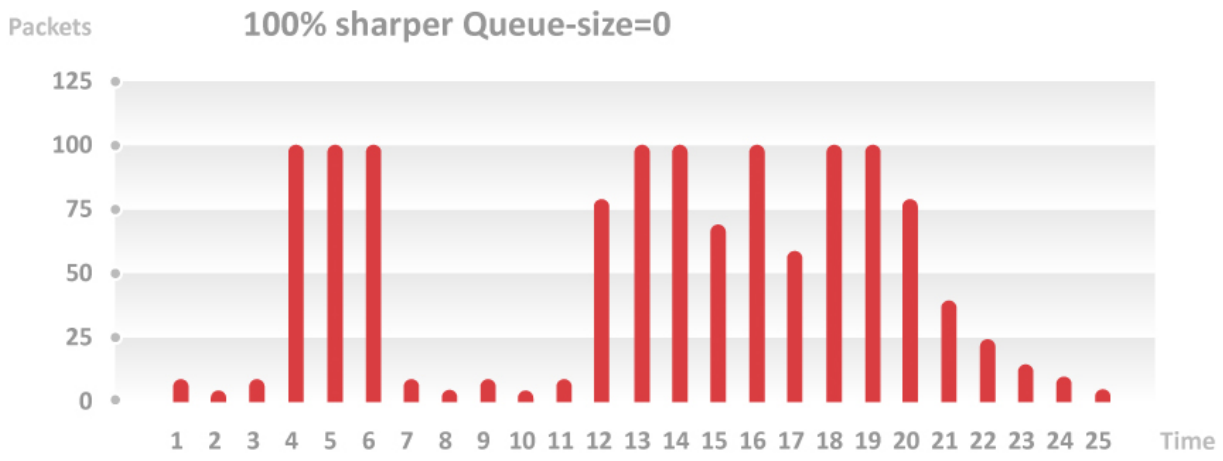


As you can see in the picture above there are **25 steps** and there is a total of **1610 incoming packets** over this time frame.

100% Shaper

A queue is 100% shaper when every packet that is over allowed limits will be dropped immediately. This way all packages that are not dropped will be sent out without any delay.

Let's apply **max-limit=100 packets per step** limitation to our example:



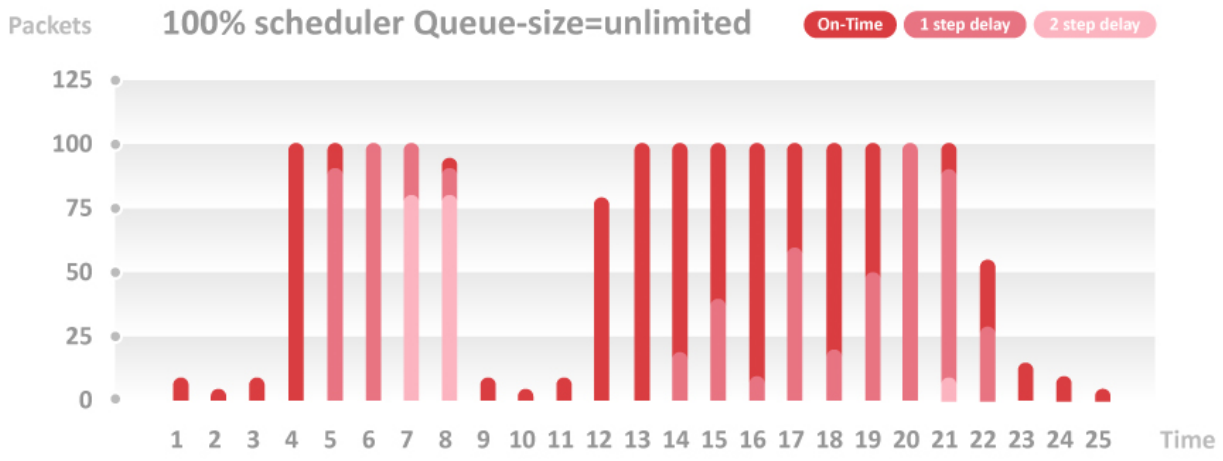
With this type of limitation, only 1250 out of 1610 packets were able to pass the queue (**22,4% packet drop**), but all packets arrive without delay.

100% Scheduler

A queue is 100% Scheduler when there are no packet drops at all, all packets are queued and will be sent out at the first possible moment.

In each step, the queue must send out queued packets from previous steps first and only then send out packets from this step, this way it is possible to keep the right sequence of packets.

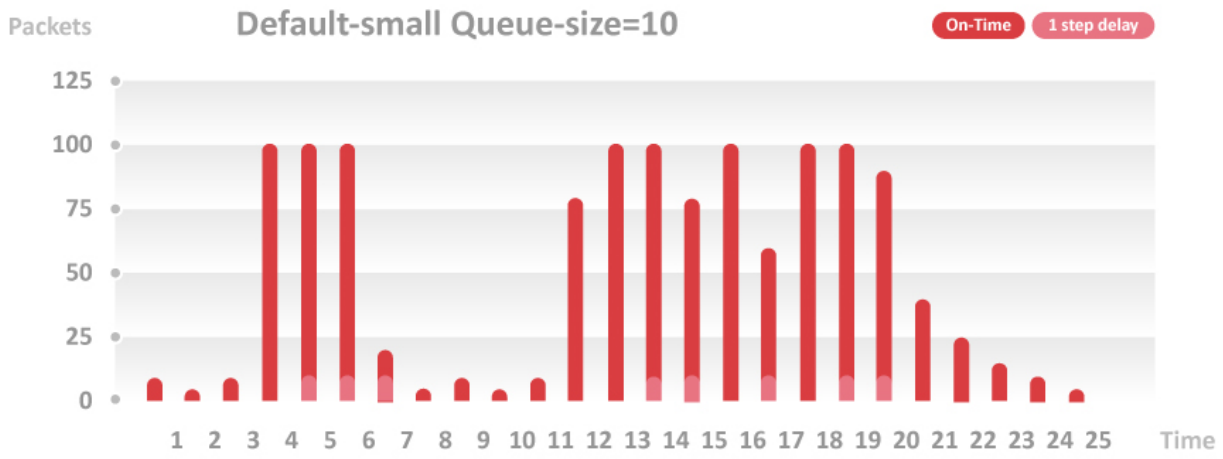
We will again use the same limit (**100 packets per step**).



There was no packet loss, but 630 **(39,1%) packets had 1 step delay**, and the other 170 **(10,6%) packets had 2 step delay**. (delay = latency)

Default-small queue type

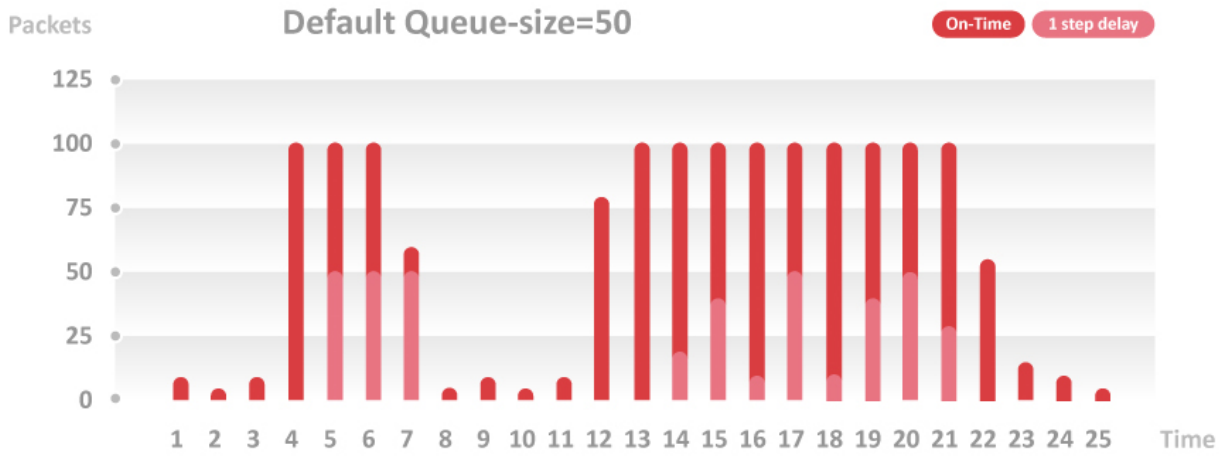
It is also possible to choose the middle way when the queue uses both of these queuing aspects (shaping and scheduling). By default, most of the queues in RouterOS have a queue size of 10.



There were 320 **(19,9%) packets dropped** and 80 **(5,0%) packets had 1 step delay**.

Default queue type

Another popular queue size in RouterOS is 50.



There were 190 **(11,8%) packets dropped** and 400 **(24,8%) packets had 1 step delay**.

Queue Burst

Introduction

Burst is a feature that allows satisfying queue requirements for additional bandwidth even if the required rate is bigger than **MIR** (**max-limit**) for a limited period of time.

Burst can occur only if **average-rate** of the queue for the last **burst-time** seconds is smaller than **burst-threshold**. Burst will stop if **average-rate** of the queue for the last **burst-time** seconds is bigger or equal to **burst-threshold.**

The burst mechanism is simple - if a burst is allowed **max-limit** value is replaced by the **burst-limit** value. When the burst is disallowed **max-limit** value remains unchanged.

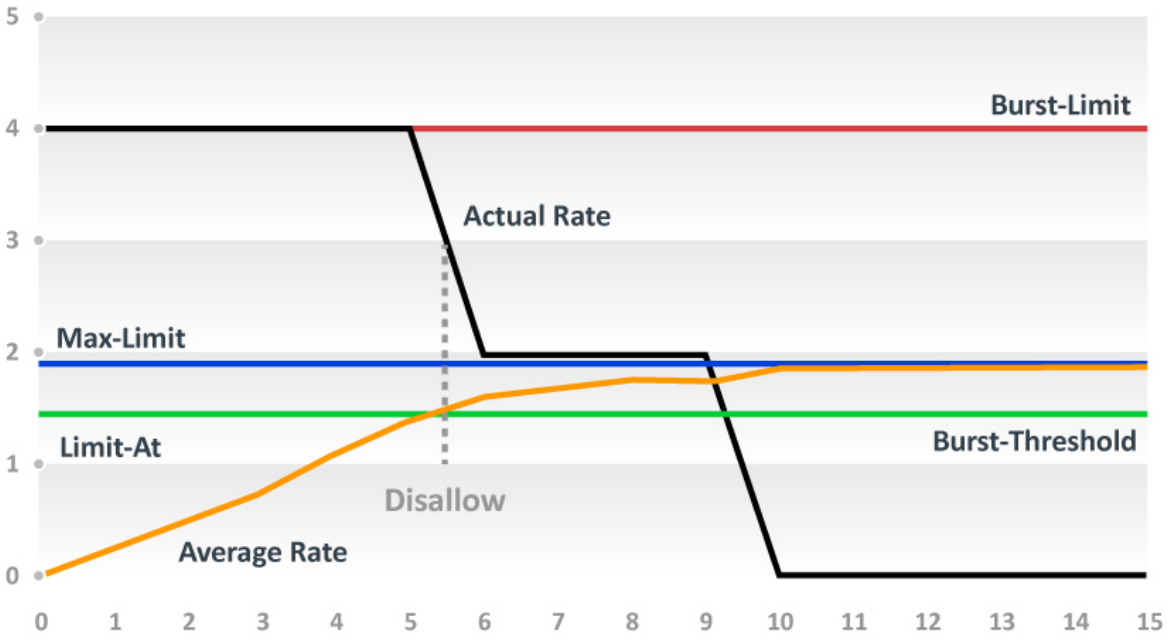
1. **burst-limit** (NUMBER) : maximal upload/download data rate which can be reached while the burst is allowed;
2. **burst-time** (TIME) : period of time, in seconds, over which the average data rate is calculated. (This is NOT the time of actual burst);
3. **burst-threshold** (NUMBER) : this is value of burst on/off switch;
4. **average-rate** (read-only) : Every 1/16 part of the **burst-time**, the router calculates the average data rate of each class over the last **burst-time** seconds;
5. **actual-rate** (read-only) : actual traffic transfer rate of the queue;

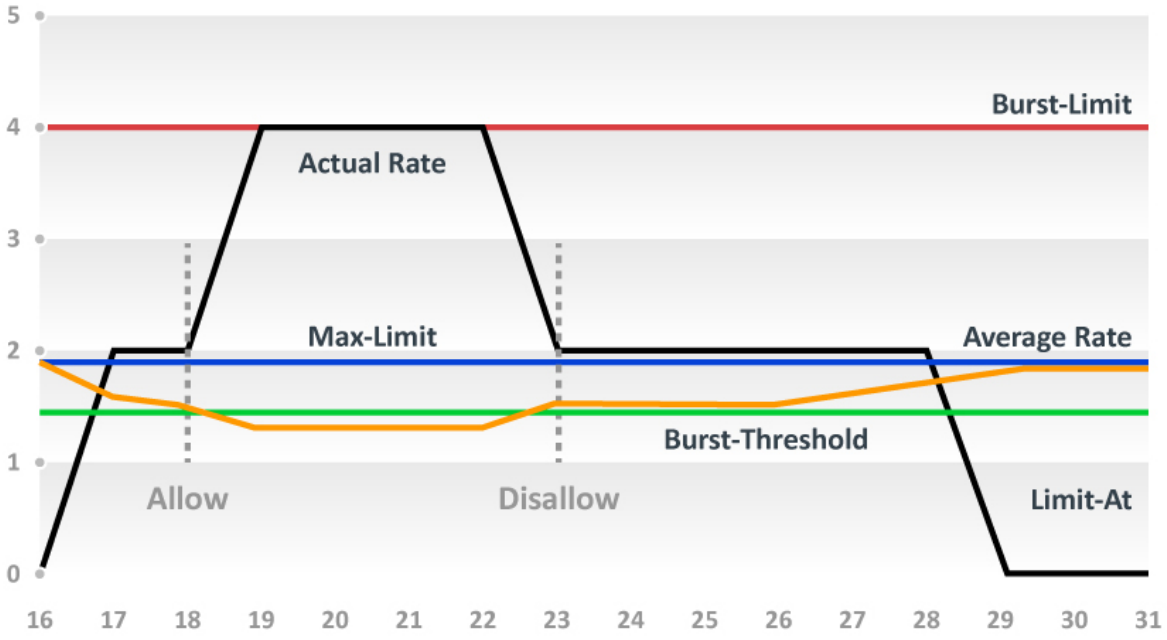
Example

Values: **limit-at=1M** , **max-limit=2M** , **burst-threshold=1500k** , **burst-limit=4M**

The client will try to download two 4MB (32Mb) blocks of data, the first download will start at zero seconds, and the second download will start at 17th second. Traffic was unused at the last minute.

Burst-time=16s





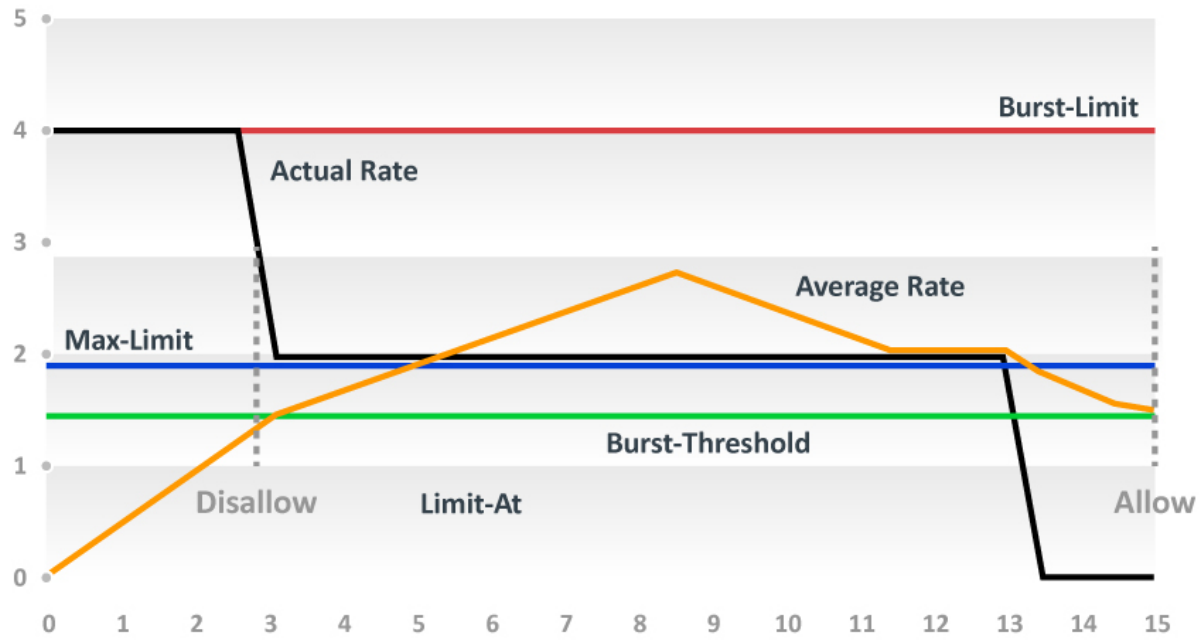
As we can see as soon as the client requested bandwidth it was able to get 4Mpbs burst for 6 seconds. This is longest possible burst with given values *(lon gest-burst-time = burst-threshold * burst-time / burst-limit).* As soon as the burst runs out rest of the data will be downloaded with 2Mbps. This way block of data was downloaded in 9 seconds - without burst, it would take 16 seconds. Burst has 7 seconds to recharge before the next download will start.

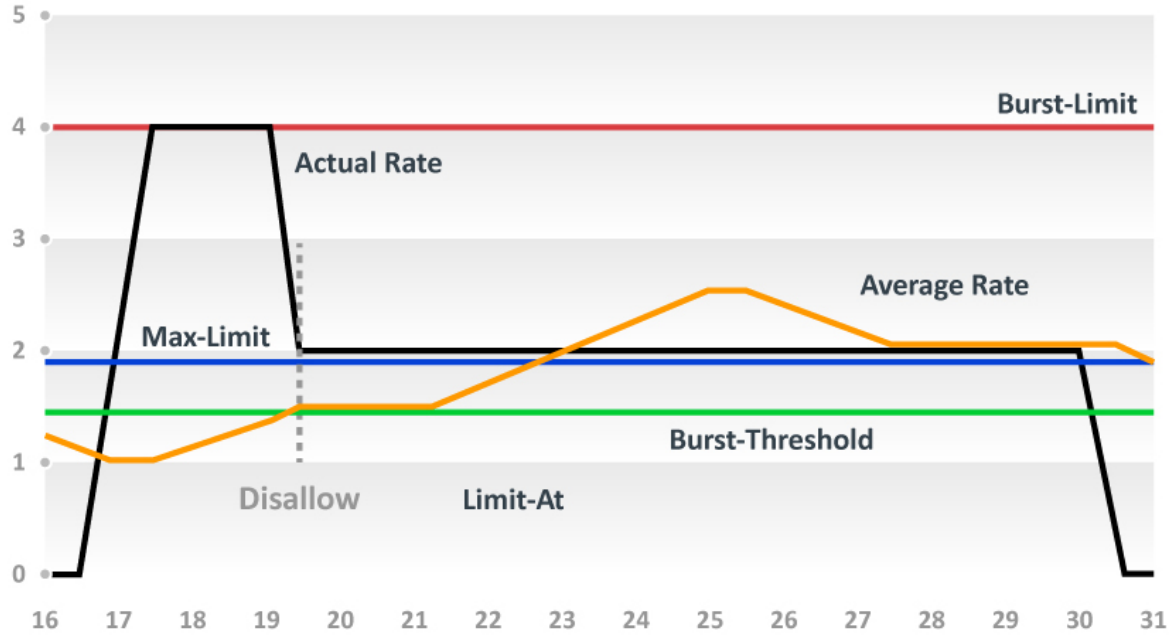
Note that burst is still disallowed when download started and it kicks in only afterward - in the middle of a download. So with this example, we proved that a burst may happen in the middle of a download. The burst was ~4 seconds long and the second block was downloaded 4 seconds faster than without burst.

The average rate is calculated every 1/16 of burst time so in this case 1s

Time	average-rate	burst	actual-rate
0	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0)/16=0Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
1	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+4)/16=250Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
2	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+4+4)/16=500Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
3	(0+0+0+0+0+0+0+0+0+0+0+0+0+4+4+4)/16=750Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
4	(0+0+0+0+0+0+0+0+0+0+0+0+4+4+4+4)/16=1000Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
5	(0+0+0+0+0+0+0+0+0+0+0+4+4+4+4+4)/16=1250Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
6	(0+0+0+0+0+0+0+0+0+0+4+4+4+4+4+4)/16=1500Kbps	average-rate = burst-threshold → Burst **not** allowed	**2Mbps**
7	(0+0+0+0+0+0+0+0+0+4+4+4+4+4+4+2)/16=1625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
8	(0+0+0+0+0+0+0+0+4+4+4+4+4+4+2+2)/16=1750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
9	(0+0+0+0+0+0+0+4+4+4+4+4+4+2+2+2)/16=1875Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
10	(0+0+0+0+0+0+4+4+4+4+4+4+2+2+2+2)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	**0Mbps**
11	(0+0+0+0+0+4+4+4+4+4+4+2+2+2+2+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
12	(0+0+0+0+4+4+4+4+4+4+2+2+2+2+0+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
13	(0+0+0+4+4+4+4+4+4+2+2+2+2+0+0+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
14	(0+0+4+4+4+4+4+4+2+2+2+2+0+0+0+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
15	(0+4+4+4+4+4+4+2+2+2+2+0+0+0+0+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
16	(4+4+4+4+4+4+2+2+2+2+0+0+0+0+0+0)/16=2Mbps	average-rate > burst-threshold → Burst not allowed	0Mbps
17	(4+4+4+4+4+2+2+2+2+0+0+0+0+0+0+0)/16=1750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
18	(4+4+4+4+2+2+2+2+0+0+0+0+0+0+0+2)/16=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps
19	(4+4+4+2+2+2+2+0+0+0+0+0+0+2+2)/16=1375Kbps	average-rate < burst-threshold → Burst **is** allowed	4Mbps
20	(4+4+2+2+2+2+0+0+0+0+0+0+2+2+4)/16=1375Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
21	(4+2+2+2+2+0+0+0+0+0+0+2+2+4+4)/16=1375Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
22	(2+2+2+2+0+0+0+0+0+0+2+2+4+4+4)/16=1375Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps
23	(2+2+2+0+0+0+0+0+0+2+2+4+4+4+4)/16=1500Kbps	average-rate = burst-threshold → Burst **not** allowed	2Mbps
24	(2+2+0+0+0+0+0+0+0+2+2+4+4+4+4+2)/16=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps
25	(2+0+0+0+0+0+0+0+2+2+4+4+4+4+2+2)/16=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps
26	(0+0+0+0+0+0+0+2+2+4+4+4+4+2+2+2)/16=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps
27	(0+0+0+0+0+0+2+2+4+4+4+4+2+2+2+2)/16=1625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
28	(0+0+0+0+0+2+2+4+4+4+4+2+2+2+2+2)/16=1750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps
29	(0+0+0+0+2+2+4+4+4+4+2+2+2+2+2+2)/16=1875Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps
30	(0+0+0+2+2+4+4+4+4+2+2+2+2+2+2+0)/16=1875Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps
31	(0+0+2+2+4+4+4+4+2+2+2+2+2+2+0+0)/16=1875Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps

Burst-time=8s





If we decrease burst-time to 8 seconds - we are able to see that in this case, bursts are only at the beginning of downloads The average rate is calculated every 1/16th of burst time, so in this case every 0.5 seconds.

Time	average-rate	burst	actual-rate
0.0	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0)/8=0Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
0.5	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+2)/8=250Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
1.0	(0+0+0+0+0+0+0+0+0+0+0+0+0+0+2+2)/8=500Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
1.5	(0+0+0+0+0+0+0+0+0+0+0+0+0+2+2+2)/8=750Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
2.0	(0+0+0+0+0+0+0+0+0+0+0+0+2+2+2+2)/8=1000Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
2.5	(0+0+0+0+0+0+0+0+0+0+0+2+2+2+2+2)/8=1250Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
3.0	(0+0+0+0+0+0+0+0+0+0+2+2+2+2+2+2)/8=1500Kbps	average-rate = burst-threshold → Burst **not** allowed	**2Mbps** (1Mb per 0,5sek)
3.5	(0+0+0+0+0+0+0+0+0+2+2+2+2+2+2+1)/8=1625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
4.0	(0+0+0+0+0+0+0+0+2+2+2+2+2+2+1+1)/8=1750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
4.5	(0+0+0+0+0+0+0+2+2+2+2+2+2+1+1+1)/8=1875Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
5.0	(0+0+0+0+0+0+2+2+2+2+2+2+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
5.5	(0+0+0+0+0+2+2+2+2+2+2+1+1+1+1+1)/8=2125Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
6.0	(0+0+0+0+2+2+2+2+2+2+1+1+1+1+1+1)/8=2250Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
6.5	(0+0+0+2+2+2+2+2+2+1+1+1+1+1+1+1)/8=2375Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
7.0	(0+0+2+2+2+2+2+2+1+1+1+1+1+1+1+1)/8=2500Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
7.5	(0+2+2+2+2+2+2+1+1+1+1+1+1+1+1+1)/8=2625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
8.0	(2+2+2+2+2+2+1+1+1+1+1+1+1+1+1+1)/8=2750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
8.5	(2+2+2+2+2+1+1+1+1+1+1+1+1+1+1+1)/8=2625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
9.0	(2+2+2+2+1+1+1+1+1+1+1+1+1+1+1+1)/8=2500Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
9.5	(2+2+2+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2375Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
10.0	(2+2+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2250Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
10.5	(2+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2125Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
11.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
11.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
12.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
12.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
13.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	**0Mbps** (0Mb per 0,5sek)
13.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+0)/8=1875Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)
14.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+0+0)/8=1750Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)
14.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+0+0+0)/8=1625Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)
15.0	(1+1+1+1+1+1+1+1+1+1+1+1+0+0+0+0)/8=1500Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)
15.5	(1+1+1+1+1+1+1+1+1+1+1+0+0+0+0+0)/8=1375Kbps	average-rate < burst-threshold → Burst **is** allowed	0Mbps (0Mb per 0,5sek)
16.0	(1+1+1+1+1+1+1+1+1+1+0+0+0+0+0+0)/8=1250Kbps	average-rate < burst-threshold → Burst is allowed	0Mbps (0Mb per 0,5sek)
16.5	(1+1+1+1+1+1+1+1+1+0+0+0+0+0+0+0)/8=1125Kbps	average-rate < burst-threshold → Burst is allowed	0Mbps (0Mb per 0,5sek)
17.0	(1+1+1+1+1+1+1+1+0+0+0+0+0+0+0+0)/8=1000Kbps	average-rate < burst-threshold → Burst is allowed	**2Mbps** (1Mb per 0,5sek)
17.5	(1+1+1+1+1+1+1+0+0+0+0+0+0+0+0+1)/8=1000Kbps	average-rate < burst-threshold → Burst is allowed	**4Mbps** (2Mb per 0,5sek)
18.0	(1+1+1+1+1+1+0+0+0+0+0+0+0+0+1+2)/8=1125Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
18.5	(1+1+1+1+1+0+0+0+0+0+0+0+0+1+2+2)/8=1250Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
19.0	(1+1+1+1+0+0+0+0+0+0+0+0+1+2+2+2)/8=1375Kbps	average-rate < burst-threshold → Burst is allowed	4Mbps (2Mb per 0,5sek)
19.5	(1+1+1+0+0+0+0+0+0+0+0+1+2+2+2+2)/8=1500Kbps	average-rate = burst-threshold → Burst **not** allowed	2Mbps (1Mb per 0,5sek)
20.0	(1+1+0+0+0+0+0+0+0+0+1+2+2+2+2+1)/8=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)

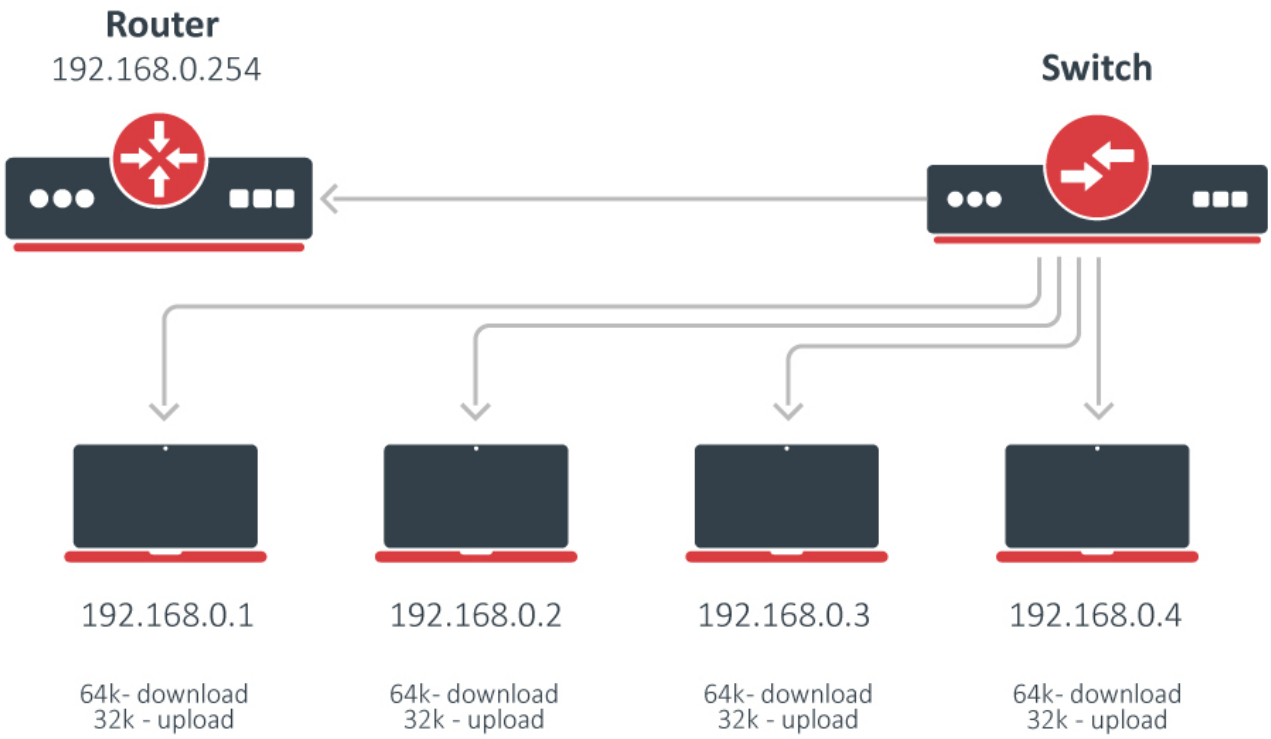
20.5	(1+0+0+0+0+0+0+0+0+1+2+2+2+2+1+1)/8=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
21.0	(0+0+0+0+0+0+0+0+1+2+2+2+2+1+1+1)/8=1500Kbps	average-rate = burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
21.5	(0+0+0+0+0+0+0+1+2+2+2+2+1+1+1+1)/8=1625Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
22.0	(0+0+0+0+0+0+1+2+2+2+2+1+1+1+1+1)/8=1750Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
22.5	(0+0+0+0+0+1+2+2+2+2+1+1+1+1+1+1)/8=1875Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
23.0	(0+0+0+0+1+2+2+2+2+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
23.5	(0+0+0+1+2+2+2+2+1+1+1+1+1+1+1+1)/8=2125Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
24.0	(0+0+1+2+2+2+2+1+1+1+1+1+1+1+1+1)/8=2250Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
24.5	(0+1+2+2+2+2+1+1+1+1+1+1+1+1+1+1)/8=2375Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
25.0	(1+2+2+2+2+1+1+1+1+1+1+1+1+1+1+1)/8=2500Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
25.5	(2+2+2+2+1+1+1+1+1+1+1+1+1+1+1+1)/8=2500Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
26.0	(2+2+2+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2375Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
26.5	(2+2+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2250Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
27.0	(2+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2125Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
27.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
28.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
28.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
29.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
29.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
30.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	2Mbps (1Mb per 0,5sek)
30.5	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1)/8=2000Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)
31.0	(1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+0)/8=1875Kbps	average-rate > burst-threshold → Burst not allowed	0Mbps (0Mb per 0,5sek)

PCQ example

Per Connection Queue (PCQ) is a queuing discipline that can be used to dynamically equalize or shape traffic for multiple users, using little administration. It is possible to divide PCQ scenarios into three major groups: equal bandwidth for a number of users, certain bandwidth equal distribution between users, and unknown bandwidth equal distribution between users.

Equal Bandwidth for a Number of Users

Use PCQ type queue when you need to equalize the bandwidth [and set max limit] for a number of users. We will set the 64kbps download and 32kbps upload limits.



There are two ways how to make this: using mangle and queue trees, or, using simple queues.

1. Mark all packets with packet-marks upload/download: (lets consider that ether1-WAN is the public interface to the Internet and ether2-LAN is a local interface where clients are connected):

```
/ip firewall mangle add chain=prerouting action=mark-packet \
   in-interface=ether2-LAN new-packet-mark=client_upload
/ip firewall mangle add chain=prerouting action=mark-packet \
   in-interface=ether1-WAN new-packet-mark=client_download
```

2. Setup two PCQ queue types - one for download and one for upload. *dst-address* is a classifier for the user's download traffic, and *src-address* for upload traffic:

```
/queue type add name="PCQ_download" kind=pcq pcq-rate=64000 pcq-classifier=dst-address
/queue type add name="PCQ_upload" kind=pcq pcq-rate=32000 pcq-classifier=src-address
```

3. Finally, two queue rules are required, one for download and one for upload:

```
/queue tree add parent=global queue=PCQ_download packet-mark=client_download
/queue tree add parent=global queue=PCQ_upload packet-mark=client_upload
```

If you don't like using mangle and queue trees, you can skip step 1, do step 2, and step 3 would be to create one simple queue as shown here:

```
/queue simple add target=192.168.0.0/24 queue=PCQ_upload/PCQ_download
```

Queue types

Overview

Queue type (also known as a queuing discipline) refers to the policy by which various data packets are managed in a network queue.

This policy determines which packet to send next from the queue when a data transmission has completed. The queue type impacts how the network handles data traffic, which in turn affects factors such as the distribution of bandwidth among different flows, latency, and the handling of congestion.

Queue types can be simple or complex, and different types have different strengths and weaknesses. For example, a First In, First Out (FIFO) queue is simple and ensures fairness in the order of packet processing, but it can lead to issues such as head-of-line blocking. More complex queue types, such as those that implement fair queuing or congestion avoidance, can help manage network performance more effectively, but they may require more resources and configuration.

Therefore, the choice of queue type depends on the specific needs and characteristics of the network.

Queue type introduction

1. **BFIFO (Byte-oriented First In, First Out):**
    - **Features:** BFIFO is a simple byte-oriented queuing discipline that sends out packets in the order they arrived and based on their size.
    - **Advantages:** Simplicity and fairness in packet management.
    - **Disadvantages:** BFIFO can lead to head-of-line blocking, where a large packet can delay all the smaller packets behind it. Also, there is no mechanism to prevent congestion or prioritize packets.
2. **PFIFO (Packet-oriented First In, First Out):**
    - **Features:** Like BFIFO but operates on a packet basis rather than bytes.
    - **Advantages:** Easy to implement and understand. It's fair in terms of packet numbers.
    - **Disadvantages:** Like BFIFO, it suffers from head-of-line blocking and doesn't have any congestion management mechanism.
3. **CAKE (Common Applications Kept Enhanced):**
    - **Features:** A comprehensive queue management system designed to combat bufferbloat and ensure fair queuing. It combines CoDel, FQ, and other technologies.
    - **Advantages:** Excellent at managing latency under any network condition. It's particularly useful in consumer broadband connections.
    - **Disadvantages:** More complex to configure than simpler queuing disciplines. Not ideal for all use cases, particularly where fine-tuned control is required.
4. **CoDel (Controlled Delay):**
    - **Features:** CoDel aims to improve bufferbloat by dropping packets to control queue delay.
    - **Advantages:** Good at managing latency and avoiding bufferbloat.
    - **Disadvantages:** CoDel on its own does not provide fair queuing.
5. **FQ_CoDel (Fair Queuing Controlled Delay):**
    - **Features:** Combines the latency management of CoDel with fair queuing.
    - **Advantages:** Excellent latency management and fair distribution of bandwidth among flows. Generally considered a good default choice.
    - **Disadvantages:** More complex than simpler queue disciplines, which can make it harder to configure.
6. **MQ_PFIFO (Multiqueue Packet First In, First Out):**
    - **Features:** MQ_PFIFO is an extension of PFIFO that supports multiple queues.
    - **Advantages:** Allows different queues for different types or priorities of traffic.
    - **Disadvantages:** Still suffers from head-of-line blocking and lacks a built-in congestion management mechanism.
7. **RED (Random Early Detection):**
    - **Features:** RED aims to anticipate and prevent congestion by randomly dropping packets before the queue becomes full.
    - **Advantages:** Helps to prevent congestion and can improve overall network performance.
    - **Disadvantages:** Configuration can be complex and needs to be tuned for the network's specific characteristics. Misconfiguration can lead to reduced performance.
8. **SFQ (Stochastic Fairness Queuing):**
    - **Features:** SFQ aims to ensure a fair distribution of resources among flows by assigning them to different dynamic queues.
    - **Advantages:** Helps to prevent a single flow from dominating the connection.
    - **Disadvantages:** SFQ does not manage latency or congestion. The number of queues can also increase memory usage.

In choosing a queue discipline, you'll need to consider the characteristics of your network and what you prioritize most, such as latency, fairness, simplicity, or congestion management.

CAKE

CAKE (Common Applications Kept Enhanced)

Description:

CAKE (Common Applications Kept Enhanced) is a modern, advanced queue management algorithm designed to cope with varying network conditions and different types of traffic. It was developed to address the limitations of older algorithms and provide an all-in-one solution to several common network problems.

CAKE's approach is to maintain fairness, minimize bufferbloat, and manage different types of traffic with minimal configuration and tuning. This is achieved through various techniques including flow isolation, bandwidth shaping, and prioritization of small packets.

Property	Description
Bandwith Limit:	This allows you to manually set the bandwidth limit for the CAKE shaper. For instance, if you're aware that your ISP provides you with a 100Mbps connection, you could set the rate to 100Mbps to prevent CAKE from trying to use more bandwidth than available.
Autorate Ingress	This is useful in situations where the connection quality varies. For example, if you're on a cellular data connection which can fluctuate wildly based on signal strength, this option allows CAKE to adjust the shaper bandwidth dynamically to match the estimated capacity of the link.
Overhead	The overhead parameter in the CAKE queue discipline allows you to specify the per-packet overhead (in bytes) to account for when shaping traffic. This is particularly important when dealing with technologies such as DSL, where protocol encapsulation adds additional bytes to each packet. Accurately accounting for these overheads will lead to more precise control of the actual bandwidth being used.
MPU	The Minimum Packet Unit (MPU) parameter allows you to round up the size of each packet to a specified minimum. This is useful for link layer technologies that have a minimum packet size. For example, if your link layer technology has a minimum packet size of 64 bytes, you can configure CAKE with `mpu 64`.
ATM	This is for Asynchronous Transfer Mode, a type of network technology often used in DSL broadband connections. ATM uses fixed 53-byte cells, each of which can carry 48 bytes of payload. The `atm` keyword compensates for this overhead.
Overhead Scheme	The overhead compensation feature in CAKE allows it to account for the extra bytes added by various link layer technologies, which can be significant in some cases. This is important because CAKE operates on the packet sizes reported by the Linux kernel, which do not include these extra bytes. If the overhead is not accounted for, CAKE's shaper might allow more data onto the link than it can actually handle, leading to unexpected packet loss.
RTT	Manually specify an RTT. This is for advanced users who know the exact RTT they want to use.
RTT Scheme	The CAKE queue discipline allows you to specify the Round Trip Time (RTT) it should consider when managing traffic. This is important because the time it takes for a packet to travel from a source to a destination and back impacts how CAKE manages network congestion. If the actual RTT of your network is close to the value you specify, both the throughput and latency of your network should be well managed.
Diffserv	Diffserv (Differentiated Services) is a mechanism used to prioritize and classify network traffic based on the Diffserv field in the IP packet header. CAKE (Common Applications Kept Enhanced) provides Diffserv support, allowing traffic to be divided into "tins" (traffic classes) and applying different treatment to each tin.
Flow Mode	When flow isolation is enabled, CAKE puts packets from different flows into different queues. Each queue maintains its own Active Queue Management (AQM) state, and packets are delivered from each queue fairly using a DRR++ (Deficit Round Robin++) algorithm. This algorithm works to minimize latency for "sparse" flows, or flows that contain fewer packets.
ACK Filter	ACK or acknowledgement filters are a feature of the Cake algorithm that allows for the handling of TCP ACK (acknowledgment) packets. TCP ACK packets are essential to the function of the TCP protocol; they acknowledge the receipt of TCP segments, providing a form of flow control. However, these packets can consume a significant portion of your upload bandwidth, especially when downloading large files. By implementing ACK filtering, Cake can compress these ACKs, reducing their impact on upload bandwidth.
NAT	These options control how CAKE handles traffic when Network Address Translation (NAT) is being used. `nat` makes CAKE consider the post-NAT addresses and ports when isolating flows, which can be useful when you're shaping traffic on a router that is also performing NAT for its connected devices.
Wash	The "wash" option in the CAKE queue discipline is used to address the issue of frequently mis-marked traffic entering or exiting a DiffServ (Differentiated Services) domain. When traffic passes through networks or providers, there is a chance that the DSCP (Differentiated Services Code Point) markings could be modified or incorrectly assigned.

Overhead Schemes

The overhead compensation feature in CAKE allows it to account for the extra bytes added by various link layer technologies, which can be significant in some cases. This is important because CAKE operates on the packet sizes reported by the Linux kernel, which do not include these extra bytes. If the overhead is not accounted for, CAKE's shaper might allow more data onto the link than it can actually handle, leading to unexpected packet loss.

1. **Manual Overhead Specification**: You can manually specify the overhead in bytes. For instance, if you know your link layer adds 18 bytes of overhead to each packet, you can configure CAKE with `overhead 18`. Negative values are also accepted, within a range of -64 to 256 bytes.
2. **MPU**: The Minimum Packet Unit (MPU) parameter allows you to round up the size of each packet to a specified minimum. This is useful for link layer technologies that have a minimum packet size. For example, if your link layer technology has a minimum packet size of 64 bytes, you can configure CAKE with `mpu 64`.
3. **ATM**: This is for Asynchronous Transfer Mode, a type of network technology often used in DSL broadband connections. ATM uses fixed 53-byte cells, each of which can carry 48 bytes of payload. The `atm` keyword compensates for this overhead.
4. **PTM**: This is for Packet Transfer Mode, another network technology often used in VDSL2 connections. PTM uses a 64b/65b encoding scheme, which effectively reduces the usable bandwidth by a small amount. The `ptm` keyword compensates for this overhead.
5. **Failsafe Overhead Keywords**: The `raw` and `conservative` keywords are provided for quick-and-dirty setup. `raw` turns off all overhead compensation in CAKE, and `conservative` compensates for more overhead than is likely to occur on any widely-deployed link technology.
6. **ADSL Overhead Keywords**: Most ADSL modems use ATM cell framing and have additional overhead due to the PPPoA or PPPoE protocol used. Keywords such as `pppoa-vcmux`, `pppoe-llc`, etc. are provided to account for these overheads.
7. **VDSL2 Overhead Keywords**: VDSL2 uses PTM instead of ATM and may also use PPPoE. Keywords such as `pppoe-ptm` and `bridged-ptm` are provided to account for these overheads.
8. **DOCSIS Cable Overhead Keyword**: DOCSIS is the standard for providing Internet service over cable-TV infrastructure. The `docsis` keyword is provided to account for the overhead of DOCSIS.
9. **Ethernet Overhead Keywords**: These keywords account for the overhead of Ethernet frames, including the preamble, inter-frame gap, and Frame Check Sequence. `ethernet` and `ether-vlan` are provided for Ethernet and Ethernet with VLAN respectively[1].

RTT Schemes

The CAKE queue discipline allows you to specify the Round Trip Time (RTT) it should consider when managing traffic. This is important because the time it takes for a packet to travel from a source to a destination and back impacts how CAKE manages network congestion. If the actual RTT of your network is close to the value you specify, both the throughput and latency of your network should be well managed.

Here are the RTT settings you can use, what they mean, and when you might use them:

1. `rtt TIME`: Manually specify an RTT. This is for advanced users who know the exact RTT they want to use.
2. `datacentre`: This is for extremely high-performance networks, such as a 10 Gigabit Ethernet (10GigE) network in a data center. The RTT is assumed to be 100 microseconds. Example: Use this if you're managing network traffic in a high-capacity data center.
3. `lan`: This is for pure Ethernet networks, such as those you might find in a home or office environment. The RTT is assumed to be 1 millisecond. Example: Use this if you're managing traffic on a wired home or office network, but not when shaping for an Internet access link.
4. `metro`: This is for traffic mostly within a single city. The RTT is assumed to be 10 milliseconds. Example: Use this if you're managing traffic for a network in a single city, like a city-wide corporate network.
5. `regional`: This is for traffic mostly within a European-sized country. The RTT is assumed to be 30 milliseconds. Example: Use this if you're managing traffic for a network that spans a country.
6. `internet`: This is suitable for most Internet traffic. The RTT is assumed to be 100 milliseconds. Example: Use this if you're managing general Internet traffic on a typical broadband connection.
7. `oceanic`: This is for Internet traffic with generally above-average latency, such as that suffered by Australasian residents. The RTT is assumed to be 300 milliseconds. Example: Use this if you're managing traffic in a location with high latency, like Australia or New Zealand.
8. `satellite`: This is for traffic via geostationary satellites. The RTT is assumed to be 1000 milliseconds (1 second). Example: Use this if you're managing traffic for a network that uses a satellite internet connection.
9. `interplanetary`: This disables Active Queue Management (AQM) actions, because the RTT is so long (3600 seconds). It's named "interplanetary" because the distance from Earth to Jupiter is about one light-hour. Example: This is not typically used in standard networking situations, but might be useful in extremely high latency situations, such as experimental long-distance communication scenarios.

Remember, these are guidelines, and the best setting depends on the specific characteristics and requirements of your network. If you are unsure, the `internet` setting is a good starting point for most scenarios[1].

FLOW ISOLATION PARAMETER

When flow isolation is enabled, CAKE puts packets from different flows into different queues. Each queue maintains its own Active Queue Management (AQM) state, and packets are delivered from each queue fairly using a DRR++ (Deficit Round Robin++) algorithm. This algorithm works to minimize latency for "sparse" flows, or flows that contain fewer packets.

The key aspect here is the method by which CAKE determines different flows, known as "flow isolation." CAKE uses a set-associative hashing algorithm to reduce flow collisions.

1. **flowblind** - This parameter disables flow isolation, and all traffic goes through a single queue for each 'tin' or traffic class. *Useful in scenarios where specific flow isolation is not needed or desired, such as when you want to process all traffic the same way regardless of source or destination.*
2. **srchost -** Here, flows are determined solely by the source address. This could be beneficial on the outgoing path of an Internet Service Provider (ISP) backhaul. *A telecom company might use this to ensure fair use of its backbone network by different regions or customers.*
3. **dsthost -** With this parameter, flows are characterized only by their destination address. This might be beneficial on the incoming path of an ISP backhaul. *An enterprise could use this to balance incoming traffic to its different servers.*
4. **hosts -** In this case, flows are defined by source-destination host pairs. This is host isolation rather than flow isolation. *This might be useful in a data center network, where you want to ensure that communication between specific pairs of servers is fair.*
5. **flows** - Flows are characterized by the entire 5-tuple: source address, destination address, transport protocol, source port, and destination port. This is the kind of flow isolation performed by SFQ and fq_codel. *This could be used by a cloud provider to ensure fairness among different virtual machines communicating over various protocols and ports.*
6. **dual-srchost** - Here, flows are defined by the 5-tuple, and fairness is applied first over source addresses, then over individual flows. This is a good choice for outgoing traffic from a Local Area Network (LAN) to the internet. *A university might use this to prevent any single user or device from hogging the internet connection, no matter how many different connections they're using.*
7. **dual-dsthost** - In this case, flows are defined by the 5-tuple, and fairness is applied first over destination addresses, then over individual flows. This is suitable for incoming traffic to a LAN from the internet. *A large company could use this to prevent any single server or system from overwhelming the network's incoming bandwidth.*
8. **triple-isolate** - This is the default setting where flows are defined by the 5-tuple. Fairness is applied over both source and destination addresses intelligently, as well as over individual flows. This prevents any one host on either side of the link from monopolizing it with a large number of flows. *A Internet Service Provider (ISP) might use this to ensure fair service to all its customers, regardless of how many connections they have and whether they*

Ack Filter

ACK or acknowledgement filters are a feature of the Cake algorithm that allows for the handling of TCP ACK (acknowledgment) packets. TCP ACK packets are essential to the function of the TCP protocol; they acknowledge the receipt of TCP segments, providing a form of flow control. However, these packets can consume a significant portion of your upload bandwidth, especially when downloading large files. By implementing ACK filtering, Cake can compress these ACKs, reducing their impact on upload bandwidth.

1. **ack-filter** - This enables the ACK filter feature. With this option, Cake will attempt to identify and filter out ACK packets that are not conveying significant new information or do not need to be sent immediately, helping to improve the utilization of the upload bandwidth.
2. **ack-filter-aggressive** - This is a more aggressive version of the ack-filter option. It will result in more ACK packets being compressed or filtered out, which can lead to further improvements in upload bandwidth utilization but may potentially impact TCP performance.The use of the ack-filter or ack-filter-aggressive options depends on your specific network conditions and requirements. For instance, if you find that ACK packets are using a large portion of your available upload bandwidth, then enabling the ACK filter might help. On the other hand, if you're experiencing issues with TCP performance and suspect that ACK filtering might be contributing, you could try disabling it or using the less aggressive option.

Wash

The "wash" option in the CAKE queue discipline is used to address the issue of frequently mis-marked traffic entering or exiting a DiffServ (Differentiated Services) domain. When traffic passes through networks or providers, there is a chance that the DSCP (Differentiated Services Code Point) markings could be modified or incorrectly assigned.

To illustrate this with a real-life example and analogy:

Let's imagine you are running a busy airport. Different airlines have assigned different priority levels to their passengers based on their ticket classes (economy, business, first-class). These priority levels are analogous to the DSCP markings in a network.

However, as passengers move through various transit airports, there is a possibility that the assigned priority levels could be changed or mis-marked due to different airline policies or inconsistencies at the transit airports. This is similar to the mis-marking of traffic in transit networks.

Now, in the context of the airport, let's assume that upon arrival at your airport, passengers are divided into separate queues based on their priority levels. This division ensures that passengers in higher-priority classes are processed more quickly and receive better service.

However, because the priority levels assigned at previous airports may not be reliable, it becomes necessary to "wash" or clear the assigned priority levels before the passengers enter the queues at your airport. This is similar to the "wash" option in CAKE, which clears extra DSCP markings after priority queuing has taken place.

In situations where inbound traffic's DSCP markings cannot be trusted (similar to cases like Comcast Cable), it is recommended to use a single queue "besteffort" mode with the "wash" option. This means that all incoming traffic is treated as the default "best effort" class, and any potentially unreliable or mis-marked DSCP values are cleared before further processing.

In our airport analogy, using a single queue "besteffort" mode with "wash" would mean that regardless of the priority assigned at previous airports, all passengers are initially treated as standard passengers (best effort class) until their priority can be accurately determined and assigned at your airport.

This approach ensures that even if the DSCP markings of incoming traffic have been mis-marked or modified during transit, the traffic is treated fairly and uniformly within your network, without relying on potentially unreliable markings from external sources.

By applying the "wash" option in CAKE, network administrators can mitigate the impact of mis-marked or modified DSCP markings, providing a more consistent and reliable Quality of Service (QoS) treatment within their network domain.

Diffserv RFC2474 and RFC2475

Diffserv (Differentiated Services) is a mechanism used to prioritize and classify network traffic based on the Diffserv field in the IP packet header. CAKE (Common Applications Kept Enhanced) provides Diffserv support, allowing traffic to be divided into "tins" (traffic classes) and applying different treatment to each tin. Here's a breakdown of the Diffserv presets in CAKE along with real-world examples:

1. besteffort -The "besteffort" preset in CAKE disables priority queuing and places all traffic into a single tin. This means that all traffic is treated equally without any specific prioritization. *This preset can be suitable for non-critical or low-priority traffic, such as general web browsing or background file downloads, where equal treatment is sufficient.*
2. precedence - The "precedence" preset enables the legacy interpretation of the TOS (Type of Service) "Precedence" field. However, its usage on the modern internet is discouraged, as it is an outdated mechanism. *In the past, this field was used to indicate different levels of priority, such as high, medium, or low, but it is no longer widely used or recommended.*
3. diffserv4 - The "diffserv4" preset provides a general-purpose Diffserv implementation with four tins:
    - Bulk: This tin corresponds to CS1 (Class Selector 1) or LE (Low Extra), and it has a threshold of 6.25%. Traffic in this tin typically has a low priority.
    - Best Effort: This tin is for general traffic that doesn't fall into any specific Diffserv class. It has a threshold of 100%, meaning it receives all remaining bandwidth.
    - Video: This tin encompasses AF4x, AF3x, CS3, AF2x, CS2, TOS4, and TOS1. It has a threshold of 50%, providing a moderate priority for video traffic.
    - Voice: This tin covers CS7, CS6, EF (Expedited Forwarding), VA (Voice Admit), CS5, and CS4. It has a threshold of 25%, giving high priority to voice traffic.

In a network where video streaming, voice over IP (VoIP), and general internet traffic coexist, this preset can ensure that video and voice traffic receive appropriate priority, while bulk and best-effort traffic are handled accordingly.

4. diffserv3 (default) - The "diffserv3" preset is the default Diffserv implementation in CAKE, providing three tins:

    - Bulk: Similar to the "diffserv4" preset, this tin represents CS1 or LE with a 6.25% threshold, serving as a low-priority traffic class.
    - Best Effort: This tin is for general traffic and has a threshold of 100%, treating all remaining traffic equally.
    - Voice: This tin covers CS7, CS6, EF, VA, and TOS4. It has a threshold of 25% and applies a reduced CoDel (Controlled Delay) interval, giving high priority to voice traffic.

In a network where voice traffic requires high priority, such as a VoIP system, while other traffic falls into a general category, the "diffserv3" preset can ensure appropriate priority for voice packets while maintaining fairness for other traffic.

4. diffserv8 or fwmark MASK - The "fwmark" option in CAKE allows for overriding the tin selection based on a firewall mark (fwmark) associated with each packet. The fwmark value is masked and used to determine the tin number for the packet.

By leveraging fwmark-based overriding, network administrators can create custom firewall policies that dictate the Diffserv treatment for specific packets. This can be useful for implementing specific QoS policies based on unique requirements or network conditions.

It's important to note that the choice of a specific Diffserv preset or custom configuration depends on the network's traffic characteristics, QoS requirements, and the desired prioritization of different traffic classes.

"unlimited" or "autorate-ingress"

When using the "unlimited" or "autorate-ingress" option in the CAKE queue discipline, CAKE automatically determines the download speed based on observed packet arrival times. Here's how it works:

1. Monitoring Packet Arrival Times: CAKE continuously monitors the arrival times of incoming packets. It keeps track of the time intervals between consecutive packets to estimate the rate at which packets are being received.

2. Calculating Average Packet Arrival Rate: CAKE calculates the average packet arrival rate based on the observed arrival times. By dividing the number of received packets by the total time elapsed, it determines the average rate at which packets are arriving.
3. Deriving Download Speed: From the average packet arrival rate, CAKE infers the download speed or throughput of the network connection. It assumes that the packet arrival rate corresponds to the download speed, given that each packet represents a certain amount of data.
4. Dynamic Adjustment: As CAKE continues to monitor packet arrival times, it adjusts the download speed estimation dynamically. If the arrival rate increases, CAKE will update the download speed estimate to reflect the higher throughput. Conversely, if the arrival rate decreases, CAKE will adjust the estimate accordingly.

By automatically determining the download speed, CAKE adapts to changes in the network conditions and ensures that traffic shaping and queuing algorithms are adjusted to optimize the utilization of available bandwidth.

It's worth noting that while CAKE can estimate the download speed based on packet arrival times, it does not have direct knowledge of the link capacity or the true download speed as reported by the network equipment or service provider. Instead, it relies on observed packet arrival rates to approximate the download speed for traffic shaping purposes.

PFIFO,BFIFO

PFIFO (Packet First-In, First-Out)

Description:

PFIFO (Packet First-In, First-Out) is a queue management strategy that follows the basic queue logic. The main principle of PFIFO is that the first packet to arrive is the first packet to be transmitted out. This method is simple, and straightforward, and ensures that no packet receives preferential treatment over the others.

When packets enter the queue, they are placed at the end (tail) of the queue. As space becomes available for transmission, the packets at the front (head) of the queue are transmitted first. If the queue is full when a packet arrives, the packet is dropped or other policies are followed, depending on the overall queue management strategy in place.

Characteristics:

- **Simplicity:** PFIFO is one of the simplest forms of queue management. It doesn't require complex algorithms or significant computational resources. This makes it easy to implement and understand.
- **Fairness:** All packets are treated equally, regardless of their source, destination, or content. The first packet in is the first one out.
- **No Prioritization:** There's no inherent capability to prioritize certain types of traffic over others. While this ensures fairness, it can be a drawback when dealing with different types of network traffic that might need prioritization.

Examples:

Imagine a line at the post office. Each person (packet) waiting in line represents a data packet waiting in the queue. The first person in line gets served first, then the next, and so on. If the post office is too busy and the line is full, any new person arriving would have to wait or leave (packet is dropped).

Conclusion:

While PFIFO is simple and straightforward, its lack of traffic prioritization capabilities can be a downside, especially when managing network traffic that contains different types of data. In these situations, more complex queue management algorithms, such as Priority Queuing (PQ), Class-Based Weighted Fair Queuing (CBWFQ), or Low Latency Queuing (LLQ), might be more suitable.

BFIFO (Byte First-In, First-Out)

Description:

BFIFO (Byte First-In, First-Out) is another queue management strategy, similar to PFIFO in its principle of operation, but with a key difference: BFIFO operates based on the size of packets, or bytes, rather than on the number of packets.

In a BFIFO system, when packets enter the queue, they are still placed at the end, and packets at the front are transmitted first. However, the queue size is calculated in bytes, not in number of packets. When the queue reaches its maximum byte size, any new packets that arrive are dropped, or other policies are followed.

Characteristics:

- **Byte-Based:** Unlike PFIFO, BFIFO calculates the queue size based on bytes, which allows for a more precise control of bandwidth usage.
- **Fairness:** BFIFO also ensures fairness as it treats all packets equally, regardless of their source, destination, or content.
- **No Prioritization:** Similar to PFIFO, there's no inherent capability in BFIFO to prioritize certain types of traffic over others.

Examples:

Consider a public bus as an example of a BFIFO system. The bus has a maximum capacity of passengers it can carry (similar to the byte size of the queue). Even if there is a line of people waiting (packets), if the bus is full, new passengers cannot get on (packets are dropped).

Comparison: PFIFO vs. BFIFO

While both PFIFO and BFIFO are FIFO (First-In, First-Out) strategies and follow the basic queue logic, there are some differences:

- **Queue Size:** PFIFO considers the queue size in terms of the number of packets, while BFIFO calculates the queue size in bytes. This means that BFIFO takes into account the size of each packet, which can allow for more accurate control of bandwidth usage.
- **Packet Dropping:** In PFIFO, packets are dropped when the queue is full in terms of the number of packets. In BFIFO, packets are dropped when the byte size of the queue is exceeded, even if this means dropping a packet that is larger than the remaining space in the queue.
- **Fairness:** Both PFIFO and BFIFO are fair in terms of the order of packet transmission – the first in is the first out. However, in a situation where packets are of different sizes, PFIFO could lead to a situation where a large packet takes up a lot of resources but is treated the same as a smaller packet. On the other hand, BFIFO's byte-based approach can provide more balanced resource usage.
- **Complexity:** Both strategies are simple compared to more complex queuing strategies that prioritize certain types of traffic. However, BFIFO is slightly more complex than PFIFO because it requires tracking the total size of all packets in the queue.

Overall, the choice between PFIFO and BFIFO depends on the specific requirements of your network and the characteristics of your traffic.

RAW

Introduction

Firewall RAW table allows to selectively bypass or drop packets before connection tracking that way significantly reducing the load on CPU. The tool is very useful for DsS/DDoS attack mitigation.

The RAW table does not have matchers that depend on connection tracking (like connection-state, layer7, etc.).
If a packet is marked to bypass the connection tracking packet de-fragmentation will not occur.

Chains

There are two predefined chains in RAW tables:

- **prerouting** - used to process any packet entering the router
- **output** - used to process packets originated from the router and leaving it through one of the interfaces. Packets passing through the router are not processed against the rules of the output chain

Packet flow diagrams illustrate how packets are processed in RouterOS.

Properties

Property	Description
action (*action name*; Default: **accept**)	Action to take if a packet is matched by the rule: • accept - accept the packet. A packet is not passed to the next firewall rule. • add-dst-to-address-list - add destination address to address list specified by `address-list` parameter • add-src-to-address-list - add source address to address list specified by `address-list` parameter • drop - silently drop the packet • jump - jump to the user-defined chain specified by the value of `jump-target` parameter • log - add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip: port->dst-ip: port and length of the packet. After a packet is matched it is passed to the next rule in the list, similar as `passthrough` • notrack - do not send a packet to connection tracking. • passthrough - ignore this rule and go to the next one (useful for statistics). • return - passes control back to the chain from where the jump took place
address-list (*string*; Default:)	Name of the address list to be used. Applicable if action is `add-dst-to-address-list` or `add-src-to-address-list`
address-list-timeout (*none-dynamic* | *none-static* | *time*; Default: **none-dynamic**)	Time interval after which the address will be removed from the address list specified by `address-list` parameter. Used in conjunction with `add-dst-to-address-list` or `add-src-to-address-list` actions • Value of none-dynamic (`00:00:00`) will leave the address in the address list till reboot • Value of none-static will leave the address in the address list forever and will be included in configuration export/backup
chain (*name*; Default:)	Specifies to which chain rule will be added. If the input does not match the name of an already defined chain, a new chain will be created
comment (*string*; Default:)	Descriptive comment for the rule
dscp (*integer: 0..63*; Default:)	Matches DSCP IP header field

dst-address (*IP/netmask | IP range*; Default:)	Matches packets which destination is equal to specified IP or falls into a specified IP range
dst-address-list (*name*; Default:)	Matches destination address of a packet against user-defined address list
dst-address-type (*unicast | local | broadcast | multicast*; Default:)	Matches destination address type: • unicast - IP address used for point to point transmission • local - if dst-address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
dst-limit (*integer[/time],integer,dst-address | dst-port | src-address[/time]*; Default:)	Matches packets until a given rate is exceeded. Rate is defined as packets per time interval. As opposed to the limit matcher, every flow has its own limit. Flow is defined by a mode parameter. Parameters are written in the following format: `count[/time],burst,mode[/expire]`. • **count** - packet count per time interval per-flow to match • **time** - specifies the time interval in which the packet count per flow cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets per flow to match: this number gets recharged by one every `time/count`, up to this number • **mode** - this parameter specifies what unique fields define flow (src-address, dst-address, src-and-dst-address, dst-address-and-port, addresses-and-dst-port) • **expire** - specifies interval after which flow with no packets will be allowed to be deleted (optional)
dst-port (*integer[-integer]: 0..65535*; Default:)	List of destination port numbers or port number ranges
fragment (*yes|no*; Default:)	Matches fragmented packets. The first (starting) fragment does not count. If connection tracking is enabled there will be no fragments as the system automatically assembles every packet
hotspot (*auth | from-client | http | local-dst | to-client*; Default:)	
icmp-options (*integer:integer*; Default:)	Matches ICMP type: code fields
in-bridge-port (*name*; Default:)	Actual interface the packet has entered the router if the incoming interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings.
in-interface (*name*; Default:)	Interface the packet has entered the router
in-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as in-interface
ingress-priority (*integer: 0..63*; Default:)	Matches ingress the priority of the packet. Priority may be derived from VLAN, WMM, or MPLS EXP bit. Read more
ipsec-policy (*in | out, ipsec | none*; Default:)	Matches the policy used by IPsec. Value is written in the following format: **`direction,policy`**. The direction is Used to select whether to match the policy used for decapsulation or the policy that will be used for encapsulation. • in - valid in the PREROUTING chain • out - valid in the OUTPUT chain • ipsec - matches if the packet is subject to IPsec processing; • none - matches packet that is not subject to IPsec processing (for example, IpSec transport packet). For example, if a router receives an IPsec encapsulated Gre packet, then the rule `ipsec-policy=in,ipsec` will match Gre packet, but the rule `ipsec-policy=in,none` will match the ESP packet.

ipv4-options (*any | loose-source-routing | no-record-route | no-router-alert | no-source-routing | no-timestamp | none | record-route | router-alert | strict-source-routing | timestamp*; Default:)	Matches IPv4 header options. • any - match packet with at least one of the ipv4 options • loose-source-routing - match packets with a loose source routing option. This option is used to route the internet datagram based on information supplied by the source • no-record-route - match packets with no record route option. This option is used to route the internet datagram based on information supplied by the source • no-router-alert - match packets with no router alter option • no-source-routing - match packets with no source routing option • no-timestamp - match packets with no timestamp option • record-route - match packets with record route option • router-alert - match packets with router alter option • strict-source-routing - match packets with strict source routing option • timestamp - match packets with a timestamp
jump-target (*name*; Default:)	Name of the target chain to jump to. Applicable only if `action=jump`
limit (*integer,time,integer*; Default:)	Matches packets up to a limited rate (packet rate or bit rate). A rule using this matcher will match until this limit is reached. Parameters are written in the following format: `count[/time],burst:mode`. • **count** - packet or bit count per time interval to match • **time** - specifies the time interval in which the packet or bit count cannot be exceeded (optional, 1s will be used if not specified) • **burst** - initial number of packets or bits to match: this number gets recharged every 10ms so burst should be at least 1/100 of a rate per the second • **mode** - packet or bit mode
log (*yes | no; Default:* **no**)	Add a message to the system log containing the following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port, and length of the packet.
log-prefix (*string*; Default:)	Adds specified text at the beginning of every log message. Applicable if *action=log* or *log=yes* configured.
nth (*integer,integer*; Default:)	Matches every nth packet: *nth=2,1* rule will match every first packet of 2, hence, 50% of all the traffic that is matched by the rule
out-bridge-port (*name*; Default:)	Actual interface the packet is leaving the router if the outgoing interface is a bridge. Works only if **use-ip-firewall** is enabled in bridge settings
out-interface (; Default:)	Interface the packet is leaving the router
out-interface-list (*name*; Default:)	Set of interfaces defined in interface list. Works the same as out-interface
packet-size (*integer[-integer]:0..65535*; Default:)	Matches packets of specified size or size range in bytes.
per-connection-classifier (*ValuesToHash:Denominator/Remainder*; Default:)	PCC matcher allows dividing traffic into equal streams with the ability to keep packets with a specific set of options in one particular stream
port (*integer[-integer]: 0..65535*; Default:)	Matches if any (source or destination) port matches the specified list of ports or port ranges. Applicable only if `protocol` is TCP or UDP
priority (*integer: 0..63*; Default:)	
protocol (*name or protocol ID*; Default: **tcp**)	Matches particular IP protocol specified by protocol name or number
psd (*integer,time,integer,integer*; Default:)	Attempts to detect TCP and UDP scans. Parameters are in the following format `WeightThreshold, DelayThreshold, LowPortWeight, HighPortWeight` • **WeightThreshold** - total weight of the latest TCP/UDP packets with different destination ports coming from the same host to be treated as port scan sequence • **DelayThreshold** - delay for the packets with different destination ports coming from the same host to be treated as possible port scan subsequence • **LowPortWeight** - the weight of the packets with privileged (<1024) destination port • **HighPortWeight** - the weight of the packet with non-privileged destination port

random (*integer: 1..99*; Default:)	Matches packets randomly with a given probability
src-address (*Ip/Netmaks, Ip range*; Default:)	Matches packets which source is equal to specified IP or falls into a specified IP range
src-address-list (*name*; Default:)	Matches source address of a packet against user-defined address list
src-address-type (*unicast* | *local* | *broadcast* | *multicast*; Default:)	Matches source address type: • unicast - IP address used for point to point transmission • local - if an address is assigned to one of the router's interfaces • broadcast - packet is sent to all devices in a subnet • multicast - packet is forwarded to a defined group of devices
src-port (*integer[-integer]: 0..65535*; Default:)	List of source ports and ranges of source ports. Applicable only if a protocol is TCP or UDP
src-mac-address (*MAC address*; Default:)	Matches source MAC address of the packet
tcp-flags (*ack* | *cwr* | *ece* | *fin* | *psh* | *rst* | *syn* | *urg*; Default:)	Matches specified TCP flags • ack - acknowledging data • cwr - congestion window reduced • ece - ECN-echo flag (explicit congestion notification) • fin - close connection • psh - push function • rst - drop connection • syn - new connection • urg - urgent data
tcp-mss (*integer[-integer]: 0..65535*; Default:)	Matches TCP MSS value of an IP packet
time (*time-time,sat* | *fri* | *thu* | *wed* | *tue* | *mon* | *sun*; Default:)	Allows to create a filter based on the packets' arrival time and date or, for locally generated packets, departure time and date
tls-host (*string*; Default:)	Allows matching traffic based on TLS hostname. Accepts GLOB syntax for wildcard matching. Note that the matcher will not be able to match hostname if the TLS handshake frame is fragmented into multiple TCP segments (packets).
ttl (*integer: 0..255*; Default:)	Matches packets TTL value

UPnP

Introduction

The MikroTik RouterOS supports Universal Plug and Play architecture for transparent peer-to-peer network connectivity of personal computers and network-enabled intelligent devices or appliances.

UPnP enables data communication between any two devices under the command of any control device on the network. Universal Plug and Play is completely independent of any particular physical medium. It supports networking with automatic discovery without any initial configuration, whereby a device can dynamically join a network. DHCP and DNS servers are optional and will be used if available on the network. UPnP implements a simple yet powerful NAT traversal solution, that enables the client to get full two-way peer-to-peer network support from behind the NAT.

There are two interface types for UPnP: **internal** (the one local clients are connected to) and **external** (the one the Internet is connected to). **A router may only have one active external interface with a 'public' IP address on it**, and as many internal interfaces as needed, all with source-NATted 'internal' IP addresses. The protocol works by creating dynamic NAT entries.

The UPnP protocol is used for many modern applications, like most DirectX games, as well as for various Windows Messenger features like remote assistance, application sharing, file transfer, voice, video from behind a firewall.

Configuration

General properties

```
/ip upnp
```

Property	Description
allow-disable-external-interface (*yes* / *no* ; Default: **yes**)	whether or not should the users are allowed to disable the router's external interface. This functionality (for users to be able to turn the router's external interface off without any authentication procedure) is required by the standard, but as it is sometimes not expected or unwanted in UPnP deployments which the standard was not designed for (it was designed mostly for home users to establish their own local networks), you can disable this behavior
enabled (*yes* / *no* ; Default: **no**)	Enable UPnP service
show-dummy-rule (*yes* / *no* ; Default: **yes**)	Enable a workaround for some broken implementations, which are handling the absence of UPnP rules incorrectly (for example, popping up error messages). This option will instruct the server to install a dummy (meaningless) UPnP rule that can be observed by the clients, which refuse to work correctly otherwise

If you do not disable the **allow-disable-external-interface**, any user from the local network will be able (without any authentication procedures) to disable the router's external interface

UPnP Interfaces

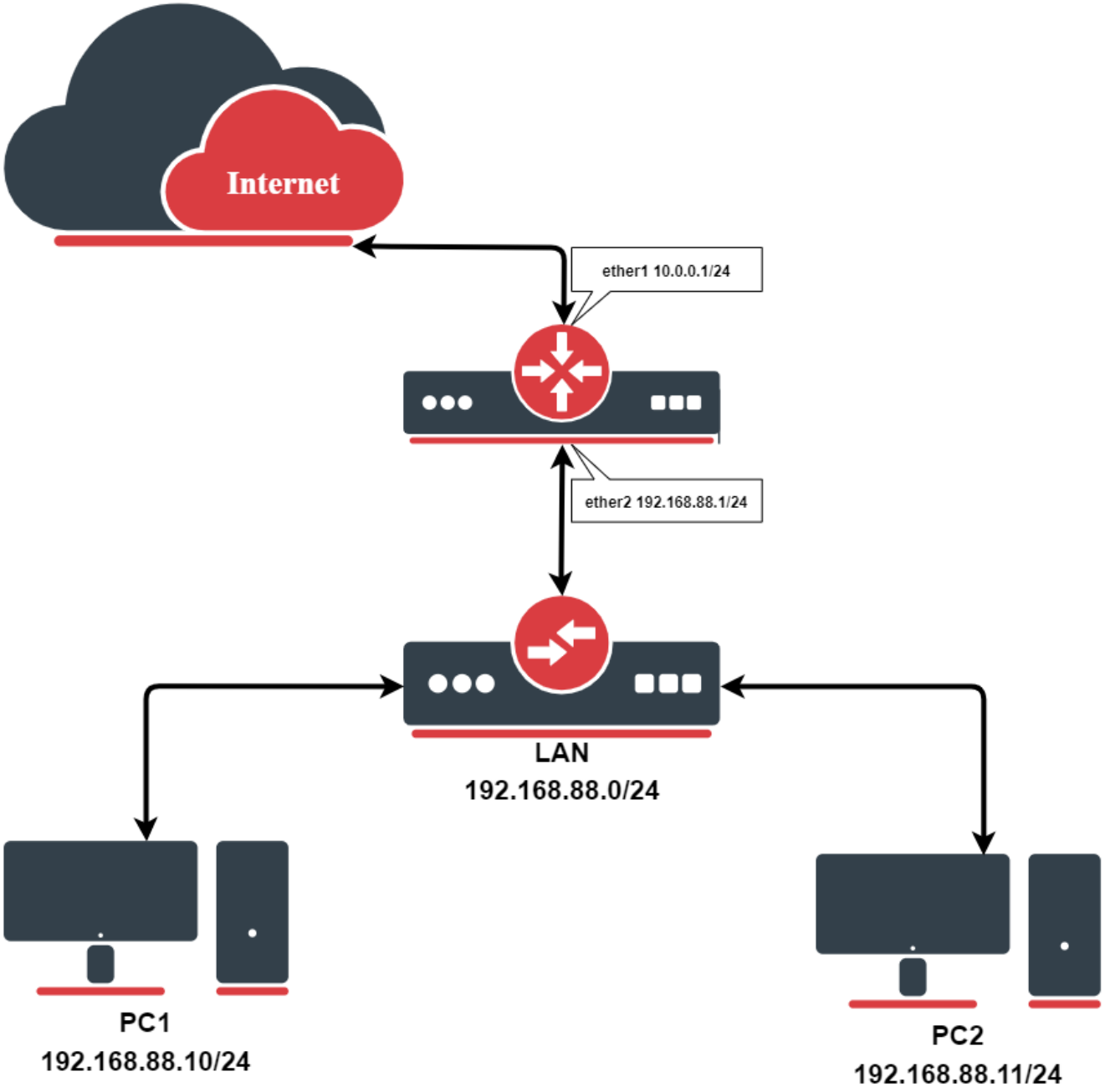
```
/ip upnp interfaces
```

Property	Description
interface (*string*; Default:)	Interface name on which uPnP will be running

type (*external* | *internal*; Default: **no**)	UPnP interface type: • external - the interface a global IP address is assigned to • internal - router's local interface the clients are connected to
forced-external-ip (*Ip*; Default:)	Allow specifying what public IP to use if the external interface has more than one IP available.

⚠ In more complex setups with VLANs, where the VLAN interface is considered as the LAN interface, the VLAN interface itself should be specified as the internal interface for UPnP to work properly.

Configuration Example



We have masquerading already enabled on our router:

```
[admin@MikroTik] ip upnp> /ip firewall src-nat print
Flags: X - disabled, I - invalid, D - dynamic
  0   chain=srcnat action=masquerade out-interface=ether1
[admin@MikroTik] ip upnp>
```

To enable the UPnP feature:

```
[admin@MikroTik] ip upnp> set enable=yes
[admin@MikroTik] ip upnp> print
                          enabled: yes
    allow-disable-external-interface: yes
                  show-dummy-rule: yes
[admin@MikroTik] ip upnp>
```

Now, all we have to do is to add interfaces:

```
[admin@MikroTik] ip upnp interfaces> add interface=ether1 type=external
[admin@MikroTik] ip upnp interfaces> add interface=ether2 type=internal
[admin@MikroTik] ip upnp interfaces> print
Flags: X - disabled
  #   INTERFACE TYPE
  0 X ether1    external
  1 X ether2    internal

[admin@MikroTik] ip upnp interfaces> enable 0,1
```

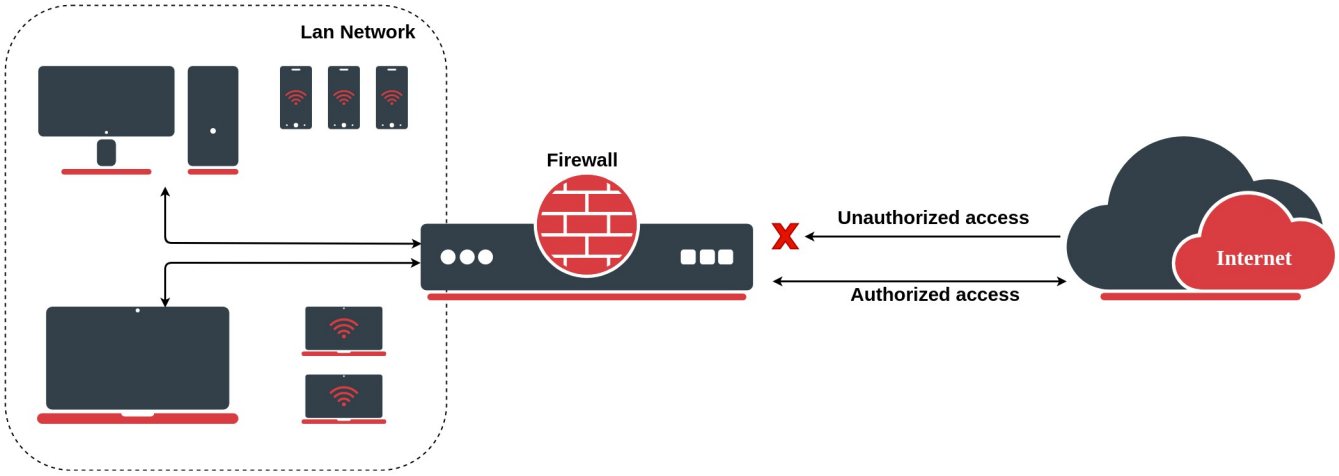
Now once the client from the internal interface side will send UPnP request dynamic NAT rules will be created on the router, example rules could look something similar to these:

```
[admin@MikroTik] > ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic

0 chain=srcnat action=masquerade out-interface=ether1

1 D ;;; upnp 192.168.88.10: ApplicationX
chain=dstnat action=dst-nat to-addresses=192.168.88.10 to-ports=55000 protocol=tcp
dst-address=10.0.0.1 in-interface=ether1 dst-port=55000

2 D ;;; upnp 192.168.88.10: ApplicationX
chain=dstnat action=dst-nat to-addresses=192.168.88.10 to-ports=55000 protocol=udp
dst-address=10.0.0.1 in-interface=ether1 dst-port=55000
```

Firewall and QoS Case Studies

Basic Concepts

Introduction



The firewall implements stateful (by utilizing connection tracking) and stateless packet filtering and thereby provides security functions that are used to manage data flow to, from, and through the router. Along with the Network Address Translation (NAT), it serves as a tool for preventing unauthorized access to directly attached networks and the router itself as well as a filter for outgoing traffic.

Network firewalls keep outside threats away from sensitive data available inside the network. Whenever different networks are joined together, there is always a threat that someone from outside of your network will break into your LAN. Such break-ins may result in private data being stolen and distributed, valuable data being altered or destroyed, or entire hard drives being erased. Firewalls are used as a means of preventing or minimizing the security risks inherent in connecting to other networks. A properly configured firewall plays a key role in efficient and secure network infrastructure deployment.

MikroTik RouterOS has a very powerful firewall implementation with features including:

- stateful packet inspection
- peer-to-peer protocols filtering
- traffic classification by:
  - source MAC address
  - IP addresses (network or list) and address types (broadcast, local, multicast, unicast)
  - port or port range
  - IP protocols
  - protocol options (ICMP type and code fields, TCP flags, IP options, and MSS)
  - interface the packet arrived from or left through
  - internal flow and connection marks
  - DSCP byte
  - packet content
  - rate at which packets arrive and sequence numbers
  - packet size

- packet arrival time

And much more!

How It works

The firewall operates by means of firewall rules. Each rule consists of two parts - the **matcher** which matches traffic flow against given conditions and the **action** which defines what to do with the matched packet.

RouterOS utilizes 5 sub-facilities of the firewall:

- Connection tracking
- Filters
- NAT
- Mangle
- RAW

Connection states

To completely understand firewall rules, first, you have to understand various states which might apply to a particular network packet. There are five connection states in RouterOS:

- **NEW** - The NEW state tells us that the packet is the first packet that we see. This means that the first packet that the conntrack module sees, within a specific connection, will be matched. For example, if we see an SYN packet and it is the first packet in a connection that we see, it will match;
- **ESTABLISHED -** The *ESTABLISHED* state has seen traffic in both directions and will then continuously match those packets. *ESTABLISHED* connections are fairly easy to understand. The only requirement to get into an *ESTABLISHED* state is that one host sends a packet and that it, later on, gets a reply from the other host. The *NEW* state will upon receipt of the reply packet to or through the firewall change to the *ESTABLISHE* state;
- **RELATED** - A connection is considered *RELATED* when it is related to another already *ESTABLISHED* connection. For a connection to be considered as *RELATED,* we must first have a connection that is considered *ESTABLISHED.* The *ESTABLISHED* connection will then spawn a connection outside of the main connection. The newly spawned connection will then be considered *RELATED,* for example, a packet that begins the FTP data connection;
- **INVALID** - The *INVALID* state means that the packet can't be identified or that it does not have any state. It is suggested to *DROP* everything in this state;
- **UNTRACKED** - A packet that was set to bypass connection tracking in the Firewall RAW table;

Configuration Example

Let's look at the basic firewall setup to protect the router. By default RouterOS firewall accepts everything, blocking is achieved by adding a filter rule to drop everything at the end of all rules. For out router we want to allow only ICMP, ssh, and Winbox and drop the rest:

```
/ip firewall filter
add chain=input connection-state=invalid action=drop comment="Drop Invalid connections"
add chain=input connection-state=established,related,untracked action=accept comment="Allow Established/Related
/Untracked connections"
add chain=input protocol=icmp action=accept comment="Allow ICMP"
add chain=input protocol=tcp ports=8291,22 action=accept comment="Allow Winbox and SSH"
add chain=input action=drop comment="Drop everything else"
```

RouterOS also allows filtering packets before connection tracking and selectively send only specific traffic to connection tracking. This allows us to significantly reduce the load on the CPU and mitigate DOS/DDoS attacks. Configuration of such rules is done in the RAW filtering table.

Additional */ip firewall filter* configuration examples find under the Building Your First Firewall section.

Connection Tracking

Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relate all of the packets which may make up that connection. NAT relies on this information to translate all related packets in the same way. Because of connection tracking, you can use stateful firewall functionality even with stateless protocols such as UDP.

A list of tracked connections can be seen in the */ip firewall connection* for ipv4 and */ipv6 firewall connection* for IPv6.

```
[admin@MirkoTik] /ip firewall connection> print
Flags: S - seen-reply, A - assured
# PR.. SRC-ADDRESS DST-ADDRESS TCP-STATE TIMEOUT
0 udp 10.5.8.176:5678 255.255.255.255:5678 0s
1 udp 10.5.101.3:646 224.0.0.2:646 5s
2 ospf 10.5.101.161 224.0.0.5 9m58s
3 udp 10.5.8.140:5678 255.255.255.255:5678 8s
4 SA tcp 10.5.101.147:48984 10.5.101.1:8291 established 4m59s

[admin@MirkoTik] /ipv6 firewall connection> print
Flags: S - seen reply, A - assured
# PRO.. SRC-ADDRESS DST-ADDRESS TCP-STATE
0 udp fe80::d6ca:6dff:fe77:3698 ff02::1
1 udp fe80::d6ca:6dff:fe98:7c28 ff02::1
2 ospf fe80::d6ca:6dff:fe73:9822 ff02::5
```

Based on connection table entries arrived packet can get assigned one of the connection states: **new, invalid, established, related,** or **untracked**.

There are two different methods when the packet is considered **new**. The first one is in the case of stateless connections (like UDP) when there is no connection entry in the connection table. The other one is in the case of a stateful protocol (TCP). In this case, a new packet that starts a new connection is always a TCP packet with an *SYN* flag.

If a packet is not new it can belong to either an *established* or *related* connection or not belong to any connection making it *invalid*. A packet with an *establis hed* state, as most of you already guessed, belongs to an existing connection from the connection tracking table. A *related* state is very similar, except that packet belongs to a connection that is related to one of the existing connections, for example, ICMP error packets or FTP data connection packets.

Connection state **notrack** is a special case when RAW firewall rules are used to exclude connection from connection tracking. This one rule would make all forwarded traffic bypass the connection tracking engine speeding packet processing through the device.

Any other packet is considered *invalid* and in most cases should be dropped.

Based on this information we can set a basic set of filter rules to speed up packet filtering and reduce the load on the CPU by accepting *established/related* packets, dropping *invalid* packets, and working on more detailed filtering only for *new* packets.

```
ip firewall filter
add chain=input connection-state=invalid action=drop comment="Drop Invalid connections"
add chain=input connection-state=established,related,untracked action=accept comment="Allow Established/Related
/Untracked connections
```

⚠ Such a rule set must not be applied on routers with asymmetric routing, because asymmetrically routed packets may be considered invalid and dropped.

FastTrack

IPv4 FastTrack handler is automatically used for marked connections. Use firewall action "fasttrack-connection" to mark connections for FastTrack. Currently, only TCP and UDP connections can be actually FastTracked (even though any connection can be marked for FastTrack). IPv4 FastTrack handler supports NAT (SNAT, DNAT, or both).

Note that not all packets in a connection can be FastTracked, so it is likely to see some packets going through a slow path even though the connection is marked for FastTrack. This is the reason why fasttrack-connection is usually followed by an identical "*action=accept*" rule. FastTrack packets bypass firewall, connection tracking, simple queues, queue tree with *parent=global*, IP accounting, IPSec, hotspot universal client, VRF assignment, so it is up to the administrator to make sure FastTrack does not interfere with other configuration.

Requirements

IPv4 FastTrack is active if the following conditions are met:

- no mesh, metarouter interface configuration;
- sniffer, torch, or traffic generator is not running;
- */tool mac-scan* is not actively used;
- */tool ip-scan* is not actively used;
- FastPath and Route cache is enabled under *IP/Settings*

Example

For example, for SOHO routers with factory default configuration, you could FastTrack all LAN traffic with this one rule placed at the top of the Firewall Filter. The same configuration accept rule is required:

```
/ip firewall filter add chain=forward action=fasttrack-connection connection-state=established,related
/ip firewall filter add chain=forward action=accept connection-state=established,related
```

- Connection is FastTracked until the connection is closed, timed-out, or router is rebooted.
- Dummy rules will disappear only after FastTrack firewall rules will be deleted/disabled and the router rebooted.
- While FastPath and FastTrack both are enabled on the device only one can be active at a time.

Queues (except Queue Trees parented to interfaces), firewall filter, and mangle rules will not be applied for FastTracked traffic.

Services

This section lists protocols and ports used by various MikroTik RouterOS services. It helps you to determine why your MikroTik router listens to certain ports, and what you need to block/allow in case you want to prevent or grant access to certain services.

The default services are:

Property	Description
telnet	Telnet service
ftp	FTP service
www	Webfig http service
ssh	SSH service
www-ssl	Webfig HTTPS service
api	API service
winbox	Responsible for Winbox tool access, as well as Tik-App smartphone app and Dude probe
api-ssl	API over SSL service

Properties

Note that it is not possible to add new services, only existing service modifications are allowed.

Property	Description
address (*IP address/netmask | IPv6/0..128*; Default:)	List of IP/IPv6 prefixes from which the service is accessible.

certificate (*name*; default: **none**)	The name of the certificate used by a particular service. Applicable only for services that depend on certificates (*www-ssl, api-ssl*)
name (*name*; default: **none**)	Service name
port (*integer: 1..65535*; Default:)	The port particular service listens on

To restrict Winbox service access to the device only from the **192.168.88.0/24** subnet, we have to configure the following:

```
[admin@MikroTik] > ip service set [find name~"winbox"] address=192.168.88.0/24
[admin@MikroTik] > ip service print
Flags: X - disabled, I - invalid
# NAME PORT ADDRESS CERTIFICATE
0 telnet 23
1 XI ftp 21
2 XI www 80
3 ssh 22
4 XI www-ssl 443 none
5 XI api 8728
6 winbox 8291 192.168.88.0/24
7 XI api-ssl 8729 none
```

We recommend disabling unused services.

Address List

Firewall address lists allow a user to create lists of IP addresses grouped together under a common name. Firewall filter, Mangle, and NAT facilities can then use those address lists to match packets against them. The address list records can also be updated dynamically via the *action=add-src-to-address-list* or *action=add-dst-to-address-list* items found in NAT, Mangle, and Filter facilities.
Firewall rules with action *add-src-to-address-list* or *add-dst-to-address-list* works in passthrough mode, which means that the matched packets will be passed to the next firewall rules. A basic example of a dynamically created address-list:

```
[admin@MirkoTik] > ip firewall address-list add address=www.mikrotik.com list=MikroTik
[admin@MirkoTik] > ip firewall address-list print
Flags: X - disabled, D - dynamic
# LIST ADDRESS CREATION-TIME TIMEOUT
0 MikroTik www.mikrotik.com oct/09/2019 14:53:14
1 D ;;; www.mikrotik.com
MikroTik 159.148.147.196 oct/09/2019 14:53:14
```

Layer7-protocol

Layer7-protocol is a method of searching for patterns in ICMP/TCP/UDP streams. It collects the first 10 packets of a connection or the first 2KB of a connection and searches for the pattern in the collected data. If the pattern is not found in the collected data, the matcher stops inspecting further. Allocated memory is freed and the protocol is considered unknown. You should take into account that a lot of connections will significantly increase memory and CPU usage. To avoid this, add regular firewall matches to reduce the amount of data passed to layer-7 filters repeatedly.

An additional requirement is that the layer7 matcher must see both directions of traffic (incoming and outgoing). To satisfy this requirement l7 rules should be set in the forward chain. If a rule is set in the input/prerouting chain then the same rule must be also set in the output/postrouting chain, otherwise, the collected data may not be complete resulting in an incorrectly matched pattern.

In this example, we will use a pattern to match RDP packets.

```
/ip firewall layer7-protocol add name=rdp regexp="rdpdr.*cliprdr.*rdpsnd"
```

If the Layer7 matcher recognizes the connection, then the rule marks this connection as its "own" and other rules do not look at this connection anymore even if the two firewall rules with Layer7 matcher are identical.

When a user uses HTTPS, Layer7 rules will not be able to match this traffic. **Only unencrypted HTTP can be matched**.

Packet Flow in RouterOS

Understanding Packet Flow

More advanced firewall setups, or complicated tasks, such as traffic prioritization, routing policies, where it is necessary to utilize more than one RouterOS facility, require knowledge: How do these facilities work together? What happens when and why?

RouterOS packet flow diagram and flow examples will try to answer these questions.

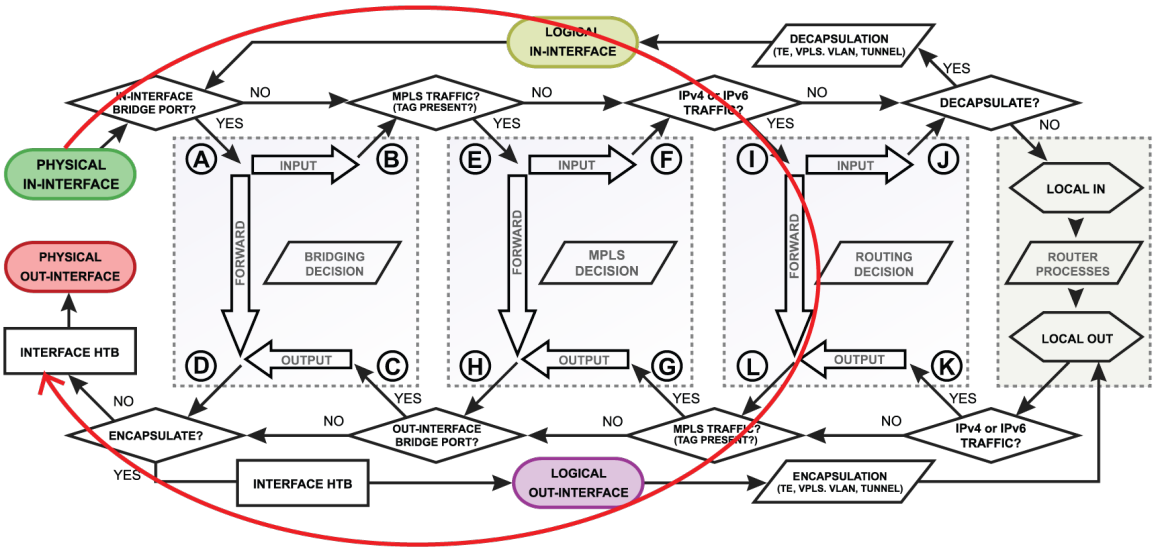
It would be very complicated to represent what is going on with the packet in one diagram, therefore a packet flow diagram is divided into three parts:

- overall diagram;
- detailed bridging, routing, and MPLS flow diagram;
- a diagram that shows what facilities and in what order are included in prerouting, input, forward, output, and postrouting.

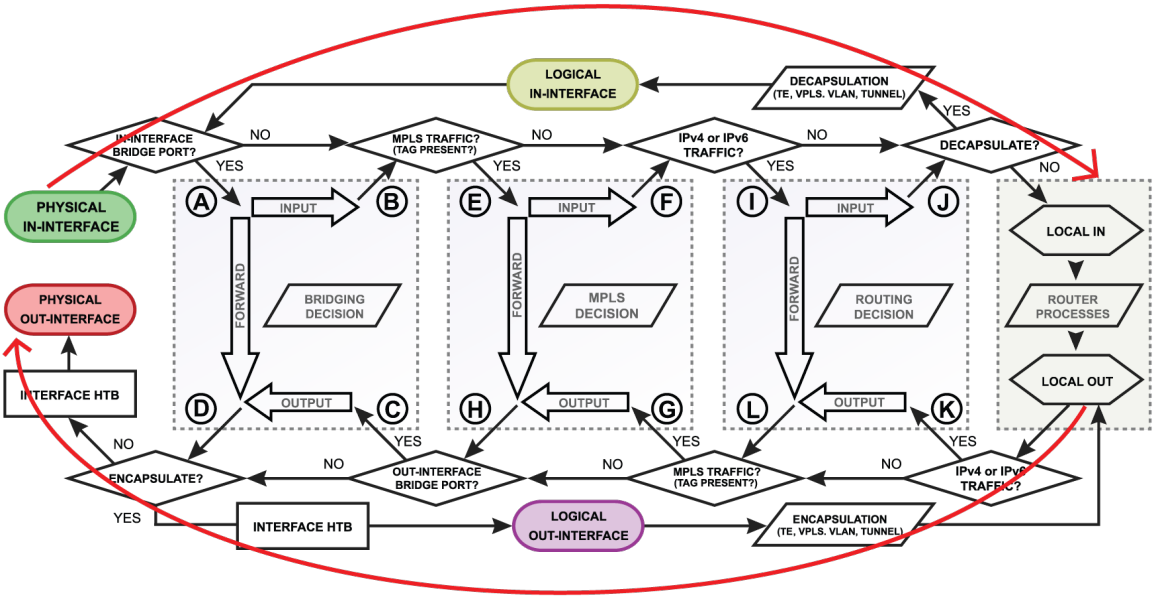
Overall Packetflow Diagram

Let's look at the overall diagram. It looks complicated at first, but after we go through the diagram with examples it will become much clearer.

There are 4 boxes in the center of the diagram: Bridging, Routing, Mpls decisions, and local router processes. So for example, if the packet needs to be routed over the router, a packet will flow as illustrated in the image below. Without looking deeper into each facility, the packet enters the in-interface, the router determines that it is IP traffic and needs to be routed, the packet goes through all routing processes and exits the out-interface.



Let's take a look at another example that will illustrate what happens if the packet's destination is a router. For example, the in-interface receives ICMP (ping) packet, its destination is the router itself, so the packet will go for *local-in* processing. After the packet is processed ICMP (ping) reply is generated inside the router *(local-out* processing) and will be sent out over the out-interface.



A simple explanation of each box before we go further with examples:

- **physical in-interface** - the starting point of the packet received by the router;
- **logical in-interface** - the starting point of the decapsulated packet (from tunnels, IPsec, etc);
- **local in** - the last point of a packet destined to router itself;
- **interface HTB (Hierarchical Token Bucket)** - interface queue;
- **physical out-interface** - last point of the packet before it is actually sent out;
- **logical out-interface** - last point of the packet before encapsulation (to tunnels, IPsec, etc);
- **local out** - the starting point of a packet generated by the router;

Now it is time to take a deeper look at what is happening inside bridging, MPLS, and routing flows.

A simple explanation of each box before we go further with examples:

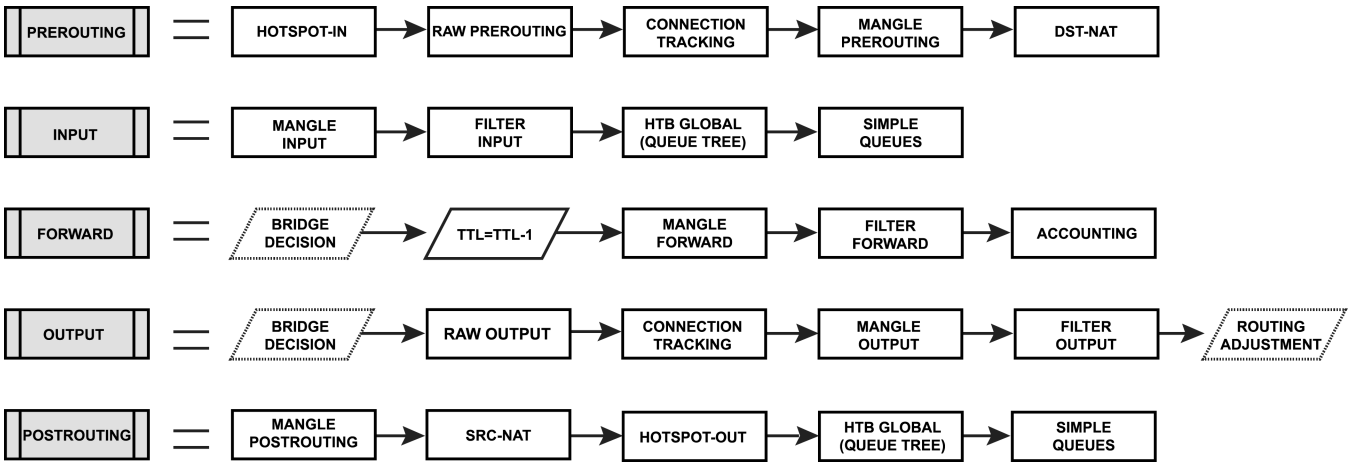
- **routing decision** - go through routes in the routing table to find a match for the destination IP address of the packet. When a match is found - the packet will be processed further, in case of no match - the packet will be discarded.;
- **mpls decision** - what to do with the packet based on MPLS forwarding tables;
- **bridging decision** - bridge goes through the MAC address table to find a match for the destination MAC address of the packet. When a match is found - the packet will be processed further, in case of no match - multiple copies of the packet will be created and packets will be flooded (sent out via all bridge ports). A single packet copy will also reach a bridge input chain as the bridge interface itself is one of the many destinations. When using `vlan-filtering=yes`, packets that are not allowed due to the "/interface bridge vlan" table, will be dropped at this stage.
- **use-ip-firewall** - whether a '*use-ip-firewall*' option is enabled in bridge settings;
- **ipsec-policy** - whether a packet matches any of configured IPsec policies;

Chains

RouterOS consist of a few default chains. These chains allow you to filter packets at various points:

- The **PREROUTING** chain: Rules in this chain apply to packets as they just arrive on the network interface. This chain is present in the *nat*, *mangle* and *raw* tables.
- The **INPUT** chain: Rules in this chain apply to packets just before they're given to a local process. This chain is present in the *mangle* and *filter* tables.
- The **OUTPUT** chain: The rules here apply to packets just after they've been produced by a process. This chain is present in the *raw*, *mangle*, nat, and *filter* tables.
- The **FORWARD** chain: The rules here apply to any packets that are routed through the current host. This chain is only present in the *mangle* and *filter* tables.
- The **POSTROUTING** chain: The rules in this chain apply to packets as they just leave the network interface. This chain is present in the *nat* and *mangle* tables.

Each of the prerouting, input, forward, output, and postrouting blocks contains even more facilities, which are illustrated in the third part of the packet flow diagram:



A simple explanation of each box before we go further with examples:

- **Hotspot-in** - allows to capture traffic that otherwise would be discarded by connection tracking - this way our Hotspot feature is able to provide connectivity even if networks settings are an incomplete mess ;
- **RAW Prerouting** - RAW table prerouting chain;
- **Connection tracking** - packet is processed by connection tracking;
- **Mangle Prerouting** - Mangle prerouting chain;
- **Mangle Input** - Mangle input chain;
- **Filter Input** - Firewall filter input chain;
- **HTB Global** - Queue tree;
- **Simple Queues** - is a feature that can be used to limit traffic for a particular target;
- **TTL** - indicates an exact place where the Time To Live (TTL) of the routed packet is reduced by 1 if TTL becomes 0, a packet will be discarded;
- **Mangle Forward** - Mangle forward chain;
- **Filter Forward** - Filter forward chain;
- **Accounting** - Authentication, Authorization, and Accounting feature processing;
- **RAW Output** - RAW table output chain;
- **Mangle Output** - Mangle output chain;
- **Filter Output** - Firewall filter output chain;
- **Routing Adjustment** - this is a workaround that allows to set up policy routing in mangle chain output (routing-mark) ;

- **Mangle Postrouting** - Mangle postrouting chain;
- **Src Nat** - Network Address Translation srcnat chain;
- **Dst Nat** - Network Address Translation dstnat chain;
- **Hotspot-out** - undo all that was done by hotspot-in for the packets that are going back to the client;

Flow of Routed Packet

Forward

Now, let's take our first example where the packet gets routed over the router and look deeper through what facilities packet goes:

We already learned that packet goes into the in-interface, the router determines that it is an IP packet and needs to be routed, and here starts the complicated process:

1. The packet enters prerouting processing:
    a. check if there is a hotspot and modify the packet for hotspot use
    b. process packet through RAW prerouting chain;
    c. send the packet through connection tracking;
    d. process packet through Mangle prerouting chain;
    e. process packet through NATs dst-nat chain;
2. Run packet through routing table to make routing decision;
3. The packet enters the forward process;
    a. check TTL value;
    b. process packet through Mangle forward chain;
    c. process packet through the Filter forward chain;
    d. send the packet to accounting processes;

4. A packet enters postrouting process;
    a. process packet through Mangle postrouting chain;
    b. process packet through NATs src-nat chain;
    c. if there is a hotspot undo any modifications made in hotspot-in;
    d. process packet through queue tree (HTB Global);
    e. process packet through simple queues;

5. Check if there is IPsec and then process through IPsec policies;

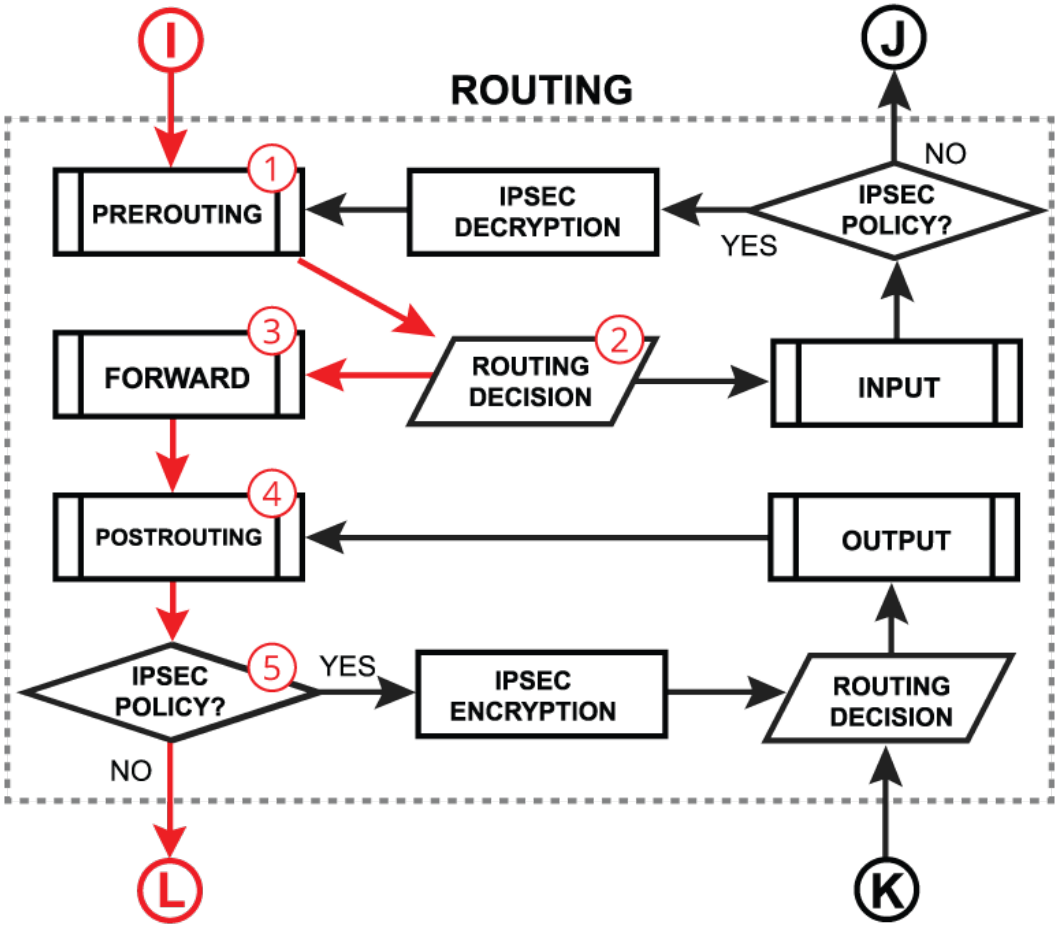
Input

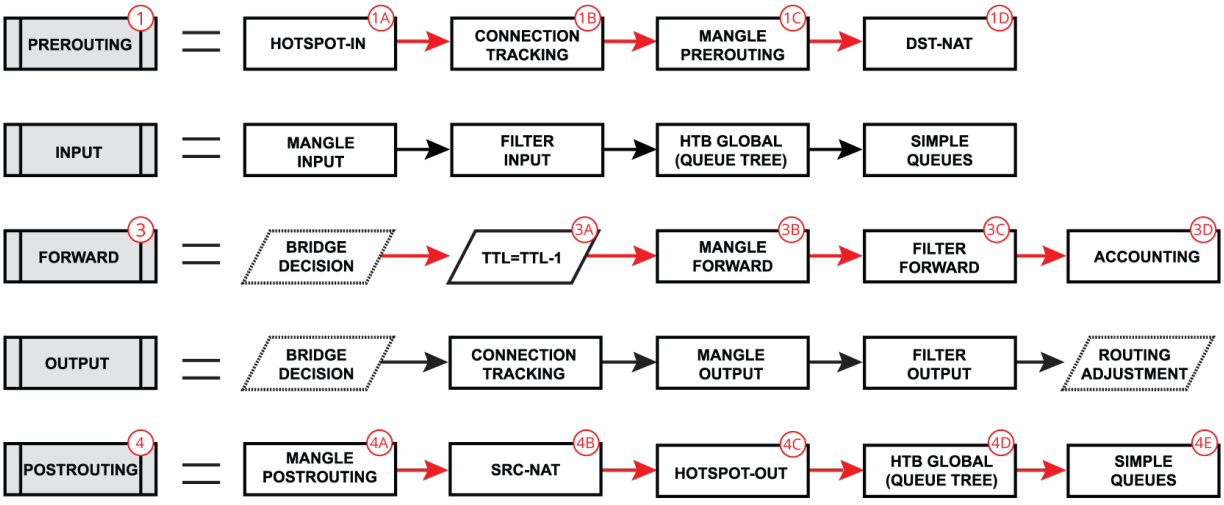
We already learned that packet goes into the in-interface, the router determines that it is an IP packet and needs to be routed, and here starts the complicated process:

1. A very similar process happens when a packet's destination is a router (routing input): Packet enters prerouting processing:
    a. - check if there is a hotspot and modify the packet for hotspot use;
    b. - process packet through RAW prerouting chain;
    c. - send a packet through connection tracking;
    d. - process packet through Mangle prerouting chain;
    e. - process packet through NATs dst-nat chain;

2. Run packet through routing table to make routing decision;

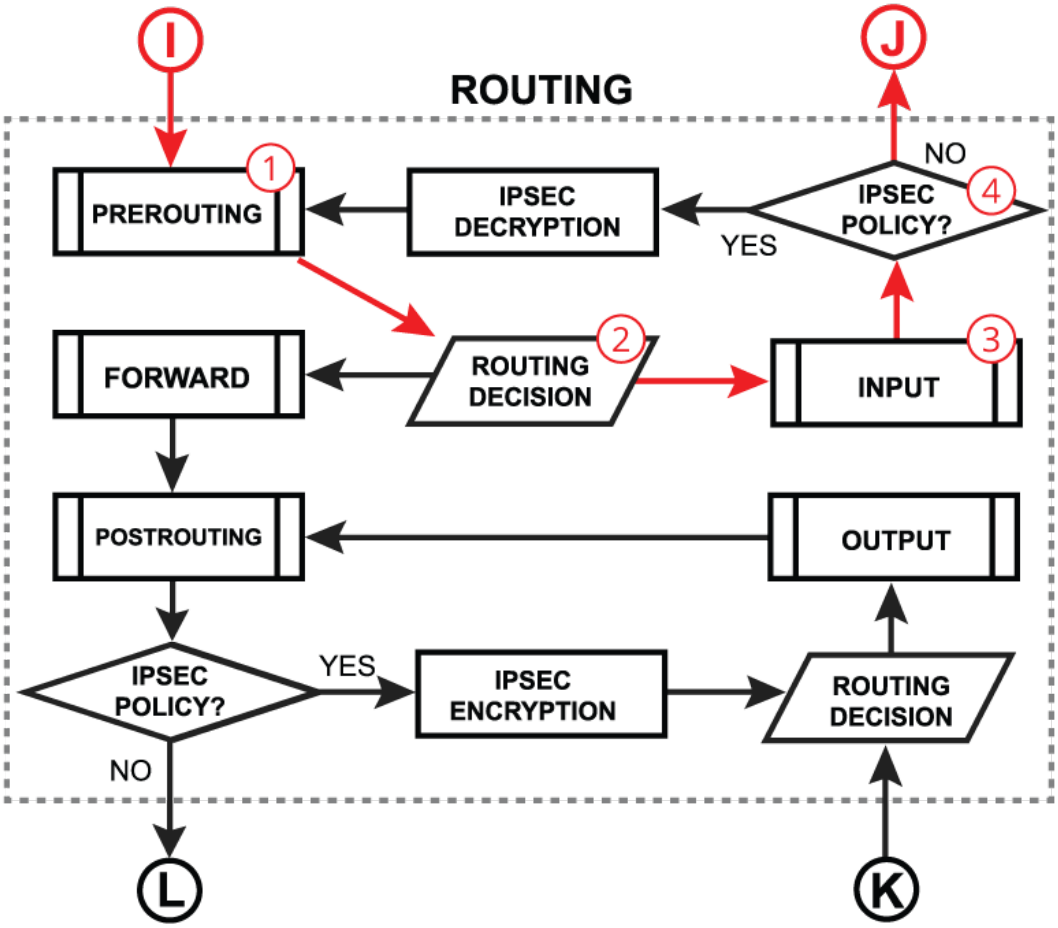
3. A Packet enters the input process;
    a. - process packet through Mangle input chain;
    b. - process packet through Filter input chain;
    c. - process packet through queue tree (HTB Global);

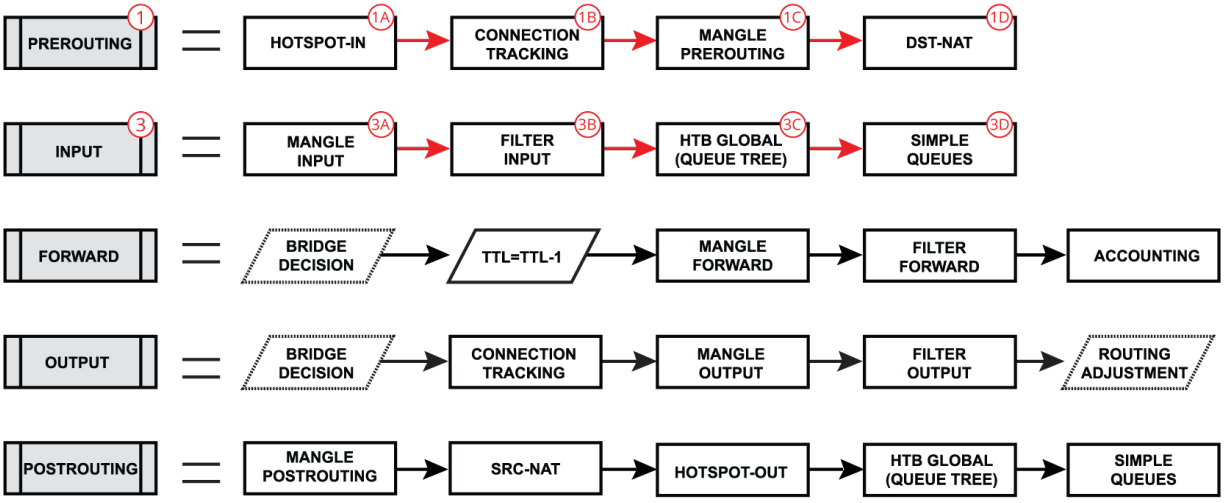




d. - process packet through simple queues;

4. Check if there is IPsec and then process through IPsec policies.





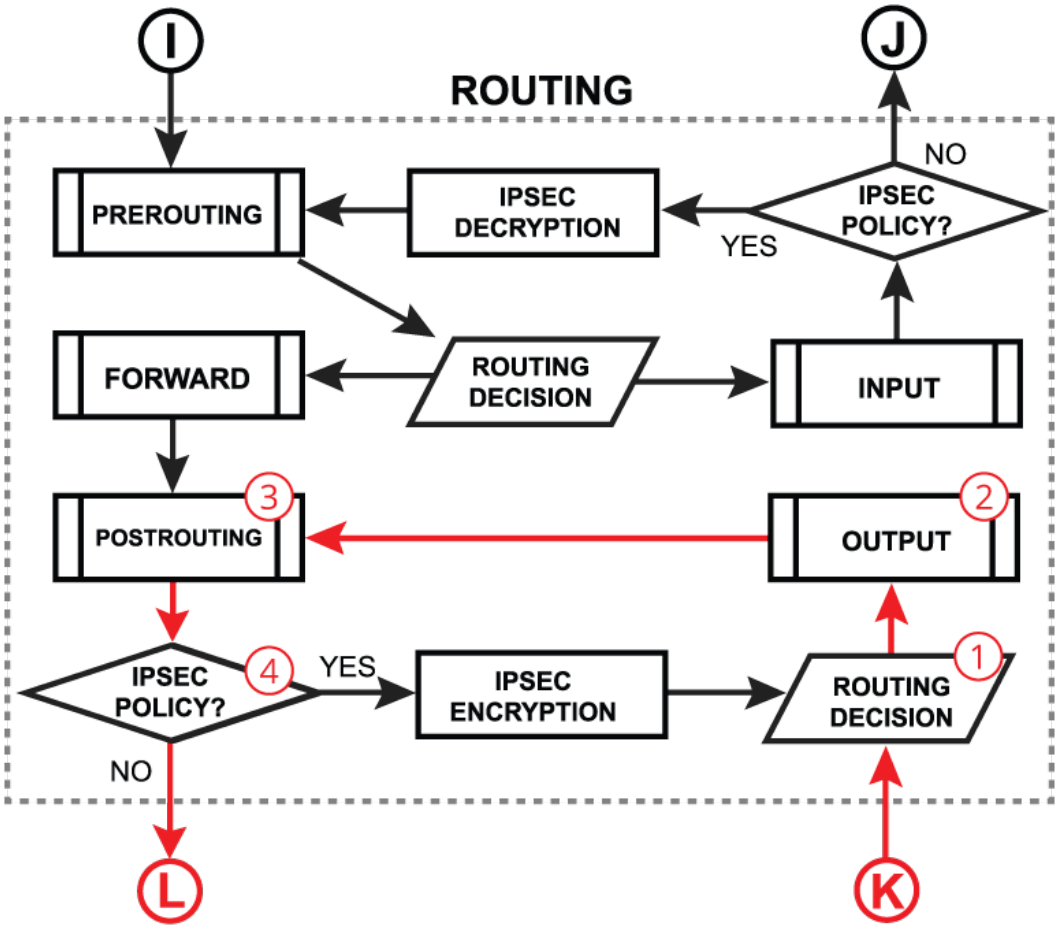
Or when a packet is originated from the router (routing output):

1. The packet is originated from the router itself
    a. the packet goes through the routing table to make a routing decision

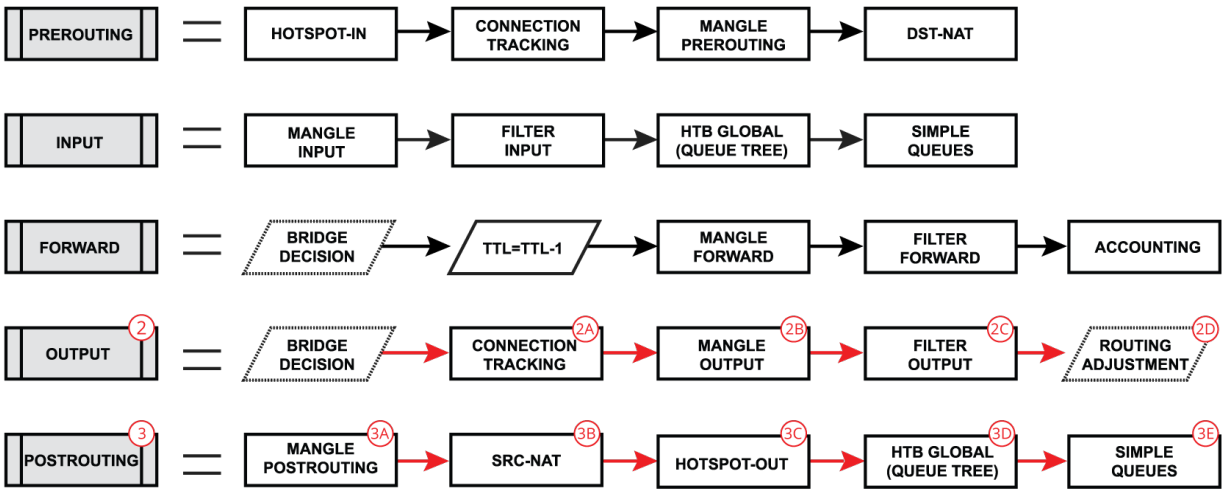
2. A packet enters the output process
    a. process packet through the Bridge decision;
    b. send the packet through connection tracking;
    c. process packet through the Mangle output chain;
    d. process packet through the Filter output chain;
    e. send the packet to routing adjustment (policy routing)

3. The packet enters postrouting process;
    a. - process packet through Mangle postrouting chain;
    b. - process packet through NATs src-nat chain;
    c. - if there is a hotspot undo any modifications made in hotspot-in;
    d. - process packet through queue tree (HTB Global);
    e. - process packet through simple queues;

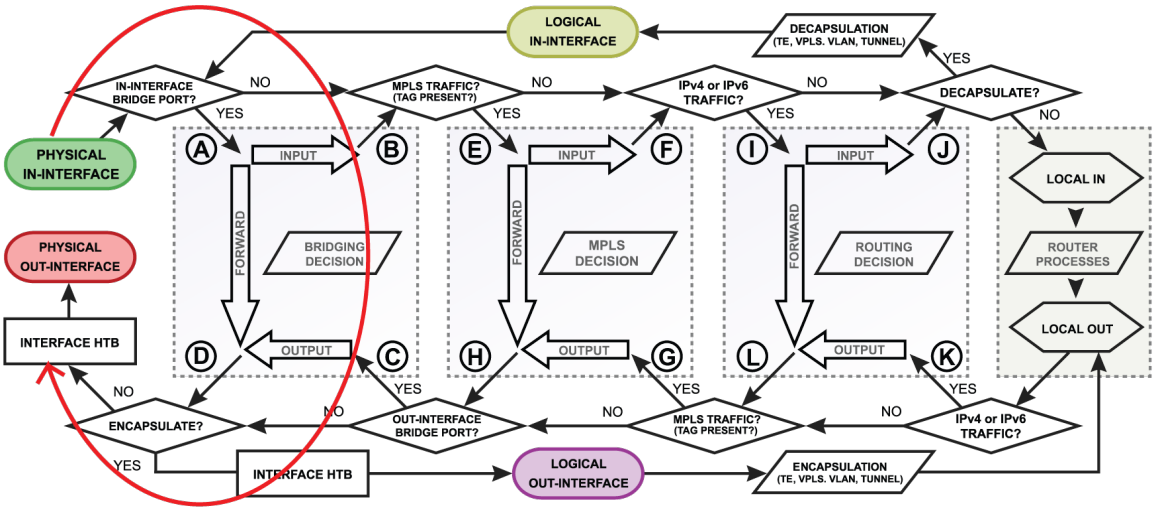
4. Check if there is IPsec and then process through IPsec policies;



Flow of Bridged Packet



Below is discussed a general bridging process in RouterOS. Most of the packets will always follow the same processing path, but in certain configurations (e.g. with enabled VLAN filtering, horizon, STP, DHCP, or IGMP snooping) some packets can be treated differently. Please visit the bridging manual for more specific information.

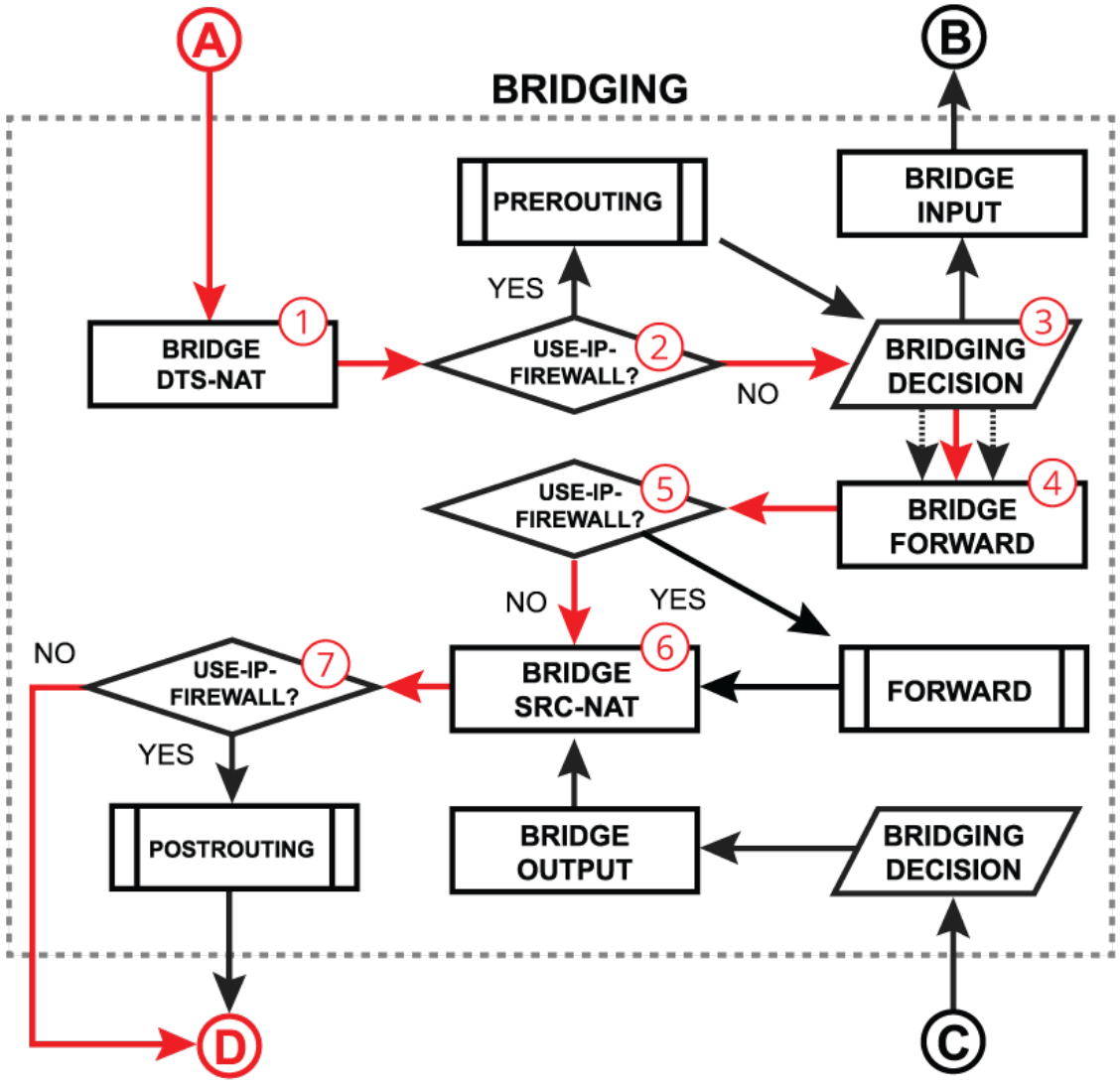


Bridge Forward

Bridge forward is a process that takes place when a packet is forwarded from one bridge port to another, essentially connecting multiple devices on the same network. After receiving a packet on the in-interface, the device determines that the in-interface is a bridge port, so it gets passed through the bridging process:

1. A packet goes through the bridge NAT dst-nat chain, where MAC destination and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
2. Checks whether the use-ip-firewall option is enabled in the bridge settings;
3. Run packet through the bridge host table to make a forwarding decision. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), gets multiplied per bridge port and then processed further in the bridge forward chain. When using `vlan-filtering=yes`, packets that are not allowed due to the "/interface bridge vlan" table, will be dropped at this stage.

4. A packet goes through the bridge filter forward chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;
5. Checks whether the use-ip-firewall option is enabled in the bridge settings;
6. A packet goes through the bridge NAT src-nat chain, where MAC source and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
7. Checks whether the use-ip-firewall option is enabled in the bridge settings;



For RouterOS v6:
When bridge `vlan-filtering` is enabled, received untagged packets might get encapsulated into the VLAN header before the "DST-NAT" block, which means these packets can be filtered using the `mac-protocol=vlan` and `vlan-encap` settings. Encapsulation can happen if the outgoing interface has `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged`.

Tagged packets might get decapsulated on the "BRIDGING DECISION" block, which means these packets will no longer match the `mac-protocol=vlan` and `vlan-encap` settings. Decapsulation can happen if the packet's VLAN ID matches the outgoing port's untagged VLAN membership.

For RouterOS v7 and newer:

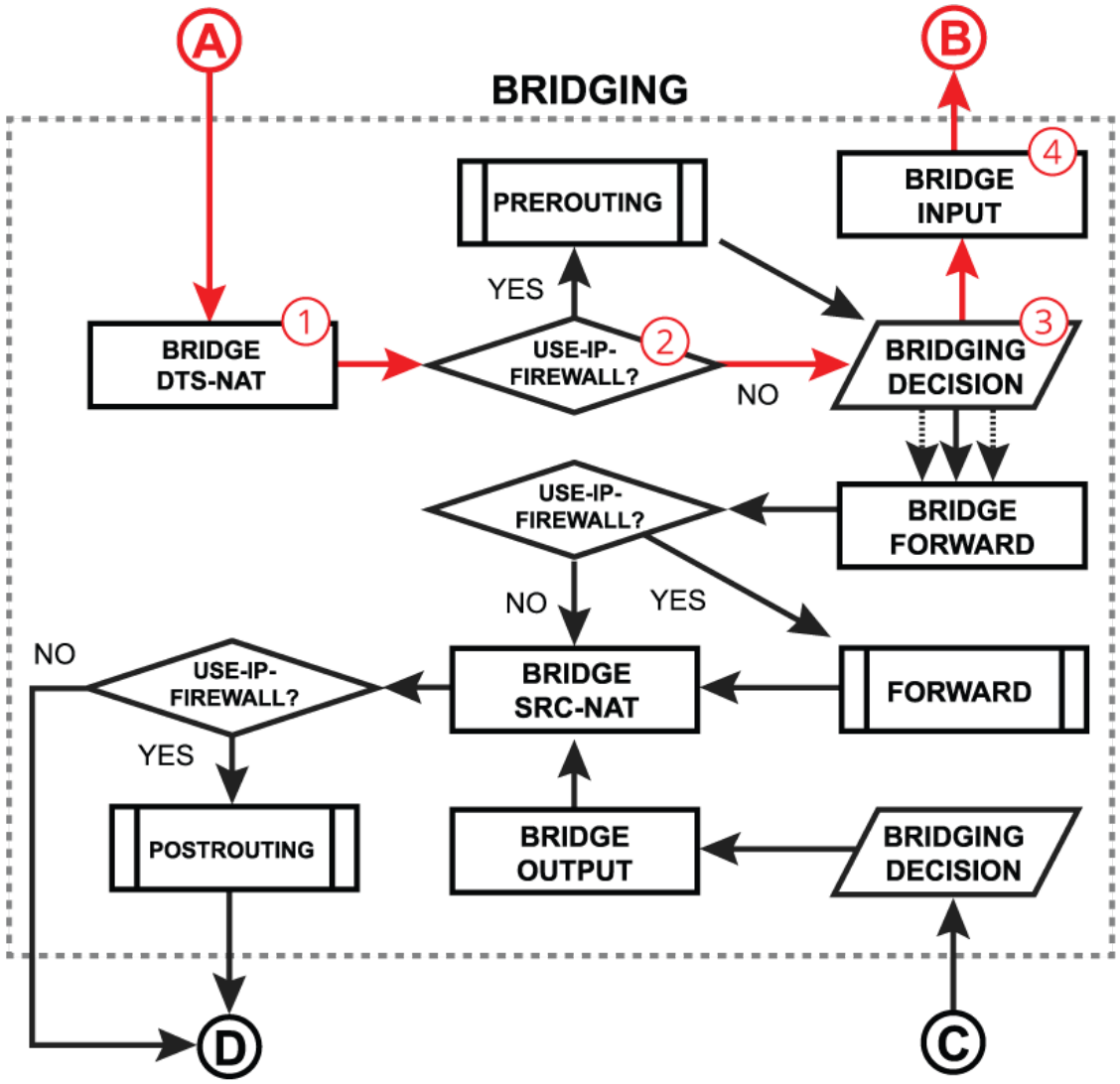
When bridge `vlan-filtering` is enabled, received untagged packets might get encapsulated into the VLAN header on the "BRIDGING-DECISION" block, which means these packets can be filtered using the `mac-protocol=vlan` and `vlan-encap` settings. Encapsulation can happen if the outgoing interface has `frame-types` set to `admit-all` or `admit-only-untagged-and-priority-tagged`.

Tagged packets might get decapsulated on the "BRIDGING DECISION" block, which means these packets will no longer match the `mac-protocol=vlan` and `vlan-encap` settings. Decapsulation can happen if the packet's VLAN ID matches the outgoing port's untagged VLAN membership.

Bridge Input

Bridge input is a process that takes place when a packet is destined for the bridge interface. Most commonly this happens when you need to reach some services that are running on the bridge interface (e.g. a DHCP server) or you need to route traffic to other networks. The very first steps are similar to the bridge forward process - after receiving a packet on the in-interface, the device determines that the in-interface is a bridge port, so it gets passed through the bridging process:

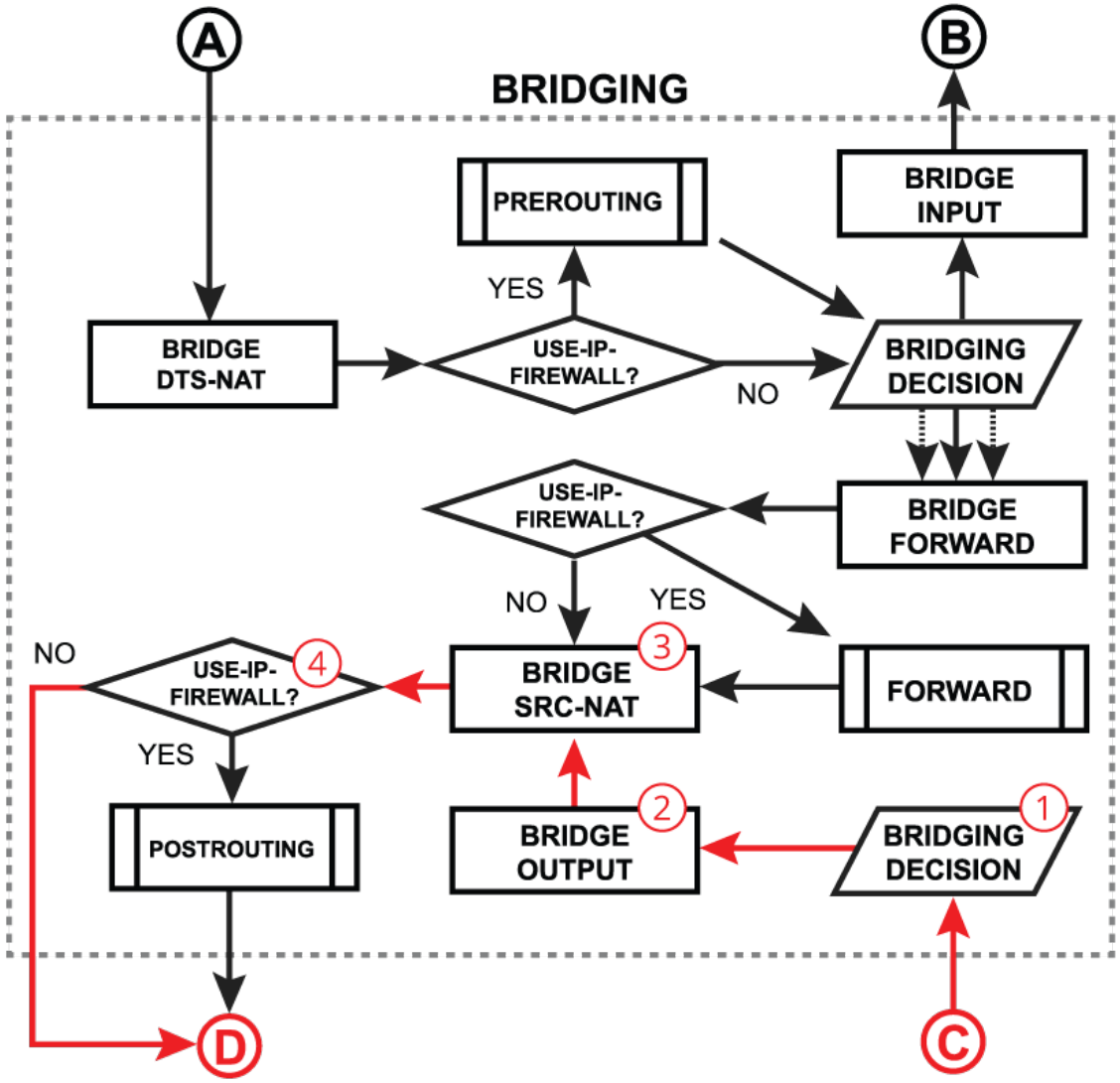
1. A packet goes through the bridge NAT dst-nat chain, where MAC destination and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
2. Checks whether the use-ip-firewall option is enabled in the bridge settings;
3. Run packet through the bridge host table to make a forwarding decision. A packet where the destination MAC address matches the bridge MAC address will be passed to the bridge input chain. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), also reaches the bridge input chain as the bridge interface itself is one of the many destinations;
4. A packet goes through the bridge filter input chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;



Bridge Output

Bridge output is a process that takes place when a packet should exit the device through one or multiple bridge ports. Most commonly this happens when a bridge interface itself tries to reach a device connected to a certain bridge port (e.g. when a DHCP server running on a bridge interface is responding to a DHCP client). After a packet is processed on other higher-level RouterOS processes and the device finally determines that the output interface is a bridge, the packet gets passed through the bridging process:

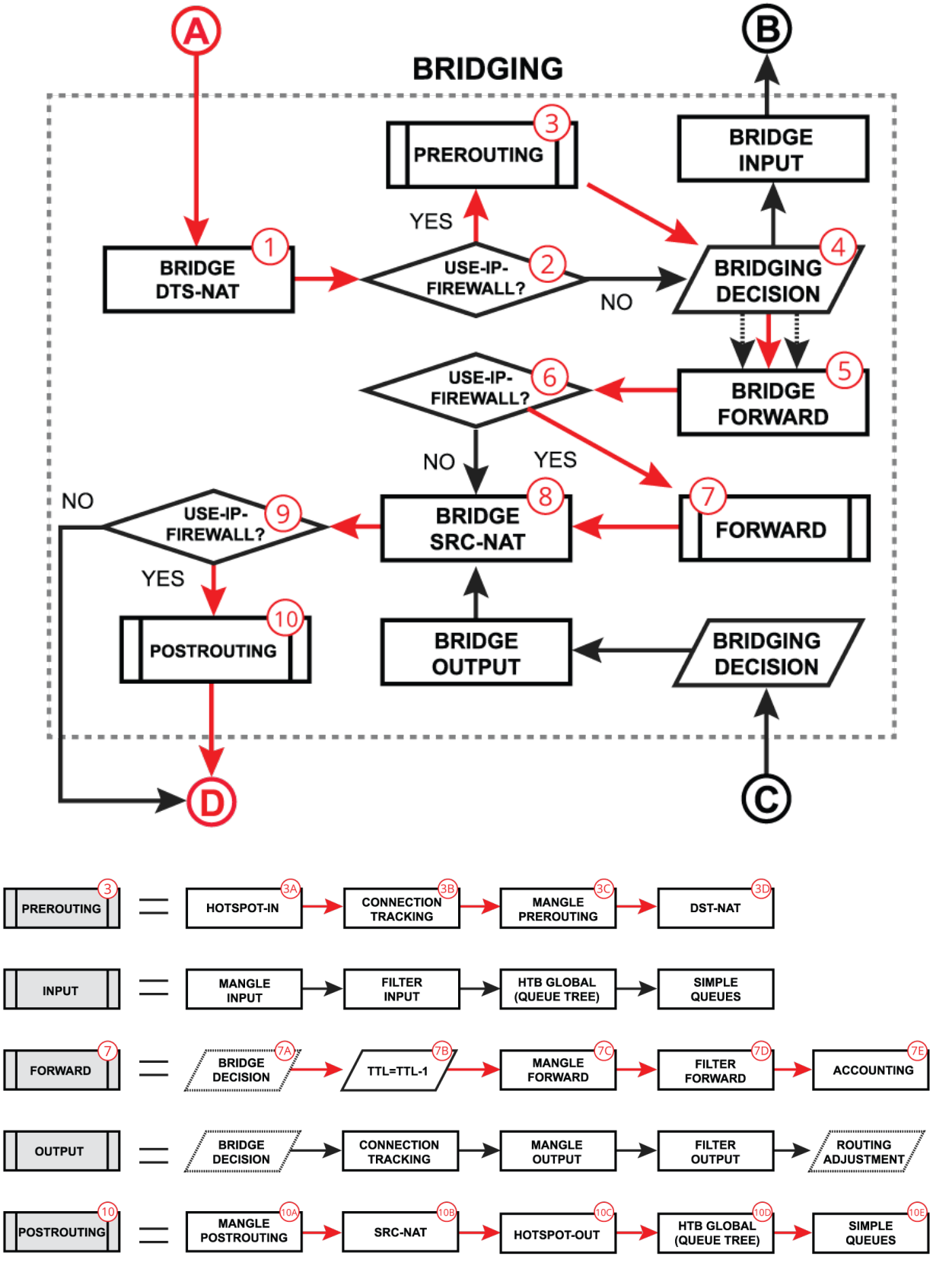
1. Run packet through the bridge host table to make a forwarding decision. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic), gets multiplied per bridge port and then processed further in the bridge output chain.
2. A packet goes through the bridge filter output chain, where priority can be changed or the packet can be simply accepted, dropped, or marked;
3. A packet goes through the bridge NAT src-nat chain, where MAC source and priority can be changed, apart from that, a packet can be simply accepted, dropped, or marked;
4. Checks whether the use-ip-firewall option is enabled in the bridge settings;



Forward With Firewall Enabled

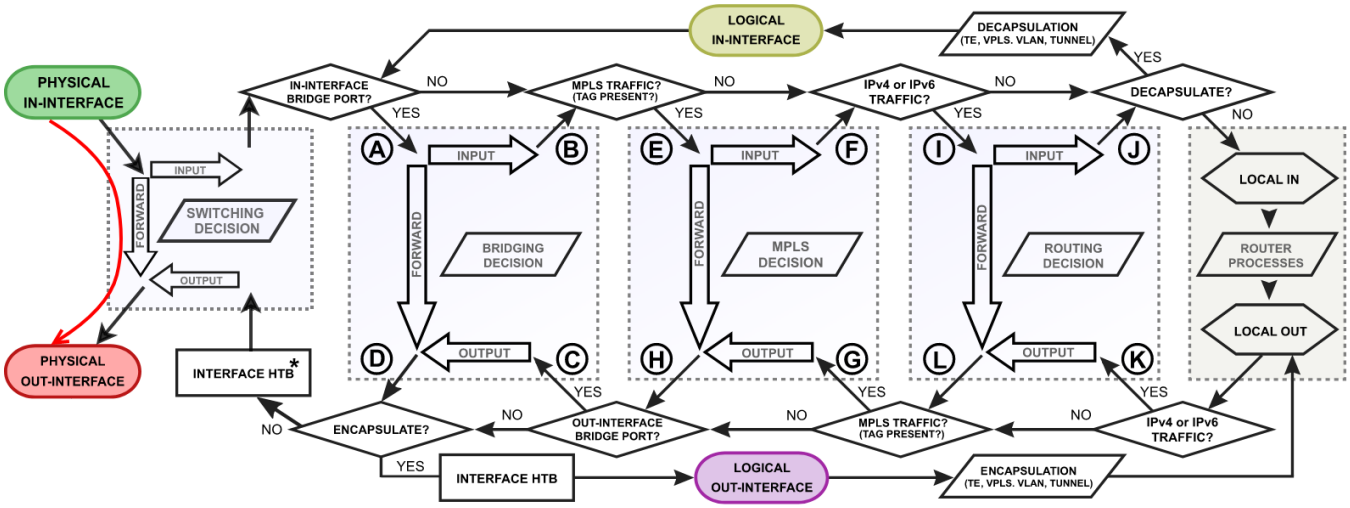
In certain network configurations, you might need to enable additional processing on routing chains for bridged traffic, for example, to use simple queues or an IP firewall. This can be done when the use-ip-firewall is enabled under the bridge settings. Note that additional processing will consume more CPU resources to handle these packets. All the steps were already discussed in previous points, below is a recap:

1. A packet goes through the bridge NAT dst-nat chain;
2. With the use-ip-firewall option enabled, the packet will be further processed in the prerouting chain;
3. A packet enters prerouting processing;
4. Run packet through the bridge host table to make forwarding decision;
5. A packet goes through the bridge filter forward chain;
6. With the use-ip-firewall option enabled, the packet will be further processed in the routing forward chain;
7. A packet enters routing forward processing;
8. A packet goes through the bridge NAT src-nat chain;
9. With the use-ip-firewall option enabled, the packet will be further processed in the postrouting chain;
10. A packet enters postrouting processing;

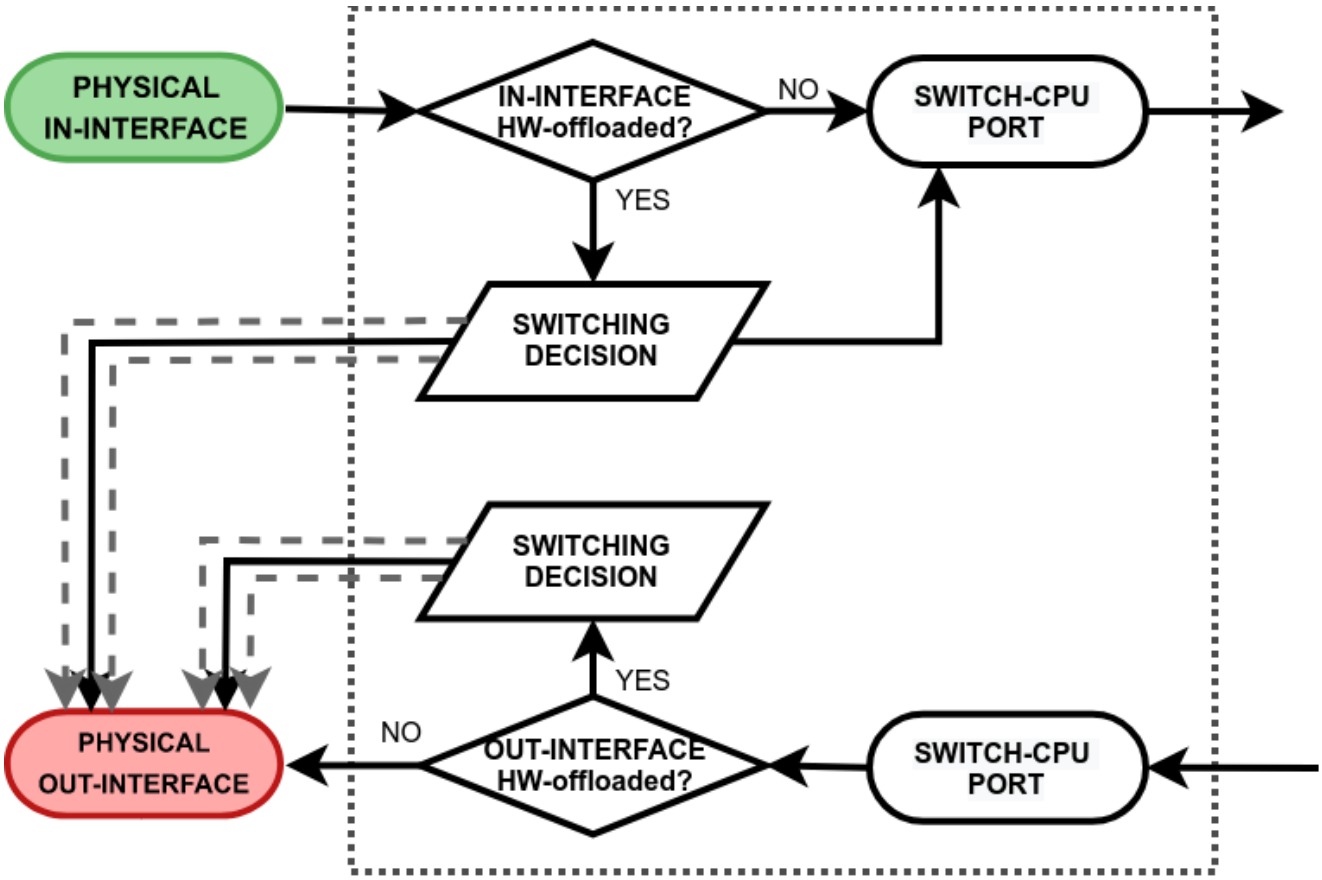


Flow of Hardware Offloaded Packet

On the previous topic, we solely discussed a software bridging that requires the main CPU processing to forward packets through the correct bridge port. Most of the MikroTik devices are equipped with dedicated switching hardware, the so-called switch chip or switch ASIC. This allows us to offload some of the bridging functions, like packet forwarding between bridge ports or packet filtering, to this specialized hardware chip without consuming any CPU resources. In RouterOS, we have named this function Bridge Hardware (HW) Offloading. Different MikroTik devices might have different switch chips and each chip has a different set of features available, so make sure to visit this article to get more details - Bridge Hardware Offloading.



Interface HTB will not work correctly when the out-interface is hardware offloaded and the bridge Fast Path is not active.



- **switching decision** - widely depends on the switch model. This block controls all the switching-related tasks, like host learning, packet forwarding, filtering, VLAN tagging/untagging, etc. Certain switch configurations can alter the packet flow;
- **switch-cpu port** - a special purpose switch port for communication between the main CPU and other switch ports. Note that the switch-cpu port does not show up anywhere on RouterOS except for the switch menu, none of the software-related configurations (e.g. interface-list) can be applied to this port. Packets that reach the CPU are automatically associated with the physical in-interface.

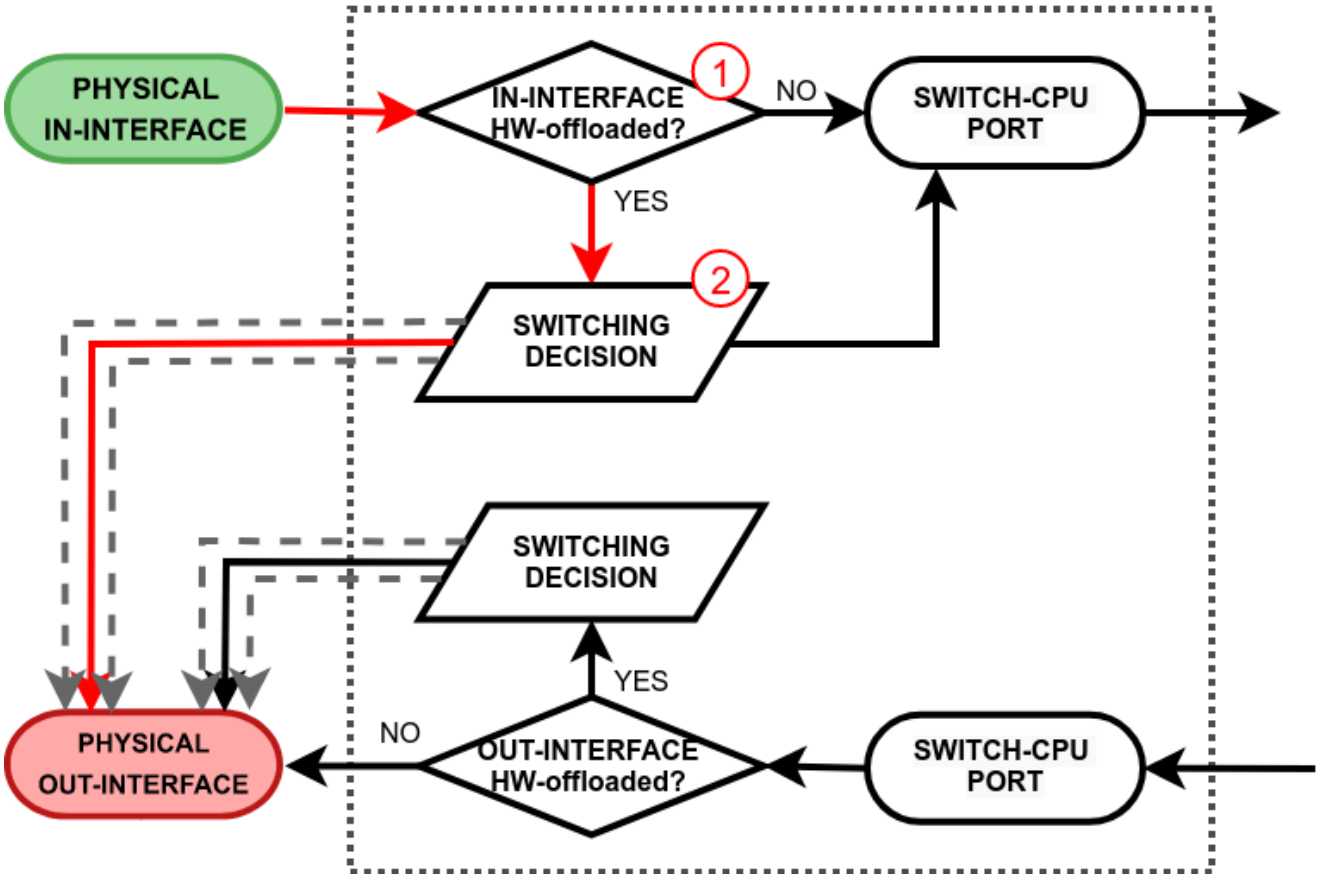
The hardware offloading, however, does not restrict a device to only hardware limited features, rather it is possible to take advantage of the hardware and software processing at the same time. This does require a profound understanding of how packets travel through the switch chip and when exactly they are passed to the main CPU.

> Switch features found in the "/interface/ethernet/switch" menu and its sub-menus, like ACL rules, mirroring, ingress/egress rate limiters, QoS, and L3HW (except inter-VLAN routing) may not rely on bridge hardware offloading. Therefore, they can **potentially** be applied to interfaces not configured within a hardware-offloaded bridge.

Switch Forward

We will further discuss a packet flow when bridge hardware offloading is enabled and a packet is forwarded between two switched ports on a single switch chip. This is the most common and also the simplest example:

1. The switch checks whether the in-interface is a hardware offloaded interface;
2. Run a packet through the switch host table to make a forwarding decision. If the switch finds a match for the destination MAC address, the packet is sent out through the physical interface. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic) gets multiplied and sent out to every hardware offloaded switch port.



Switch to CPU Input

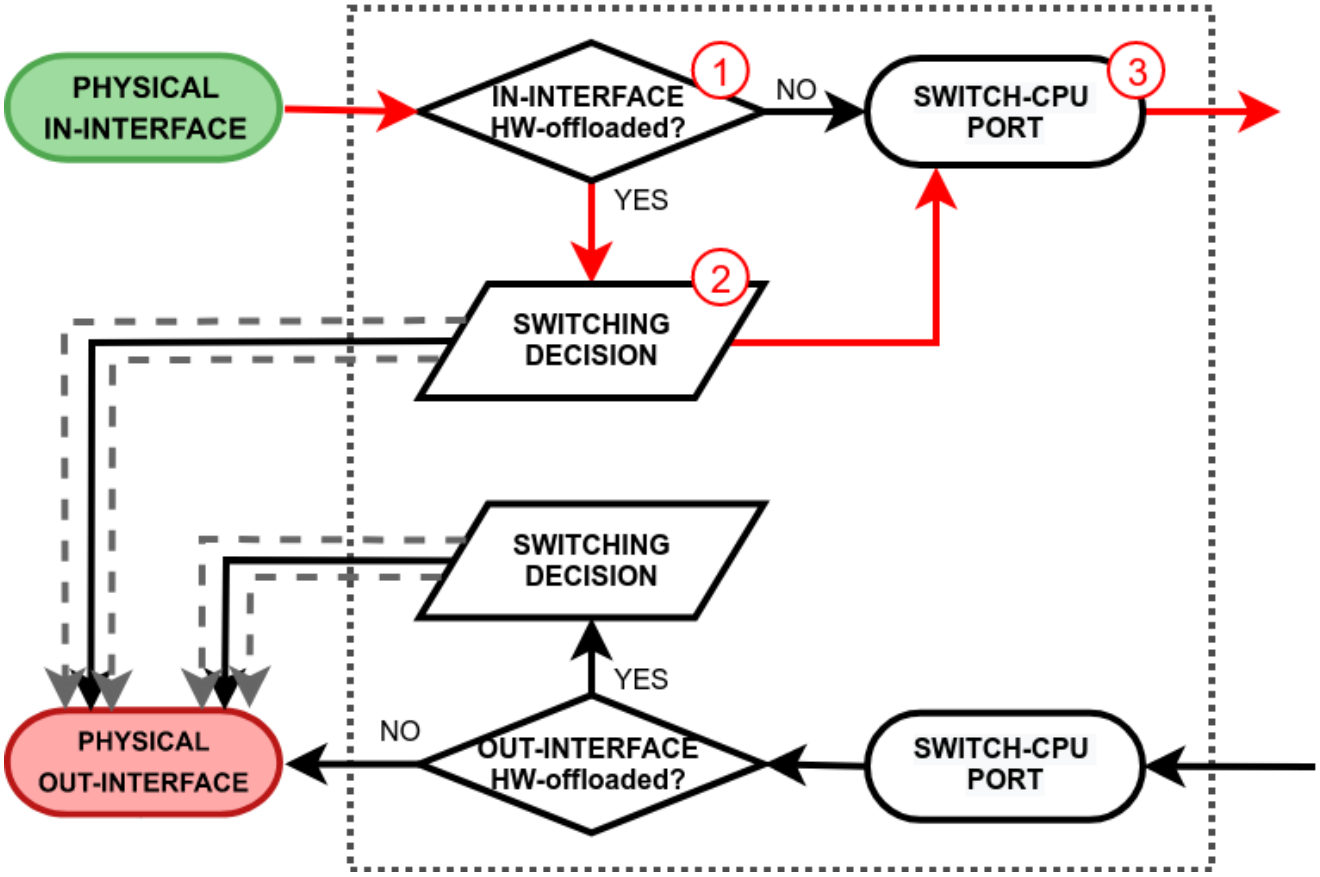
This process takes place when a packet is received on a physical interface and it is destined to switch-cpu port for further software processing. There are two paths to the switch-cpu. One where hardware offloading and switching is not even used (e.g. a standalone interface for routing or a bridged interface but with deliberately disabled HW offloading), so the packet is simply passed further for software processing. Another path is taken when hardware offloading is active on the in-interface. This will cause the packet to pass through the switching decision and there are various reasons why the switch might forward the packet to the switch-cpu port:

- a packet's destination MAC address match with a local MAC address, e.g. when a packet is destined to a local bridge interface;
- a packet might get flooded to all switch ports including the switch-cpu port, e.g. when broadcast, multicast, or unknown unicast traffic is received;

- a switch might have learned that some hosts can only be reached through the CPU (switch-cpu port learning is discussed in the next section), e. g. when a bridge contains HW and non-HW offloaded interfaces, such as wireless, EoIP, and even Ethernet interfaces;
- a packet is intentionally copied to the switch-cpu, e.g. for a packet inspection;
- a packet is triggered by the switch configuration and should be processed in software, e.g. a DHCP or IGMP snooping.

See the packet walkthrough when an in-interface is hardware offloaded:

1. The switch checks whether the in-interface is a hardware offloaded interface;
2. Run a packet through the switch host table to make a forwarding decision. In case any of the above-mentioned points are true, the packet gets forwarded to the switch-cpu port.
3. The packet exits through the switch-cpu port and it will be further processed by the RouterOS packet flow.



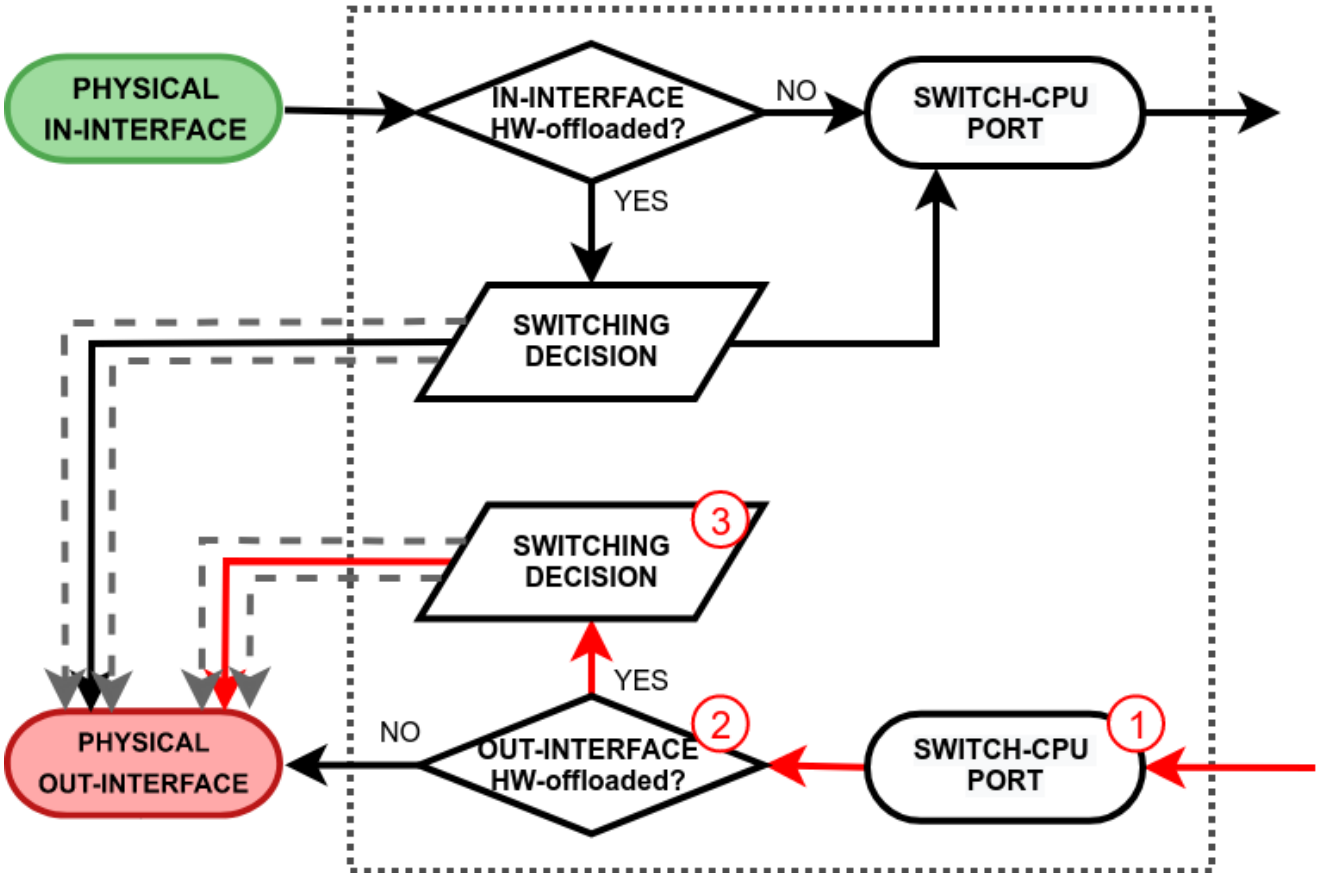
⚠ Any received packet that was flooded by the switch chip will not get flooded again by the software bridge to the same HW offloaded switch group. This prevents the formation of duplicate packets.

CPU Output to Switch

This process takes place when a packet exits the RouterOS software processing and is received on the switch-cpu port. Again, there are two paths the packet can take. One where hardware offloading and switching are not even used (e.g. a standalone interface for routing or a bridged interface but with deliberately disabled HW offloading), so the packet is simply sent out through the physical out-interface. Another path is taken when hardware offloading is active on the out-interface. This will cause the packet to pass through the switching decision. Just like any other switch port, the switch will learn the source MAC addresses from packets that are received on the switch-cpu port. This does come in handy when a bridge contains HW and non-HW offloaded interfaces, so the switch can learn which frames should be forwarded to the CPU. See the packet walkthrough when an out-interface is hardware offloaded:

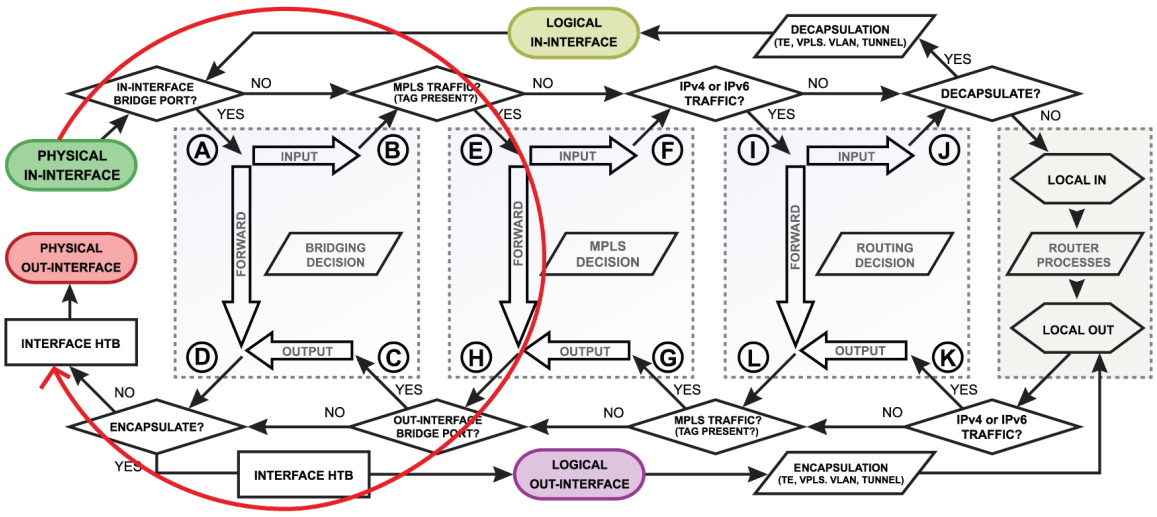
1. A packet that exits the RouterOS software processing is received on the switch-cpu port;
2. The switch checks whether the out-interface is a hardware offloaded interface;
3. Run a packet through the switch host table to make a forwarding decision. If the switch finds a match for the destination MAC

address, the packet is sent out through the physical interface. A packet that ends up being flooded (e.g. broadcast, multicast, unknown unicast traffic) gets multiplied and sent out to every hardware offloaded switch port.

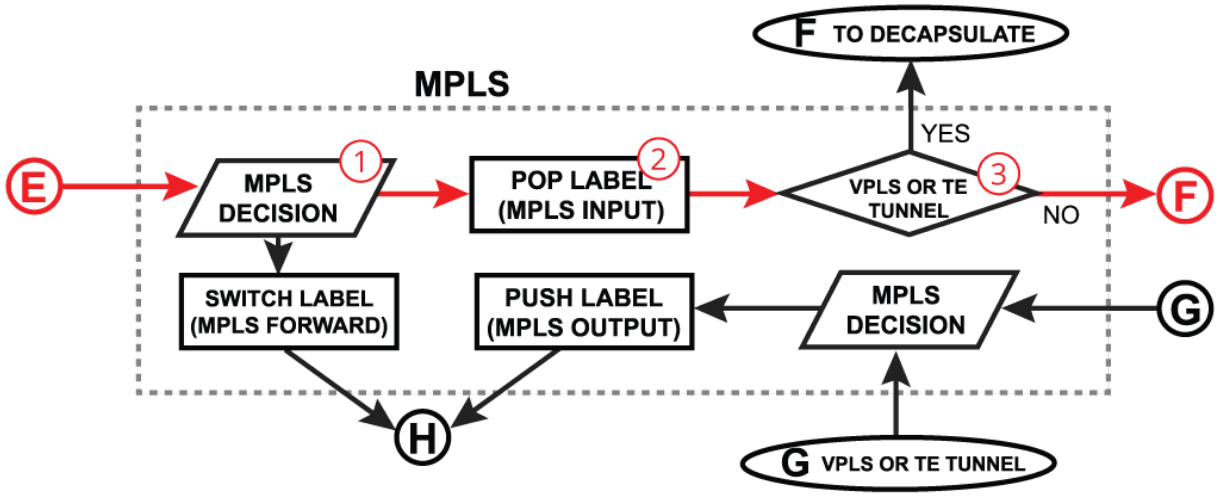


A software bridge that sends a flooded packet through HW offloaded interfaces, will only send a single packet copy per HW offloaded switch group rather than per HW offloaded interface. The actual flooding will be done by the switch chip, this prevents the formation of duplicate packets.

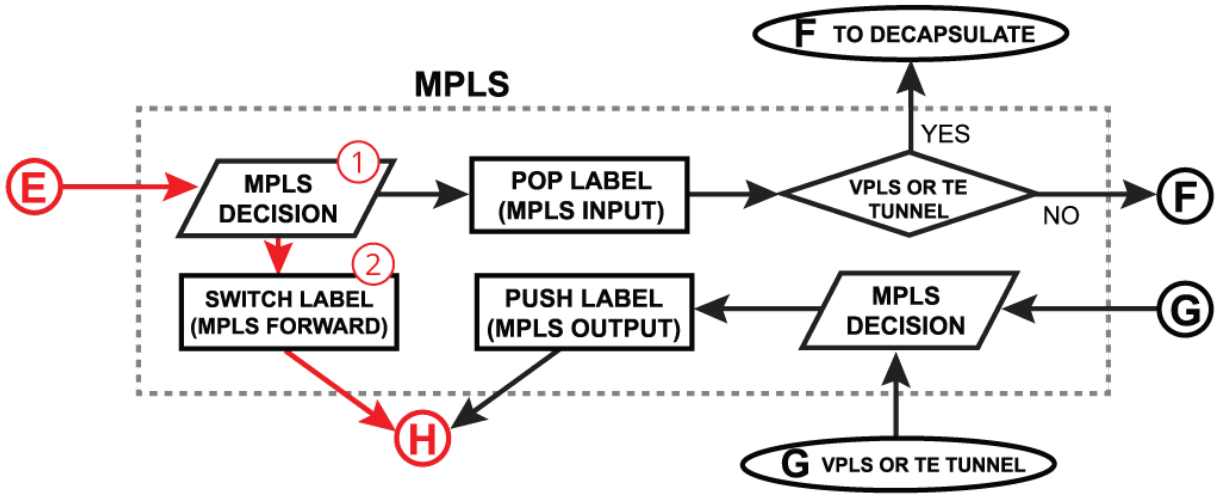
Flow of MPLS Packet



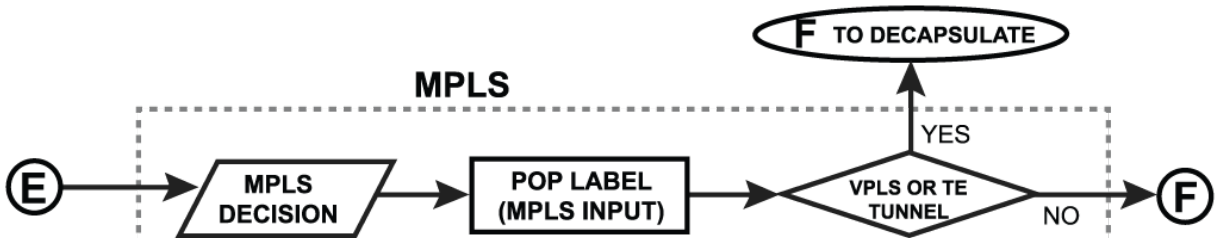
Pop Label

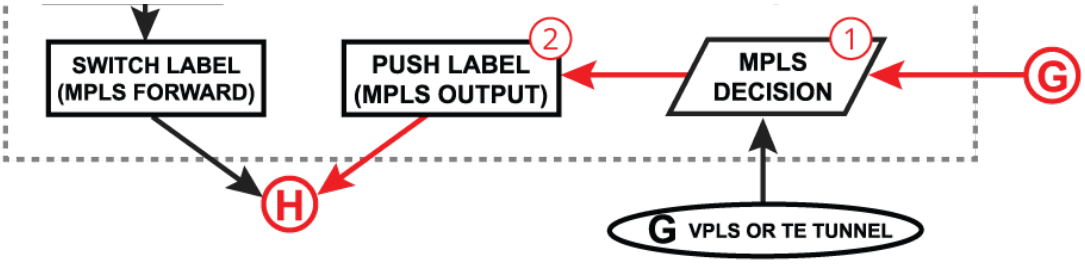


Switch Label



Push Label





MPLS IP VPN

In VPNv4 setups packet arriving at PE router that needs to be forwarded to the CE router is not a typical "forward".

If incoming label and destination is bound to the VRF Then after MPLS label is popped and:

- destination address is local to the router, then packet is moved to LOCAL_IN
- destination address is in the CE network, then packet is moved to LOCAL_OUT

> Forwarded packets from MPLS cloud to the CE network will not show up in the forward.

For example traffic from src:111.15.0.1 to dst:111.13.0.1

```
[admin@CCR2004_2XS] /mpls/forwarding-table> print
Flags: L - LDP, P - VPN
Columns: LABEL, VRF, PREFIX, NEXTHOPS
#   LABEL  VRF      PREFIX
NEXTHOPS

0 P    17  myVrf    111.13.0.0/24
4 L    20  main     203.0.113.2    { label=impl-null; nh=111.11.0.1; interface=sfp-sfpplus1 }
[admin@CCR2004_2XS] /mpls/forwarding-table>
...

[admin@CCR2004_2XS] /ip/route> print detail
Flags: D - dynamic; X - disabled, I - inactive, A - active; c - connect, s - static, r - rip, b - bgp, o -
ospf, i - is-is, d - dhcp, v - vpn, m - modem, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp

   DAc   dst-address=111.11.0.0/24 routing-table=main gateway=sfp-sfpplus1 immediate-gw=sfp-sfpplus1 distance=0
scope=10 suppress-hw-offload=no
         local-address=111.11.0.2%sfp-sfpplus1
   DAc   dst-address=203.0.113.1/32 routing-table=main gateway=lo immediate-gw=lo distance=0 scope=10 suppress-
hw-offload=no local-address=203.0.113.1%lo
   DAo   dst-address=203.0.113.2/32 routing-table=main gateway=111.11.0.1%sfp-sfpplus1 immediate-gw=111.11.0.1%
sfp-sfpplus1 distance=110 scope=20 target-scope=10
         suppress-hw-offload=no
   DAc   dst-address=111.13.0.0/24 routing-table=myVrf gateway=sfp-sfpplus2@myVrf immediate-gw=sfp-sfpplus2
distance=0 scope=10 suppress-hw-offload=no
         local-address=111.13.0.2%sfp-sfpplus2@myVrf
   DAy   dst-address=111.15.0.0/24 routing-table=myVrf gateway=203.0.113.2 immediate-gw=111.11.0.1%sfp-sfpplus1
distance=200 scope=40 target-scope=30 suppress-hw-offload=no [admin@CCR2004_2XS] /ip/route>
```

Packet will be seen in the output and postrouting chains, because now it is locally originated packet with source MAC address equal to vrfInterface:

```
08:10:55 firewall,info output: in:(unknown 0) out:sfp-sfpplus2, connection-state:established src-mac f2:b5:e9:
17:18:3b, proto ICMP (type 8, code 0), 111.15.0.1->111.13.0.1, len 56
08:10:55 firewall,info postrouting: in:myVrf out:sfp-sfpplus2, connection-state:established src-mac f2:b5:e9:17:
18:3b, proto ICMP (type 8, code 0), 111.15.0.1->111.13.0.1, len 56
```

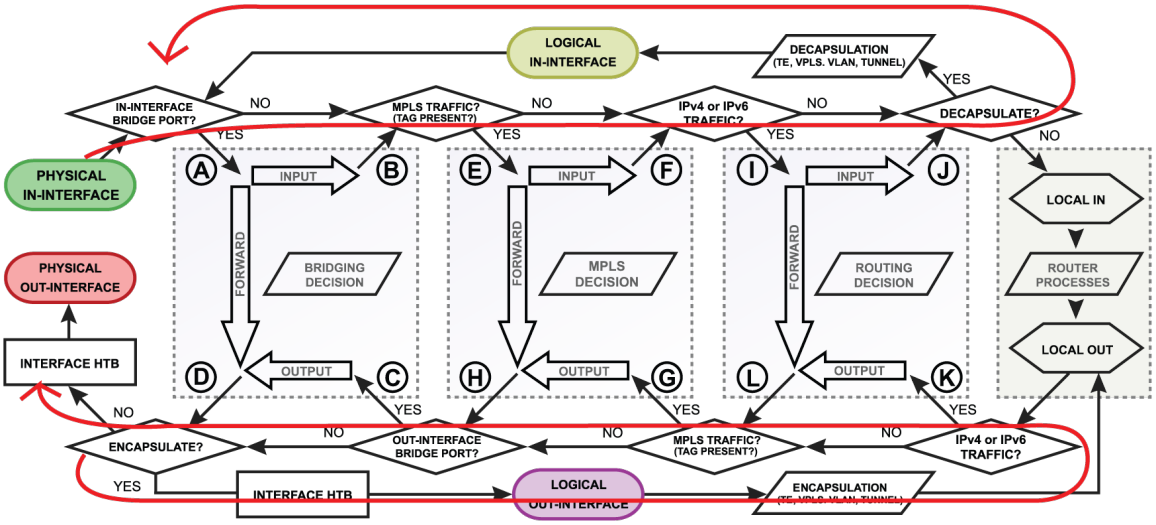
On the other hand, for packets routed in the direction CE→PE will be seen in the "forward" as any other routed IP traffic before being sent to the MPLS:

```
 08:10:55 firewall,info prerouting: in:sfp-sfpplus2 out:(unknown 0), connection-state:established src-mac dc:2c:
6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
 08:10:55 firewall,info forward: in:sfp-sfpplus2 out:sfp-sfpplus1, connection-state:established src-mac dc:2c:
6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
 08:10:55 firewall,info postrouting: in:sfp-sfpplus2 out:sfp-sfpplus1, connection-state:established src-mac dc:
2c:6e:46:f8:93, proto ICMP (type 0, code 0), 111.13.0.1->111.15.0.1, len 56
```

But there can be an exception. If destination IP of reply packet is local to the router and connection tracking is performing NAT translation then connection tracking will "force" packet to move through the firewall prerouting/forward/postrouting chains.

Logical Interfaces

So far we looked at examples when in or out interfaces are actual physical interfaces (Ethernet, wireless), but how packets will flow if the router receives tunnel encapsulated packets?



Let's assume that there is an IPIP packet coming into the router. Since it is a regular ipv4 packet it will be processed through all routing-related facilities (until "J" in the diagram). Then the router will look if the packet needs to be decapsulated., in this case, it is an IPIP packet so "yes" send the packet to decapsulation. After that packet will go another loop through all the facilities but this time as a decapsulated IPv4 packet.

It is very important because the packet actually travels through the firewall twice, so if there is a strict firewall, then there should be "accept" rules for IPIP encapsulated packets as well as decapsulated IP packets.

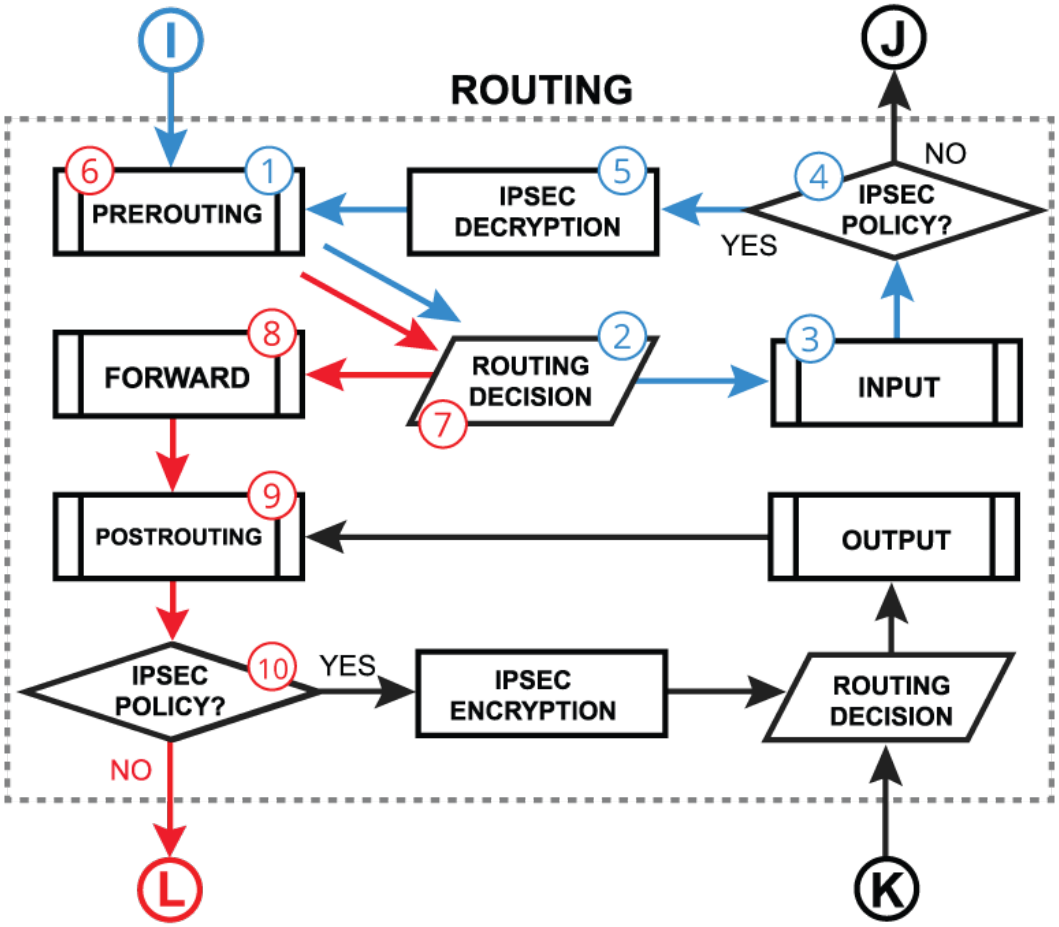
> Packet encapsulation and decapsulation using a bridge with enabled `vlan-filtering` do not relate to logical interfaces. See more details in the bridging section.

IPSec Policies

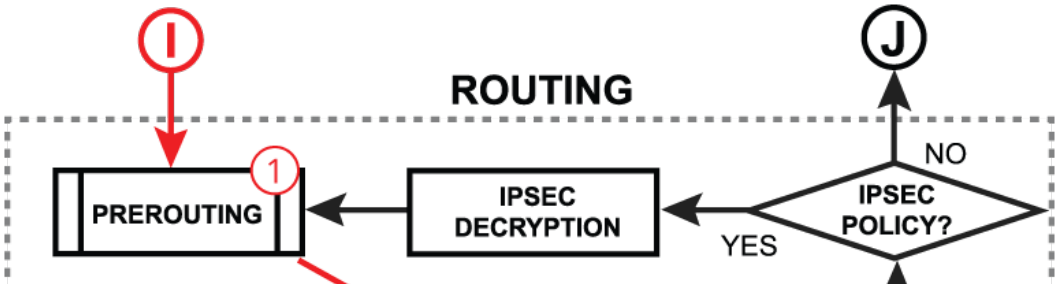
Let's take a look at another tunnel type - IPSec. This type of VPN does not have logical interfaces but is processed in a similar manner.

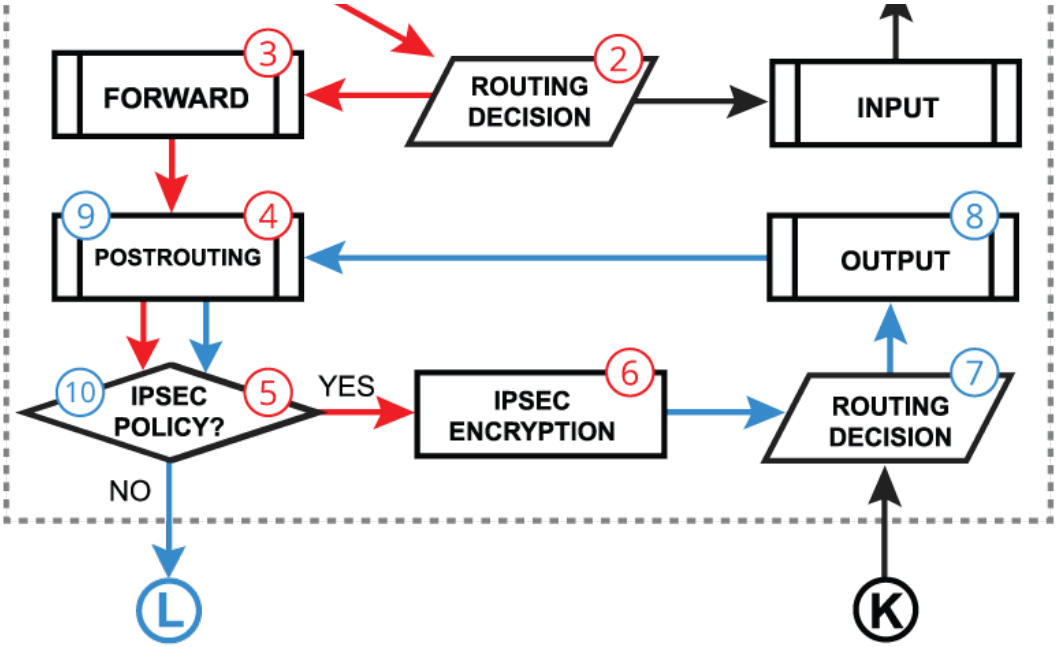
Instead of logical interfaces packets are processed through IPSec policies. After routing decision (2) and input firewall processing (3), the router tries to match the source and

destination to the IPsec policy. When policy matches the packet it is sent to decryption (5). After the decryption packet enters PREROUTING processing again (6) and starts another processing loop, but now with the decapsulated packet.



The same process is with encapsulation but in reverse order. The first IP packet gets processed through facilities, then matched against IPsec policies (5), encapsulated (6), and then sent to processing on the second loop (7-10).





Fast Path

From what we learned so far, it is quite obvious that such packet processing takes a lot of CPU resources. To fast things up FastPath was introduced in the first RouterOS v6. What it does is it skips processing in the Linux kernel, basically trading some RouterOS functionality for performance. For FastPath to work, interface driver support and specific configuration conditions are required.

How Fast Path Works

FastPath is an interface driver extension, that allows a driver to talk directly to specific RouterOS facilities and skip all others.

The packet can be forwarded by a fast path handler only if at least the source interface supports a fast path. For complete fast-forwarding, destination interface support is also required.

Currently, RouterOS has the following FastPath handlers:

- IPv4
- IPv4 FastTrack
- Traffic Generator
- MPLS
- Bridge

IPv4 FastPath handler is used if the following conditions are met:

- firewall rules are not configured;
- simple queue or queue trees with *parent=global* are not configured;
- no mesh, metarouter interface configuration;
- sniffer or torch is not running;
- connection tracking is not active;
- IP accounting is disabled;
- VRFs are not configured (`/ip route vrf` is empty);

- A hotspot is not used (`/ip hotspot` has no interfaces);
- IPSec policies are not configured;
- `/tool mac-scan` is not actively used;
- `/tool ip-scan` is not actively used.

⚠ Packets will travel the FastPath way if FastTrack is used no matter if the above conditions are met.

Traffic Generator and MPLS automatically use FastPath if the interface supports this feature. Currently, MPLS fast-path applies only to MPLS switched traffic (frames that enter router as MPLS and must leave router as MPLS) - MPLS ingress and egress (including VPLS tunnel endpoints that do VPLS encap/decap) will operate as before.

A Bridge handler is used if the following conditions are met:

- there are no bridge Calea, filter, NAT rules;
- *use-ip-firewall* is disabled;
- no mesh, MetaRouter interface configuration;
- sniffer, torch, and traffic generator are not running;
- bridge vlan-filtering is disabled (condition is removed since RouterOS 7.2 version);
- bridge dhcp-snooping is disabled.

⚠ FastPath on the vlan-filtering bridge does NOT support priority-tagged packets (packets with VLAN header but VLAN ID = 0). Those packets are redirected via a slow path.

Interfaces that support FastPath:

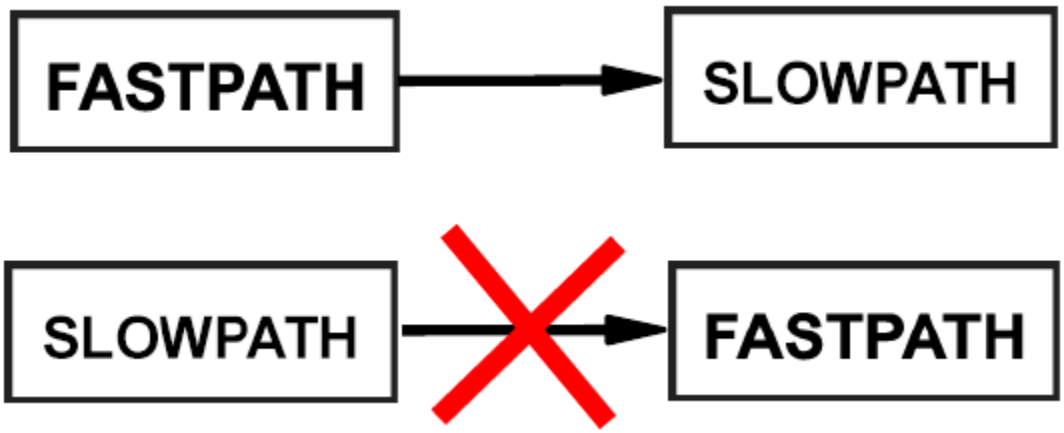
RouterBoard	Interfaces
RB6xx series	ether1,2
RB800	ether1,2
RB1100 series	ether1-11
All devices	Ethernet interfaces
	wireless interfaces
	bridge interfaces
	VLAN, VRRP interfaces
	bonding interfaces (RX only)
	PPPoE, L2TP interfaces
	EoIP, GRE, IPIP, VXLAN interfaces.

EoIP, Gre, IPIP, VXLAN and L2TP interfaces have per-interface setting *allow-fast-path*. Allowing a fast path on these interfaces has a side effect of bypassing firewall, connection tracking, simple queues, queue tree with parent=global, IP accounting, IPsec, hotspot universal client, vrf assignment for encapsulated packets that go through a fast-path. Also, packet fragments cannot be received in FastPath.

⚠ FastPath support can be verified by checking the fast-path property value in `/interface print detail`.

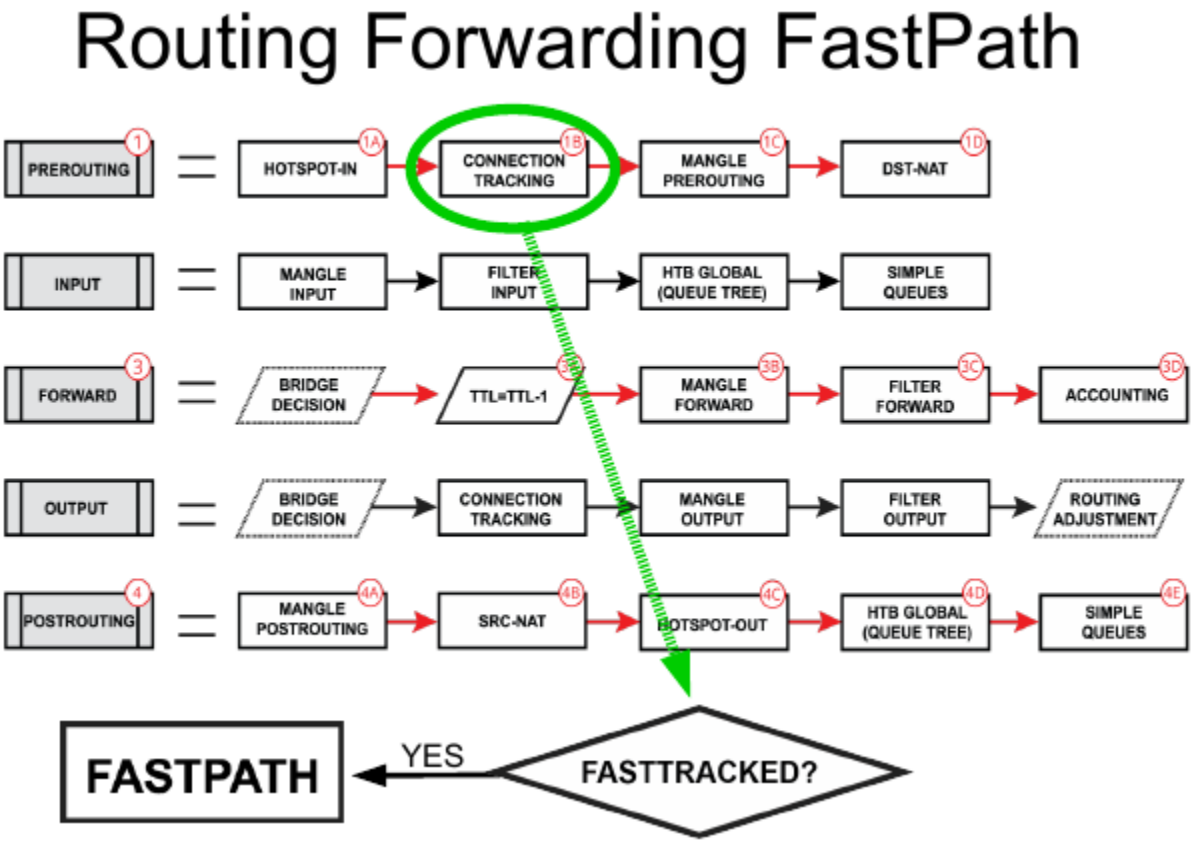
Only interface queue that guarantees FastPath is only-hardware-queue. If you need an interface queue other than hardware then the packet will not go fully FastPath, but there is not a big impact on performance, as "interface queue" is the last step in the packet flow.

The packet may go Half-FastPath by switching from FastPath to SlowPath, but not the other way around. So, for example, if the receiving interface has FastPath support, but the out interface does not, then the router will process the packet by FastPath handlers as far as it can and then proceed with SlowPath. If the receiving interface does not support FastPath but the out interface does, the packet will be processed by SlowPath all the way through the router.



FastTrack

Fasttrack can be decoded as Fast Path + Connection Tracking. It allows marking connections as "fast-tracked", marking packets that belong to fast-tracked connection will be sent fast-path way. The connection table entry for such a connection now will have a fast-tracked flag.



> ⚠ FastTrack packets bypass firewall, connection tracking, simple queues, queue tree with parent=global, ip traffic-flow(restriction removed in 6.33), IP accounting, IPSec, hotspot universal client, VRF assignment, so it is up to the administrator to make sure FastTrack does not interfere with other configuration!

To mark a connection as fast-tracked new action was implemented "*fasttrack-connection*" for firewall filter and mangle. Currently, only IPv4 TCP and UDP connections can be fast-tracked and to maintain connection tracking entries some random packets will still be sent to a slow path. This must be taken into consideration when designing firewalls with enabled "fasttrack".

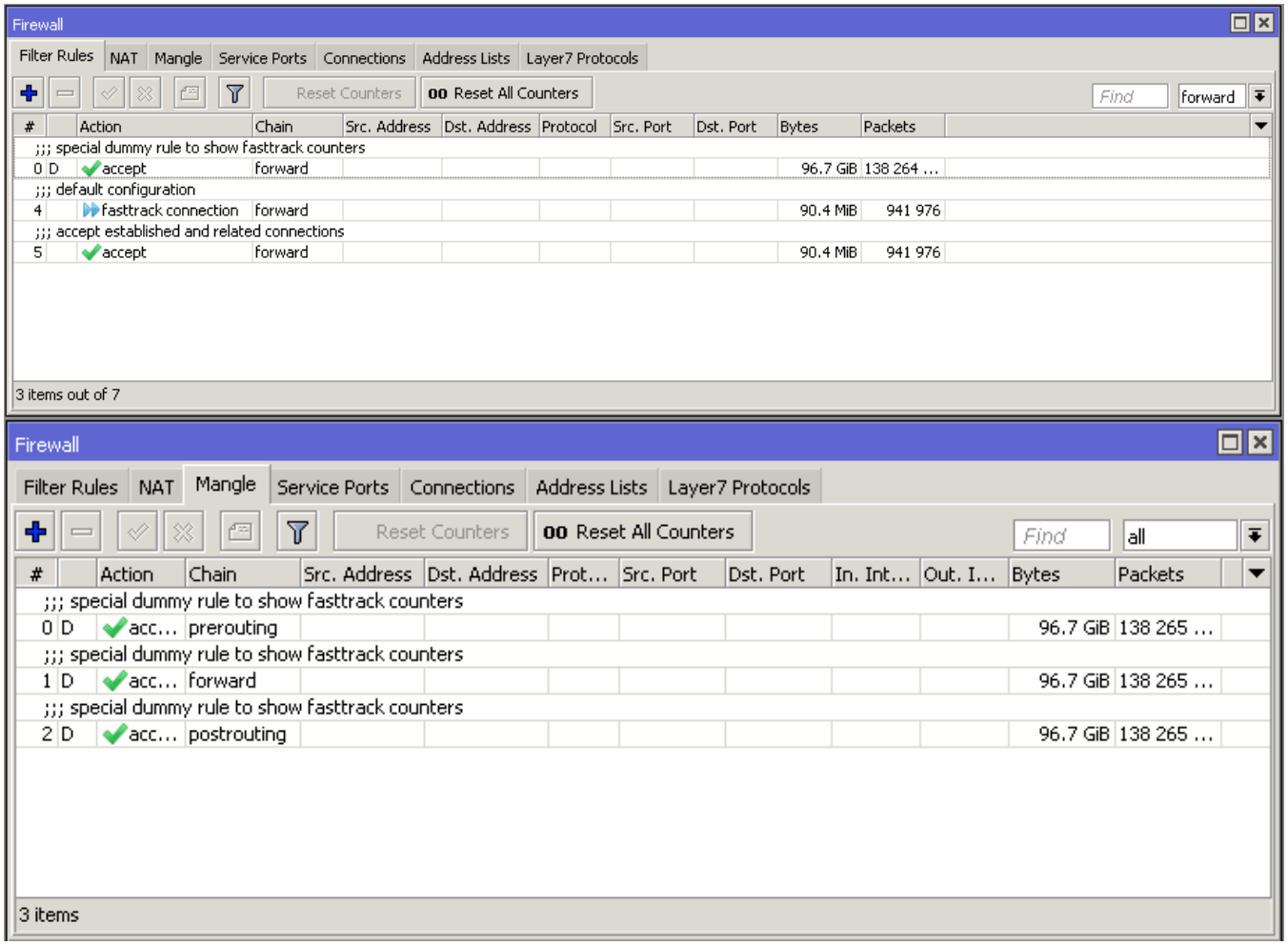
FastTrack handler also supports source and destination NAT, so special exceptions for NATed connections are not required.

Traffic that belongs to a fast-tracked connection travels in FastPath, which means that it will not be visible by other router L3 facilities (firewall, queues, IPsec, IP accounting, VRF assignment, etc). Fasttrack lookups route before routing marks have been set, so it works only with the main routing table.

The easiest way to start using this feature on home routers is to enable "fasttrack" for all *established, related* connections:

```
/ip firewall filter
add chain=forward action=fasttrack-connection connection-state=established,related \
  comment="fasttrack established/related"
add chain=forward action=accept connection-state=established,related \
  comment="accept established/related"
```

Notice that the first rule marks established/related connections as fast-tracked, the second rule is still required to accept packets belonging to those connections. The reason for this is that, as was mentioned earlier, some random packets from fast-tracked connections are still sent the slow pathway and only UDP and TCP are fast-tracked, but we still want to accept packets for other protocols.



After adding the "FastTrack" rule special dummy rule appeared at the top of the list. This is not an actual rule, it is for visual information showing that some of the traffic is traveling FastPath and will not reach other firewall rules.

FastTrack can process packets only in the main routing table so it is the system administrator duty to not FastTrack connections that are going through non-main routing table (thus connections that are processed with mangle action=mark-routing rules). Otherwise packets might be misrouted though the main routing table.

These rules appear as soon as there is at least one fast-tracked connection tracking entry and will disappear after the last fast-tracked connection times out in the connection table.

⚠ The connection is FastTracked until a connection is closed, timed out or the router is rebooted.

Configuration example: excluding specific host, from being Fast-Tracked

```
/ip firewall filter
add action=accept chain=forward connection-state=established,related src-address=192.168.88.111
add action=accept chain=forward connection-state=established,related dst-address=192.168.88.111
add action=fasttrack-connection chain=forward connection-state=established,related hw-offload=no
add action=accept chain=forward connection-state=established,related
```

In this example, we exclude host 192.168.88.111, from being Fast-tracked, by first accepting it with the firewall rule, both for source and destination. The idea is - not allowing the traffic to reach the FastTrack action.

Note: the "exclusion" rules, must be placed before fasttrack filters, order is important.

Requirements

IPv4 FastTrack is active if the following conditions are met:

- no mesh, metarouter interface configuration;
- sniffer, torch, and traffic generator are not running;
- "/tool mac-scan" is not actively used;
- "/tool ip-scan" is not actively used;
- FastPath and Route cache are enabled under IP/Settings (route cache condition does not apply to RouterOS v7 or newer);
- bridge FastPath is enabled if a connection is going over the bridge interface;

Packet flow for the visually impaired

The following document in DOCX format describes the diagram in a way optimized for visually impaired people. The descriptions are by Apex CoVantage care of Benetech. They are not being updated.

- Packet flow, optimized document.

Securing your router

Overview

The following steps are a recommendation on how to additionally protect your device with already configured strong firewall rules.

RouterOS version

Start by upgrading your RouterOS version. Some older releases have had certain weaknesses or vulnerabilities, that have been fixed. Keep your device up to date, to be sure it is secure. Click "check for updates" in Winbox or Webfig, to upgrade. We suggest you follow announcements on our security announcement blog to be informed about any new security issues.

Access to a router

Change username

Change default username *admin* to a different name. A custom name helps to protect access to your router if anybody got direct access to your router:

```
/user add name=myname password=mypassword group=full
/user disable admin
```

Change password

MikroTik routers require password configuration, we suggest using a password generator tool to create secure and non-repeating passwords. With secure password we mean:

- Minimum 12 characters;
- Include numbers, Symbols, Capital and lower case letters;
- Is not a Dictionary Word or Combination of Dictionary Words;

```
/user set myname password="!={Ba3N!"40TX+GvKBz?jTLIUcx/,"
```

Limit the MAC-access

RouterOS has built-in options for easy management access to network devices. The particular services should be shut down on production networks: **MAC-Telnet, MAC-Winbox,** and **MAC-Ping:**

```
/tool mac-server set allowed-interface-list=none
/tool mac-server mac-winbox set allowed-interface-list=none
/tool mac-server ping set enabled=no
```

Neighbor Discovery

MikroTik Neighbor discovery protocol is used to show and recognize other MikroTik routers in the network, disable neighbor discovery on all interfaces:

```
/ip neighbor discovery-settings set discover-interface-list=none
```

Bandwidth server

A bandwidth server is used to test throughput between two MikroTik routers. Disable it in the production environment:

```
/tool bandwidth-server set enabled=no
```

DNS cache

A router might have DNS cache enabled, which decreases resolving time for DNS requests from clients to remote servers. In case DNS cache is not required on your router or another router is used for such purposes, disable it:

```
/ip dns set allow-remote-requests=no
```

Other client services

RouterOS might have other services enabled (they are disabled by default RouterOS configuration). MikroTik caching proxy, socks, UPnP, and cloud services:

```
/ip proxy set enabled=no
/ip socks set enabled=no
/ip upnp set enabled=no
/ip cloud set ddns-enabled=no update-time=no
```

More Secure SSH access

It is possible to enable more strict SSH settings (add aes-128-ctr and disallow hmac sha1 and groups with sha1) with this command:

```
/ip ssh set strong-crypto=yes
```

Router interface

Ethernet/SFP interfaces

It is good practice to disable all unused interfaces on your router, in order to decrease unauthorized access to your router:

```
/interface print
/interface set X disabled=yes
```

Where **X** numbers of unused interfaces.

LCD

Some RouterBOARDs have an LCD module for informational purposes, set a pin:

```
/lcd/pin/set pin-number=3659 hide-pin-number=yes
```

or disable it:

```
/lcd/set enabled=no
```

Building Your First Firewall

Overview

We strongly suggest keeping the default firewall on. Here are a few adjustments to make it more secure. Make sure you configure additional changes when you completely understand the benefit of these particular firewall rules.

To see the default firewall rules through the CLI you can type:

```
/system default-configuration print
```

Ipv4 firewall

Protect the router itself

- work with *new* connections to decrease load on a router;
- create *address-list* for IP addresses, that are allowed to access your router;
- enable ICMP access (optionally);
- drop everything else, *log=yes* might be added to log packets that hit the specific rule;

```
/ip firewall filter
add action=accept chain=input comment="default configuration" connection-state=established,related
add action=accept chain=input src-address-list=allowed_to_router
add action=accept chain=input protocol=icmp
add action=drop chain=input
/ip firewall address-list
add address=192.168.88.2-192.168.88.254 list=allowed_to_router
```

Protect the LAN devices

We will create *address-list* with name "not_in_internet" which we will use for the firewall filter rules:

```
/ip firewall address-list
add address=0.0.0.0/8 comment=RFC6890 list=not_in_internet
add address=172.16.0.0/12 comment=RFC6890 list=not_in_internet
add address=192.168.0.0/16 comment=RFC6890 list=not_in_internet
add address=10.0.0.0/8 comment=RFC6890 list=not_in_internet
add address=169.254.0.0/16 comment=RFC6890 list=not_in_internet
add address=127.0.0.0/8 comment=RFC6890 list=not_in_internet
add address=224.0.0.0/4 comment=Multicast list=not_in_internet
add address=198.18.0.0/15 comment=RFC6890 list=not_in_internet
add address=192.0.0.0/24 comment=RFC6890 list=not_in_internet
add address=192.0.2.0/24 comment=RFC6890 list=not_in_internet
add address=198.51.100.0/24 comment=RFC6890 list=not_in_internet
add address=203.0.113.0/24 comment=RFC6890 list=not_in_internet
add address=100.64.0.0/10 comment=RFC6890 list=not_in_internet
add address=240.0.0.0/4 comment=RFC6890 list=not_in_internet
add address=192.88.99.0/24 comment="6to4 relay Anycast [RFC 3068]" list=not_in_internet
```

Brief firewall filter rule explanation:

- packets with *connection-state=established,related* added to FastTrack for faster data throughput, firewall will work with new connections only;
- drop *invalid* connection and log them with prefix "invalid";
- drop attempts to reach not public addresses from your local network, apply *address-list=not_in_internet* before, "bridge" is local network interface, log=yes attempts with prefix "!public_from_LAN";
- drop incoming packets that are not NAT`ed, ether1 is public interface, log attempts with "!NAT" prefix;
- jump to ICMP chain to drop unwanted ICMP messages
- drop incoming packets from the Internet, which are not public IP addresses, ether1 is a public interface, log attempts with prefix "!public";
- drop packets from LAN that does not have LAN IP, 192.168.88.0/24 is local network used subnet;

```
/ip firewall filter
add action=fasttrack-connection chain=forward comment=FastTrack connection-state=established,related
add action=accept chain=forward comment="Established, Related" connection-state=established,related
add action=drop chain=forward comment="Drop invalid" connection-state=invalid log=yes log-prefix=invalid
add action=drop chain=forward comment="Drop tries to reach not public addresses from LAN" dst-address-
list=not_in_internet in-interface=bridge log=yes log-prefix=!public_from_LAN out-interface=!bridge
add action=drop chain=forward comment="Drop incoming packets that are not NAT`ted" connection-nat-state=!dstnat
connection-state=new in-interface=ether1 log=yes log-prefix=!NAT
add action=jump chain=forward protocol=icmp jump-target=icmp comment="jump to ICMP filters"
add action=drop chain=forward comment="Drop incoming from internet which is not public IP" in-interface=ether1
log=yes log-prefix=!public src-address-list=not_in_internet
add action=drop chain=forward comment="Drop packets from LAN that do not have LAN IP" in-interface=bridge
log=yes log-prefix=LAN_!LAN src-address=!192.168.88.0/24
```

Allow only needed icmp codes in "icmp" chain:

```
/ip firewall filter
  add chain=icmp protocol=icmp icmp-options=0:0 action=accept \
    comment="echo reply"
  add chain=icmp protocol=icmp icmp-options=3:0 action=accept \
    comment="net unreachable"
  add chain=icmp protocol=icmp icmp-options=3:1 action=accept \
    comment="host unreachable"
  add chain=icmp protocol=icmp icmp-options=3:4 action=accept \
    comment="host unreachable fragmentation required"
  add chain=icmp protocol=icmp icmp-options=8:0 action=accept \
    comment="allow echo request"
  add chain=icmp protocol=icmp icmp-options=11:0 action=accept \
    comment="allow time exceed"
  add chain=icmp protocol=icmp icmp-options=12:0 action=accept \
    comment="allow parameter bad"
  add chain=icmp action=drop comment="deny all other types"
```

IPv6 firewall

Protect the router itself

Create an address-list from which you allow access to the device:

```
/ipv6 firewall address-list add address=fd12:672e:6f65:8899::/64 list=allowed
```

Brief IPv6 firewall filter rule explanation:

- work with *new* packets, accept *established/related* packets;
- drop *link-local* addresses from Internet(public) interface/interface-list;
- accept access to a router from *link-local* addresses, accept *multicast* addresses for management purposes, accept your source *address-list* for router access;
- drop anything else;

```
/ipv6 firewall filter
add action=accept chain=input comment="allow established and related" connection-state=established,related
add chain=input action=accept protocol=icmpv6 comment="accept ICMPv6"
add chain=input action=accept protocol=udp port=33434-33534 comment="defconf: accept UDP traceroute"
add chain=input action=accept protocol=udp dst-port=546 src-address=fe80::/10 comment="accept DHCPv6-Client
prefix delegation."
add action=drop chain=input in-interface=sit1 log=yes log-prefix=dropLL_from_public src-address=fe80::/10
add action=accept chain=input comment="allow allowed addresses" src-address-list=allowed
add action=drop chain=input
/ipv6 firewall address-list
add address=fe80::/16 list=allowed
add address=xxxx::/48 list=allowed
add address=ff02::/16 comment=multicast list=allowed
```

⚠ In certain setups where the DHCPv6 relay is used, the src address of the packets may not be from the link-local range. In that case, the src-address parameter of rule #4 must be removed or adjusted to accept the relay address.

Protect the LAN devices

Enabled IPv6 puts your clients available for public networks, set proper firewall to protect your customers.

- accept *established/related* and work with *new* packets;
- drop *invalid* packets and put prefix for rules;
- accept ICMP packets;
- accept *new* connection from your clients to the Internet;
- drop everything else.

```
/ipv6 firewall filter
add action=accept chain=forward comment=established,related connection-state=established,related
add action=drop chain=forward comment=invalid connection-state=invalid log=yes log-prefix=ipv6,invalid
add action=accept chain=forward comment=icmpv6 in-interface=!sit1 protocol=icmpv6
add action=accept chain=forward comment="local network" in-interface=!sit1 src-address-list=allowed
add action=drop chain=forward log-prefix=IPV6
```

Building Advanced Firewall

Overview

From everything we have learned so far, let's try to build an advanced firewall. In this firewall building example, we will try to use as many firewall features as we can to illustrate how they work and when they should be used the right way.

Most of the filtering will be done in the RAW firewall, a regular firewall will contain just a basic rule set to accept *established, related,* and *untracked* connections as well as dropping everything else not coming from LAN to fully protect the router.

Interface Lists

Two interface lists will be used **WAN** and **LAN** for easier future management purposes. Interfaces connected to the global internet should be added to the WAN list, in this case, it is *ether1*!

```
/interface list
  add comment=defconf name=WAN
  add comment=defconf name=LAN
/interface list member
  add comment=defconf interface=bridge list=LAN
  add comment=defconf interface=ether1 list=WAN
```

Protect the Device

The main goal here is to allow access to the router only from LAN and drop everything else.

Notice that ICMP is accepted here as well, it is used to accept ICMP packets that passed RAW rules.

```
/ip firewall filter
  add action=accept chain=input comment="defconf: accept ICMP after RAW" protocol=icmp
  add action=accept chain=input comment="defconf: accept established,related,untracked" connection-
state=established,related,untracked
  add action=drop chain=input comment="defconf: drop all not coming from LAN" in-interface-list=!LAN
```

IPv6 part is a bit more complicated, in addition, UDP traceroute, DHCPv6 client PD, and IPSec (IKE, AH, ESP) is accepted as per RFC recommendations.

```
/ipv6 firewall filter
add action=accept chain=input comment="defconf: accept ICMPv6 after RAW" protocol=icmpv6
add action=accept chain=input comment="defconf: accept established,related,untracked" connection-
state=established,related,untracked
add action=accept chain=input comment="defconf: accept UDP traceroute" dst-port=33434-33534 protocol=udp
add action=accept chain=input comment="defconf: accept DHCPv6-Client prefix delegation." dst-port=546
protocol=udp src-address=fe80::/10
add action=accept chain=input comment="defconf: accept IKE" dst-port=500,4500 protocol=udp
add action=accept chain=input comment="defconf: accept IPSec AH" protocol=ipsec-ah
add action=accept chain=input comment="defconf: accept IPSec ESP" protocol=ipsec-esp
add action=drop chain=input comment="defconf: drop all not coming from LAN" in-interface-list=!LAN
```

⚠ In certain setups where the DHCPv6 relay is used, the src address of the packets may not be from the link-local range. In that case, the src-address parameter of rule #4 must be removed or adjusted to accept the relay address.

Protect the Clients

Before the actual set of rules, let's create a necessary *address-list* that contains all IPv4/6 addresses that cannot be forwarded.

Notice that in this list multicast address range is added. It is there because in most cases multicast is not used. If you intend to use multicast forwarding, then this address list entry should be disabled.

```
/ip firewall address-list
  add address=0.0.0.0/8 comment="defconf: RFC6890" list=no_forward_ipv4
  add address=169.254.0.0/16 comment="defconf: RFC6890" list=no_forward_ipv4
  add address=224.0.0.0/4 comment="defconf: multicast" list=no_forward_ipv4
  add address=255.255.255.255/32 comment="defconf: RFC6890" list=no_forward_ipv4
```

In the same case for IPv6, if multicast forwarding is used then the multicast entry should be disabled from the *address-list.*

```
/ipv6 firewall address-list
  add address=fe80::/10  comment="defconf: RFC6890 Linked-Scoped Unicast" list=no_forward_ipv6
  add address=ff00::/8  comment="defconf: multicast" list=no_forward_ipv6
```

Forward chain will have a bit more rules than input:

- accept *established, related* and *untracked* connections;
- FastTrack *established* and *related* connections (currently only IPv4);
- drop *invalid* connections;
- drop bad forward IP`s, since we cannot reliably determine in RAW chains which packets are forwarded
- drop connections initiated from the internet (from the WAN side which is not destination NAT`ed);
- drop bogon IP`s that should not be forwarded.

We are dropping all non-dstnated IPv4 packets to protect direct attacks on the clients if the attacker knows the internal LAN network. Typically this rule would not be necessary since RAW filters will drop such packets, however, the rule is there for double security in case RAW rules are accidentally messed up.

```
/ip firewall filter
  add action=accept chain=forward comment="defconf: accept all that matches IPSec policy" ipsec-policy=in,ipsec
disabled=yes
  add action=fasttrack-connection chain=forward comment="defconf: fasttrack" connection-state=established,
related
  add action=accept chain=forward comment="defconf: accept established,related, untracked" connection-
state=established,related,untracked
  add action=drop chain=forward comment="defconf: drop invalid" connection-state=invalid
  add action=drop chain=forward comment="defconf:  drop all from WAN not DSTNATed" connection-nat-state=!dstnat
connection-state=new in-interface-list=WAN
  add action=drop chain=forward src-address-list=no_forward_ipv4 comment="defconf: drop bad forward IPs"
  add action=drop chain=forward dst-address-list=no_forward_ipv4 comment="defconf: drop bad forward IPs"
```

IPv6 *forward* chain is very similar, except that IPsec and HIP are accepted as per RFC recommendations and ICMPv6 with *hop-limit=1* is dropped.

```
/ipv6 firewall filter
add action=accept chain=forward comment="defconf: accept established,related,untracked" connection-
state=established,related,untracked
add action=drop chain=forward comment="defconf: drop invalid" connection-state=invalid
add action=drop chain=forward src-address-list=no_forward_ipv6 comment="defconf: drop bad forward IPs"
add action=drop chain=forward dst-address-list=no_forward_ipv6 comment="defconf: drop bad forward IPs"
add action=drop chain=forward comment="defconf: rfc4890 drop hop-limit=1" hop-limit=equal:1 protocol=icmpv6
add action=accept chain=forward comment="defconf: accept ICMPv6 after RAW" protocol=icmpv6
add action=accept chain=forward comment="defconf: accept HIP" protocol=139
add action=accept chain=forward comment="defconf: accept IKE" protocol=udp dst-port=500,4500
add action=accept chain=forward comment="defconf: accept AH" protocol=ipsec-ah
add action=accept chain=forward comment="defconf: accept ESP" protocol=ipsec-esp
add action=accept chain=forward comment="defconf: accept all that matches IPSec policy" ipsec-policy=in,ipsec
add action=drop chain=forward comment="defconf: drop everything else not coming from LAN" in-interface-list=!LAN
```

Notice the IPsec policy matcher rules. It is very important that IPsec encapsulated traffic bypass fast-track. That is why as an illustration we have added a disabled rule to accept traffic matching IPsec policies. Whenever IPsec tunnels are used on the router this rule should be enabled. For IPv6 it is much more simple since it does not have fast-track support.

Another approach to solving the IPsec problem is to add RAW rules, we will talk about this method later in the RAW section

Masquerade Local Network

For local devices behind the router to be able to access the internet, local networks must be masqueraded. In most cases, it is advised to use src-nat instead of masquerade, however in this case when the WAN address is dynamic it is the only option.

```
/ip firewall nat
  add action=accept chain=srcnat comment="defconf: accept all that matches IPSec policy" ipsec-policy=out,ipsec
disabled=yes
  add action=masquerade chain=srcnat comment="defconf: masquerade" out-interface-list=WAN
```

Notice the disabled policy matcher rule, the same as in firewall filters IPSec traffic must be excluded from being NATed (except specific scenarios where IPsec policy is configured to match NAT`ed address). So whenever IPsec tunnels are used on the router this rule must be enabled.

RAW Filtering

IPv4 Address Lists

Before setting RAW rules, let's create some address lists necessary for our filtering policy. RFC 6890 will be used as a reference.

First, *address-list* contains all IPv4 addresses that cannot be used as src/dst/forwarded, etc. (will be dropped immediately if such address is seen)

```
/ip firewall address-list
  add address=127.0.0.0/8 comment="defconf: RFC6890" list=bad_ipv4
  add address=192.0.0.0/24 comment="defconf: RFC6890" list=bad_ipv4
  add address=192.0.2.0/24 comment="defconf: RFC6890 documentation" list=bad_ipv4
  add address=198.51.100.0/24 comment="defconf: RFC6890 documentation" list=bad_ipv4
  add address=203.0.113.0/24 comment="defconf: RFC6890 documentation" list=bad_ipv4
  add address=240.0.0.0/4 comment="defconf: RFC6890 reserved" list=bad_ipv4
```

Another address list contains all IPv4 addresses that cannot be routed globally.

```
/ip firewall address-list
  add address=0.0.0.0/8 comment="defconf: RFC6890" list=not_global_ipv4
  add address=10.0.0.0/8 comment="defconf: RFC6890" list=not_global_ipv4
  add address=100.64.0.0/10 comment="defconf: RFC6890" list=not_global_ipv4
  add address=169.254.0.0/16 comment="defconf: RFC6890" list=not_global_ipv4
  add address=172.16.0.0/12 comment="defconf: RFC6890" list=not_global_ipv4
  add address=192.0.0.0/29 comment="defconf: RFC6890" list=not_global_ipv4
  add address=192.168.0.0/16 comment="defconf: RFC6890" list=not_global_ipv4
  add address=198.18.0.0/15 comment="defconf: RFC6890 benchmark" list=not_global_ipv4
  add address=255.255.255.255/32 comment="defconf: RFC6890" list=not_global_ipv4
```

And last two address lists for addresses that cannot be as destination or source address.

```
/ip firewall address-list
  add address=224.0.0.0/4 comment="defconf: multicast" list=bad_src_ipv4
  add address=255.255.255.255/32 comment="defconf: RFC6890" list=bad_src_ipv4
add address=0.0.0.0/8 comment="defconf: RFC6890" list=bad_dst_ipv4
  add address=224.0.0.0/4 comment="defconf: RFC6890" list=bad_dst_ipv4
```

IPv4 RAW Rules

Raw IPv4 rules will perform the following actions:

- **add disabled "accept" rule** - can be used to quickly disable RAW filtering without disabling all RAW rules;
- **accept** DHCP discovery - most of the DHCP packets are not seen by an IP firewall, but some of them are, so make sure that they are accepted;
- **drop** packets that use bogon IP`s;
- **drop** from invalid SRC and DST IP`s;
- **drop** globally unroutable IP`s coming from WAN;
- **drop** packets with source-address not equal to 192.168.88.0/24 (default IP range) coming from LAN;
- **drop** packets coming from WAN to be forwarded to 192.168.88.0/24 network, this will protect from attacks if the attacker knows internal network;
- **drop** bad ICMP, UDP, and TCP;
- **accept** everything else coming from WAN and LAN;
- **drop** everything else, to make sure that any newly added interface (like PPPoE connection to service provider) is protected against accidental misconfiguration.

```
/ip firewall raw
add action=accept chain=prerouting comment="defconf: enable for transparent firewall" disabled=yes
add action=accept chain=prerouting comment="defconf: accept DHCP discover" dst-address=255.255.255.255 dst-
port=67 in-interface-list=LAN protocol=udp src-address=0.0.0.0 src-port=68
add action=drop chain=prerouting comment="defconf: drop bogon IP's" src-address-list=bad_ipv4
add action=drop chain=prerouting comment="defconf: drop bogon IP's" dst-address-list=bad_ipv4
add action=drop chain=prerouting comment="defconf: drop bogon IP's" src-address-list=bad_src_ipv4
add action=drop chain=prerouting comment="defconf: drop bogon IP's" dst-address-list=bad_dst_ipv4
add action=drop chain=prerouting comment="defconf: drop non global from WAN" src-address-list=not_global_ipv4
in-interface-list=WAN
add action=drop chain=prerouting comment="defconf: drop forward to local lan from WAN" in-interface-list=WAN
dst-address=192.168.88.0/24
add action=drop chain=prerouting comment="defconf: drop local if not from default IP range" in-interface-
list=LAN src-address=!192.168.88.0/24
add action=drop chain=prerouting comment="defconf: drop bad UDP" port=0 protocol=udp
add action=jump chain=prerouting comment="defconf: jump to ICMP chain" jump-target=icmp4 protocol=icmp
add action=jump chain=prerouting comment="defconf: jump to TCP chain" jump-target=bad_tcp protocol=tcp
add action=accept chain=prerouting comment="defconf: accept everything else from LAN" in-interface-list=LAN
add action=accept chain=prerouting comment="defconf: accept everything else from WAN" in-interface-list=WAN
add action=drop chain=prerouting comment="defconf: drop the rest"
```

Notice that we used some optional chains, the first **TCP** chain to drop **TCP** packets known to be *invalid*.

```
/ip firewall raw
add action=drop chain=bad_tcp comment="defconf: TCP flag filter" protocol=tcp tcp-flags=!fin,!syn,!rst,!ack
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=fin,syn
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=fin,rst
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=fin,!ack
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=fin,urg
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=syn,rst
add action=drop chain=bad_tcp comment=defconf protocol=tcp tcp-flags=rst,urg
add action=drop chain=bad_tcp comment="defconf: TCP port 0 drop" port=0 protocol=tcp
```

And another chain for **ICMP**. Note that if you want a very strict firewall then such strict **ICMP** filtering can be used, but in most cases, it is not necessary and simply adds more load on the router's CPU. ICMP rate limit in most cases is also unnecessary since the Linux kernel is already limiting ICMP packets to 100pps.

```
/ip firewall raw
add action=accept chain=icmp4 comment="defconf: echo reply" icmp-options=0:0 limit=5,10:packet protocol=icmp
add action=accept chain=icmp4 comment="defconf: net unreachable" icmp-options=3:0 protocol=icmp
add action=accept chain=icmp4 comment="defconf: host unreachable" icmp-options=3:1 protocol=icmp
add action=accept chain=icmp4 comment="defconf: protocol unreachable" icmp-options=3:2 protocol=icmp
add action=accept chain=icmp4 comment="defconf: port unreachable" icmp-options=3:3 protocol=icmp
add action=accept chain=icmp4 comment="defconf: fragmentation needed" icmp-options=3:4 protocol=icmp
add action=accept chain=icmp4 comment="defconf: echo" icmp-options=8:0 limit=5,10:packet protocol=icmp
add action=accept chain=icmp4 comment="defconf: time exceeded " icmp-options=11:0-255 protocol=icmp
add action=drop chain=icmp4 comment="defconf: drop other icmp" protocol=icmp
```

IPv6 Address Lists

List of IPv6 addresses that should be dropped instantly

```
/ipv6 firewall address-list
add address=::1/128 comment="defconf: RFC6890 lo" list=bad_ipv6
add address=::ffff:0:0/96 comment="defconf: RFC6890 IPv4 mapped" list=bad_ipv6
add address=2001::/23 comment="defconf: RFC6890" list=bad_ipv6
add address=2001:db8::/32 comment="defconf: RFC6890 documentation" list=bad_ipv6
add address=2001:10::/28 comment="defconf: RFC6890 orchid" list=bad_ipv6
add address=::/96 comment="defconf: ipv4 compat" list=bad_ipv6
```

List of IPv6 addresses that are not globally routable

```
/ipv6 firewall address-list
add address=100::/64 comment="defconf: RFC6890 Discard-only" list=not_global_ipv6
add address=2001::/32 comment="defconf: RFC6890 TEREDO" list=not_global_ipv6
add address=2001:2::/48 comment="defconf: RFC6890 Benchmark" list=not_global_ipv6
add address=fc00::/7 comment="defconf: RFC6890 Unique-Local" list=not_global_ipv6
```

List of addresses as an invalid destination address

```
/ipv6 firewall address-list add address=::/128 comment="defconf: unspecified" list=bad_dst_ipv6
```

List of addresses as an invalid source address

```
/ipv6 firewall address-list
  add address=::/128 comment="defconf: unspecified" list=bad_src_ipv6
  add address=ff00::/8  comment="defconf: multicast" list=bad_src_ipv6
```

IPv6 RAW Rules

Raw IPv6 rules will perform the following actions:

- **add disabled accept rule** - can be used to quickly disable RAW filtering without disabling all RAW rules;
- **drop** packets that use bogon IPs;
- **drop** from invalid SRC and DST IPs;
- **drop** globally unroutable IPs coming from WAN;
- **drop** bad ICMP;
- **accept** everything else coming from WAN and LAN;
- **drop** everything else, to make sure that any newly added interface (like PPPoE connection to service provider) is protected against accidental misconfiguration.

```
/ipv6 firewall raw
add action=accept chain=prerouting comment="defconf: enable for transparent firewall" disabled=yes
add action=accept chain=prerouting comment="defconf: RFC4291, section 2.7.1" src-address=::/128 dst-
address=ff02:0:0:0:0:1:ff00::/104 icmp-options=135 protocol=icmpv6
add action=drop chain=prerouting comment="defconf: drop bogon IP's" src-address-list=bad_ipv6
add action=drop chain=prerouting comment="defconf: drop bogon IP's" dst-address-list=bad_ipv6
add action=drop chain=prerouting comment="defconf: drop packets with bad SRC ipv6" src-address-list=bad_src_ipv6
add action=drop chain=prerouting comment="defconf: drop packets with bad dst ipv6" dst-address-list=bad_dst_ipv6
add action=drop chain=prerouting comment="defconf: drop non global from WAN" src-address-list=not_global_ipv6
in-interface-list=WAN
add action=jump chain=prerouting comment="defconf: jump to ICMPv6 chain" jump-target=icmp6 protocol=icmpv6
add action=accept chain=prerouting comment="defconf: accept local multicast scope" dst-address=ff02::/16
add action=drop chain=prerouting comment="defconf: drop other multicast destinations" dst-address=ff00::/8
add action=accept chain=prerouting comment="defconf: accept everything else from WAN" in-interface-list=WAN
add action=accept chain=prerouting comment="defconf: accept everything else from LAN" in-interface-list=LAN
add action=drop chain=prerouting comment="defconf: drop the rest"
```

Notice that the optional **ICMP** chain was used. If you want a very strict firewall then such strict **ICMP** filtering can be used, but in most cases, it is not necessary and simply adds more load on the router's CPU. ICMP rate limit in most cases is also unnecessary since the Linux kernel is already limiting ICMP packets to 100pps

```
/ipv6 firewall raw
# Be aware that different operating systems originate packets with different default TTL values
add action=drop chain=icmp6 comment="defconf: rfc4890 drop ll if hop-limit!=255" dst-address=fe80::/10 hop-
limit=not-equal:255 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: dst unreachable" icmp-options=1:0-255 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: packet too big" icmp-options=2:0-255 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: limit exceeded" icmp-options=3:0-1 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: bad header" icmp-options=4:0-2 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: Mobile home agent address discovery" icmp-options=144:0-255
protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: Mobile home agent address discovery" icmp-options=145:0-255
protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: Mobile prefix solic" icmp-options=146:0-255 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: Mobile prefix advert" icmp-options=147:0-255 protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: echo request limit 5,10" icmp-options=128:0-255 limit=5,10:
packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: echo reply limit 5,10" icmp-options=129:0-255 limit=5,10:packet
protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 router solic limit 5,10 only LAN" hop-limit=equal:255
icmp-options=133:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 router advert limit 5,10 only LAN" hop-limit=equal:255
icmp-options=134:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 neighbor solic limit 5,10 only LAN" hop-limit=equal:255
icmp-options=135:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 neighbor advert limit 5,10 only LAN" hop-limit=equal:
255 icmp-options=136:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 inverse ND solic limit 5,10 only LAN" hop-limit=equal:
255 icmp-options=141:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=accept chain=icmp6 comment="defconf: rfc4890 inverse ND advert limit 5,10 only LAN" hop-limit=equal:
255 icmp-options=142:0-255 in-interface-list=LAN limit=5,10:packet protocol=icmpv6
add action=drop chain=icmp6 comment="defconf: drop other icmp" protocol=icmpv6
```

Port knocking

All available public IP addresses are constantly being port scanned by bots and services like shodan.io and anyone can use this information to perform brute force attacks and execute any known exploits. Port knocking is a cost effective way to defend against this by not exposing any ports and simply listening to connection attempts - if the correct sequence of port connection attempts is made, the client is considered safe and added to a list of secured address list that bypass the WAN firewall rules.

Setup example

We are assuming you have already set up a firewall that drops all connection attempts from the WAN port, so you will need to add additional rules before that.
First create a firewall rule that listens on a given port and adds the connected source IP to and address list - this is the first knock.

```
add action=add-src-to-address-list address-list=888 address-list-timeout=30s chain=input dst-port=888 in-
interface-list=WAN protocol=tcp
```

Then add a rule that does the same on another port, but only approves IP's that are already in the first list. You can repeat this step as many times as you like.

```
add action=add-src-to-address-list address-list=555 address-list-timeout=30s chain=input dst-port=555 in-
interface-list=WAN protocol=tcp src-address-list=888
```

Finally, the last knock will be added to an IP list that is trusted and any input is accepted.

```
add action=add-src-to-address-list address-list=secured address-list-timeout=30m chain=input dst-port=222 in-
interface-list=WAN protocol=tcp src-address-list=555
add action=accept chain=input in-interface-list=WAN src-address-list=secured
```

Knock to gain access

To access the board from WAN, a port knocking client could be used, but a simple bash one liner with nmap can do the job.

```
for x in 888,555,222; do nmap -p $x -Pn xx.xx.xx.xx; done
```

Blacklists

Unless you are using a lot of knocks, a simple port scan could accidentally trigger the correct ports in the correct order, so it is advisable to add a blacklist as well.

At the very top of your firewall stack add a drop rule for the blacklist.

```
add action=drop chain=input disabled=yes in-interface-list=WAN src-address-list=blacklist
```

Then add suspicious IP's to the blacklist.

Bad ports - ones that will never be used by a trusted user and hence have a high timeout penalty.

```
add action=add-src-to-address-list address-list=blacklist address-list-timeout=1000m chain=input disabled=yes
dst-port=666 in-interface-list=WAN protocol=tcp
```

Ports that slow down the port scanning process significantly to the point where it is pointless, but will never lock out a real user for too long. This could include every single port apart from the 'knock' ports, the key is that the source IP is not already in the secure list and hence those ports can be used after a successful knock.

```
add action=add-src-to-address-list address-list=blacklist address-list-timeout=1m chain=input disabled=yes dst-
port=21,22,23,8291,10000-60000 in-interface-list=WAN protocol=tcp src-address-list=!secured
```

Use a passphrase for each knock

You could go even further by sending a passphrase with each knock.

> **Warning**
>
> Layer7 rules are very resource intensive. Do not use unless you know what you are doing.

Then create a layer7 regex check that can be requested on the knock rule.

/ip firewall layer7-protocol add name=pass regexp="^passphrase/$"
/ip firewall filter
add action=add-src-to-address-list address-list=888 address-list-timeout=30s chain=input dst-port=888 in-interface-list=WAN protocol=udp layer7-protocol=pass

Bruteforce prevention

Here is an example of how to defend against bruteforce attacks on an ssh port. Please note, that ssh allows 3 login attempts per connection and the address lists are not cleared upon a successful login, so it is possible to blacklist yourself accidentally.

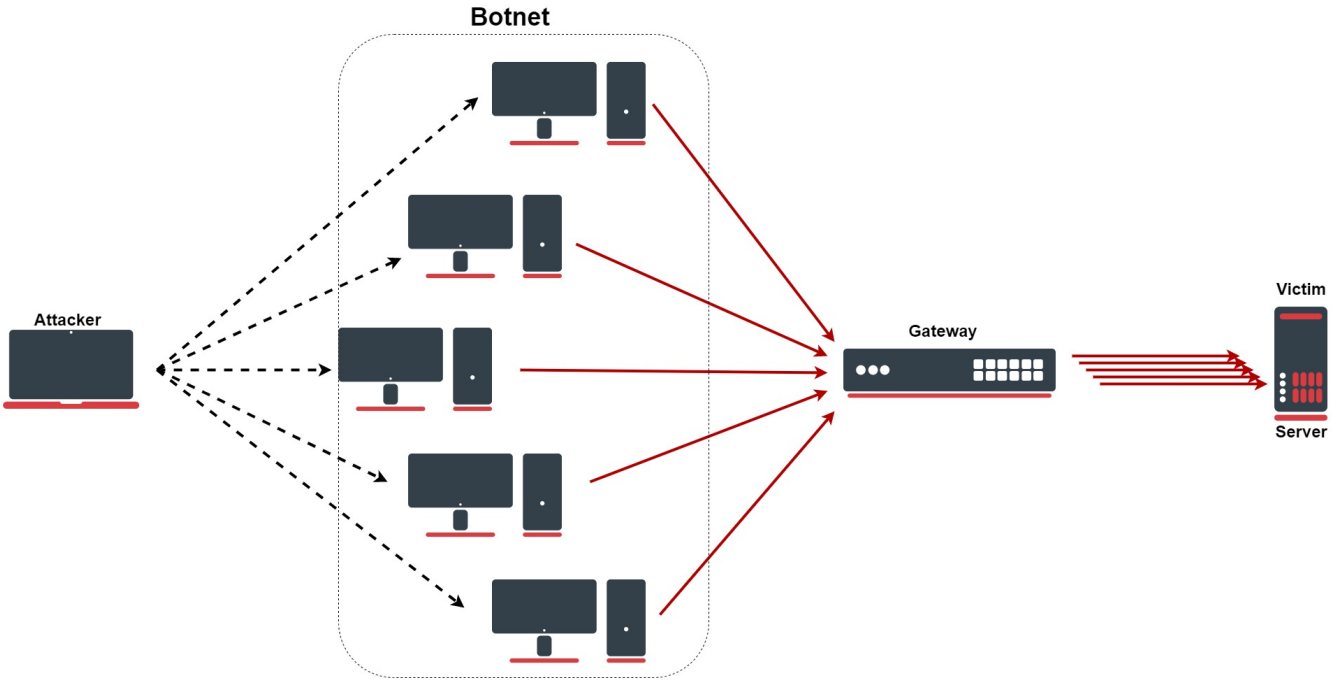
```
/ip firewall filter
add action=add-src-to-address-list address-list=bruteforce_blacklist address-list-timeout=1d chain=input
comment=Blacklist connection-state=new dst-port=22 protocol=tcp src-address-list=connection3
add action=add-src-to-address-list address-list=connection3 address-list-timeout=1h chain=input comment="Third
attempt" connection-state=new dst-port=22 protocol=tcp src-address-list=connection2,!secured
add action=add-src-to-address-list address-list=connection2 address-list-timeout=15m chain=input comment="
Second attempt" connection-state=new dst-port=22 protocol=tcp src-address-list=connection1
add action=add-src-to-address-list address-list=connection1 address-list-timeout=5m chain=input comment="First
attempt" connection-state=new dst-port=22 protocol=tcp
add action=accept chain=input dst-port=22 protocol=tcp src-address-list=!bruteforce_blacklist
```

If the timeouts were kept at 1min for all three lists - connection1/2/3 - then someone could perform 9 guesses every minute, with the above structure they can do maximum 3 guesses per 5min.

SYN/DoS/DDoS Protection

Introduction

A denial-of-service (DoS) or distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. There are several types of DDoS attacks, for example, HTTP flood, SYN flood, DNS amplification, etc.



Protection against DDoS

Configuration lines

⚠ These rules are only an improvement for firewall, do not forget to properly secure your device: Building Your First Firewall!

```
/ip firewall address-list
add list=ddos-attackers
add list=ddos-targets
/ip firewall filter
add action=return chain=detect-ddos dst-limit=32,32,src-and-dst-addresses/10s
add action=add-dst-to-address-list address-list=ddos-targets address-list-timeout=10m chain=detect-ddos
add action=add-src-to-address-list address-list=ddos-attackers address-list-timeout=10m chain=detect-ddos
/ip firewall raw
add action=drop chain=prerouting dst-address-list=ddos-targets src-address-list=ddos-attackers
```

Configuration explained

First, we will send every new connection to the specific firewall chain where we will detect DDoS:

```
/ip/firewall/filter/add chain=forward connection-state=new action=jump jump-target=detect-ddos
```

In the newly created chain, we will add the following rule with the "dst-limit" parameter. This parameter is written in the following format: **dst-limit=count[/time],burst,mode[/expire]**. We will match 32 packets with 32 packet burst based on destination and source address flow, which renews every 10 seconds. The rule will work until a given rate is exceeded.

```
/ip/firewall/filter/add chain=detect-ddos dst-limit=32,32,src-and-dst-addresses/10s action=return
```

So far all the legitimate traffic should go through the "action=return", but in the case of DoS/DDoS "dst-limit" buffer will be fulfilled and a rule will not "catch" any new traffic. Here come the next rules, which will deal with the attack. Let`s start with creating a list for attackers and victims which we will drop:

```
ip/firewall/address-list/add list=ddos-attackers
ip/firewall/address-list/add list=ddos-targets
ip/firewall/raw/add chain=prerouting action=drop src-address-list=ddos-attackers dst-address-list=ddos-targets
```

With the firewall filter section, we will add attackers in the "DDoS-attackers" and victims in list "ddos-targets" list:

```
/ip/firewall/filter/
add action=add-dst-to-address-list address-list=ddos-targets address-list-timeout=10m chain=detect-ddos
add action=add-src-to-address-list address-list=ddos-attackers address-list-timeout=10m chain=detect-ddos
```

SYN Attack

SYN Flood

An SYN flood is a form of DoS attack in which an attacker sends a succession of SYN requests to a target's system in an attempt to consume enough server resources to make the system unresponsive to legitimate traffic. Fortunately, in RouterOS we have specific feature for such an attack:

```
/ip/settings/set tcp-syncookies=yes
```

The feature works with sending back ACK packets that contain a little cryptographic hash, which the responding client will echo back with as part of its SYN-ACK packet. If the kernel doesn't see this "cookie" in the reply packet, it will assume the connection is bogus and drop it.

SYN-ACK Flood

An SYN-ACK flood is an attack method that involves sending a target server spoofed SYN-ACK packet at a high rate. The server requires significant resources to process such packets out-of-order (not in accordance with the normal SYN, SYN-ACK, ACK TCP three-way handshake mechanism), it can become so busy handling the attack traffic, that it cannot handle legitimate traffic and hence the attackers achieve a DoS/DDoS condition. In RouterOS, we can configure similar rules from the previously mentioned example, but more specifically for SYN-ACK flood:

```
/ip/firewall/filter add action=return chain=detect-ddos dst-limit=32,32,src-and-dst-addresses/10s protocol=tcp
tcp-flags=syn,ack
```

Connection rate

Introduction

Connection Rate is a firewall matcher that allows capturing traffic based on the present speed of the connection.

Theory

Each entry in the connection tracking table represents bidirectional communication. Every time packet gets associated with a particular entry, the packet size value (including IP header) is added to the "connection-bytes" value for this entry. (in other words "connection-bytes" includes both - upload and download).

Connection Rate calculates the speed of connection based on the change of "connection-bytes". The connection rate is recalculated every second and does not have any averages.

Both options "connection-bytes" and "connection-rate" work only with TCP and UDP traffic. (you need to specify a protocol to activate these options). In the "connection-rate" you can specify a range of speed that you like to capture:

```
ConnectionRate ::= [!]From-To
  From,To ::= 0..4294967295    (integer number)
```

Rule Example

These rules will capture TCP/UDP traffic that was going through the router when the connection speed was below 100kbps:

```
/ip firewall filter
add action=accept chain=forward connection-rate=0-100k protocol=tcp
add action=accept chain=forward connection-rate=0-100k protocol=udp
```

Application Example - Traffic Prioritization

Connection-rate can be used in various different ways, that still need to be realized, but the most common setup will be to detect and set lower priorities to the "heavy connections" (connections that maintain a fast rate for long periods of time (such as P2P, HTTP, FTP downloads). By doing this you can prioritize all other traffic that usually includes VOIP and HTTP browsing and online gaming.

The method described in this example can be used together with other ways to detect and prioritize traffic. As the connection-rate option does not have any averages we need to determine what will be the margin that identifies "heavy connections". If we assume that a normal HTTP browsing connection is less than 500kB (4Mb) long and VOIP requires no more than 200kbps speed, then every connection that after the first 500kB still has more than 200kbps speed can be assumed as "heavy".

(You might have different "connection-bytes" for HTTP browsing and different "connection-rate" for VOIP in your network - so, please, do your own research before applying this example)

For this example, let's assume that we have a 6Mbps upload and download connection to ISP.

Quick Start for Impatient

```
/ip firewall mangle
add chain=forward action=mark-connection connection-mark=!heavy_traffic_conn new-connection-mark=all_conn
add chain=forward action=mark-connection connection-bytes=500000-0 connection-mark=all_conn connection-
rate=200k-100M new-connection-mark=heavy_traffic_conn protocol=tcp
add chain=forward action=mark-connection connection-bytes=500000-0 connection-mark=all_conn connection-
rate=200k-100M new-connection-mark=heavy_traffic_conn protocol=udp
add chain=forward action=mark-packet connection-mark=heavy_traffic_conn new-packet-mark=heavy_traffic
passthrough=no
add chain=forward action=mark-packet connection-mark=all_conn new-packet-mark=other_traffic passthrough=no

/queue tree
add name=upload parent=public max-limit=6M
add name=other_upload parent=upload limit-at=4M max-limit=6M packet-mark=other_traffic priority=1
add name=heavy_upload parent=upload limit-at=2M max-limit=6M packet-mark=heavy_traffic priority=8
add name=download parent=local max-limit=6M
add name=other_download parent=download limit-at=4M max-limit=6M packet-mark=other_traffic priority=1
add name=heavy_download parent=download limit-at=2M max-limit=6M packet-mark=heavy_traffic priority=8
```

Explanation

In mangle, we need to separate all connections into two groups, then mark packets from their 2 groups. As we are talking about client traffic most logical place for marking would be the mangle chain forward.

Keep in mind that as soon as a "heavy" connection will have lower priority and queue will hit max-limit - heavy connection will drop speed, and connection-rate will be lower. This will result in a change to higher priority and the connection will be able to get more traffic for a short while, when again connection-rate will raise and that again will result in a change to lower priority). To avoid this we must make sure that once detected "heavy connections" will remain marked as "heavy connections" for all times.

IP Firewall mangle

This rule will ensure that that "heavy" connections will remain heavy". and mark the rest of the connections with the default connection mark:

```
/ip firewall mangle
add chain=forward action=mark-connection connection-mark=!heavy_traffic_conn new-connection-mark=all_conn
```

These two rules will mark all heavy connections based on our standards, that every connection that after the first 500kB still have more than 200kbps speed can be assumed as "heavy":

```
add chain=forward action=mark-connection connection-bytes=500000-0 \
    connection-mark=all_conn connection-rate=200k-100M new-connection-mark=heavy_traffic_conn protocol=tcp
add chain=forward action=mark-connection connection-bytes=500000-0 \
    connection-mark=all_conn connection-rate=200k-100M new-connection-mark=heavy_traffic_conn protocol=udp
```

The last two rules in mangle will simply mark all traffic from corresponding connections:

```
add chain=forward action=mark-packet connection-mark=heavy_traffic_conn new-packet-mark=heavy_traffic
passthrough=no
add chain=forward action=mark-packet connection-mark=all_conn new-packet-mark=other_traffic passthrough=no
```

Queue

This is a simple queue tree that is placed on the Interface HTB - "public" is an interface where your ISP is connected, and "local" is where are your clients. If you have more than 1 "public" or more than 1 "local" you will need to mangle upload and download separately and place the queue tree in global-out:

```
/queue tree
add name=upload parent=public max-limit=6M
add name=other_upload parent=upload limit-at=4M max-limit=6M packet-mark=other_traffic priority=1
add name=heavy_upload parent=upload limit-at=2M max-limit=6M packet-mark=heavy_traffic priority=8
add name=download parent=local max-limit=6M
add name=other_download parent=download limit-at=4M max-limit=6M packet-mark=other_traffic priority=1
add name=heavy_download parent=download limit-at=2M max-limit=6M packet-mark=heavy_traffic priority=8
```

Address-lists

Summary

```
/ip firewall address-list
```

Firewall address lists allow a user to create lists of IP addresses grouped together under a common name. Firewall filter, mangle, and NAT facilities can then use those address lists to match packets against them.

The address list records can also be updated dynamically via the action=add-src-to-address-list or action=add-dst-to-address-list items found in NAT, Mang le, and Filter facilities.

Firewall rules with action *add-src-to-address-list* or *add-dst-to-address-list* work in passthrough mode, which means that the matched packets will be passed to the next firewall rules.

Properties

Property	Description
address (*DNS Name* | *IP address/netmask* | *IP-IP*; Default:)	A single IP address or range of IPs to add to the address list or DNS name. You can input for example, '192.168.0.0-192.168.1.255' and it will auto modify the typed entry to 192.168.0.0/23 on saving.
dynamic (*yes,* no)	Allows creating data entry with dynamic form.
list (*string*; Default:)	Name for the address list of the added IP address.
timeout (*time*; Default:)	Time after address will be removed from the address list. If the timeout is not specified, the address will be stored in the address list permanently.
creation-time (*time*; Default:)	Time when the entry was created.

⚠ If the timeout parameter is not specified, then the address will be saved to the list permanently on the disk. If a timeout is specified, the address will be stored on the RAM and will be removed after a system's reboot.

Example

The following example creates a dynamic address list of people that are connecting to port 23 (telnet) on the router and drops all further traffic from them for 5 minutes. Additionally, the address list will also contain one static address list entry of 192.0.34.166/32 (www.example.com):

```
/ip firewall address-list add list=drop_traffic address=192.0.34.166/32
```

```
/ip firewall address-list print
Flags: X - disabled, D - dynamic
 #   LIST         ADDRESS
 0   drop_traffic 192.0.34.166
```

```
/ip firewall mangle add action=add-src-to-address-list address-list=drop_traffic address-list-timeout=5m
chain=prerouting dst-port=23 protocol=tcp
/ip firewall filter add action=drop chain=input src-address-list=drop_traffic
```

```
/ip firewall address-list print
Flags: X - disabled, D - dynamic
 #   LIST         ADDRESS
 0   drop_traffic 192.0.34.166
 1 D drop_traffic 1.1.1.1
 2 D drop_traffic 10.5.11.8
```

As seen in the output of the last print command, two new dynamic entries appeared in the address list (marked with a status of 'D'). Hosts with these IP addresses tried to initialize a telnet session to the router and were then subsequently dropped by the filter rule.

Layer7

Summary

Layer7-protocol is a method of searching for patterns in ICMP/TCP/UDP streams.

> ⚠ The L7 matcher is very resource-intensive. Use this feature only for very specific traffic. It is not recommended to use the L7 matcher for generic traffic, such as for blocking webpages. This will almost never work correctly and your device will exhaust its resources, trying to catch all the traffic. Use other features to block webpages by URL.

L7 matcher collects the first **10 packets** of a connection or the first **2KB** of a connection and searches for the pattern in the collected data. If the pattern is not found in the collected data, the matcher stops inspecting further. Allocated memory is freed and the protocol is considered **unknown**. You should take into account that a lot of connections will significantly increase memory and CPU usage. To avoid this, add regular firewall matchers to reduce the amount of data passed to layer-7 filters repeatedly.

An additional requirement is that the layer7 matcher must see both directions of traffic (incoming and outgoing). To satisfy this requirement l7 rules should be set in **forward** chain. If the rule is set in the **input/prerouting** chain then the same rule **must** be also set in the **output/postrouting** chain, otherwise, the collected data may not be complete resulting in an incorrectly matched pattern.

> ⛔ Layer 7 matcher is case insensitive!

Example L7 patterns compatible with RouterOS can be found on the l7-filter project page.

> ⛔ In some cases when layer 7 regular expression cannot be performed, RouterOS will log *topic=firewall, warning* with an error message stating the problem in the message!

Properties

```
/ip firewall layer7-protocol
```

Property	Description
name (*string*; Default:)	Descriptive name of l7 pattern used by configuration in firewall rules. See example >>.
regexp (*string*; Default:)	POSIX compliant regular expression is used to match a pattern.

Examples

Simple L7 usage example

First, add Regexp strings to the protocols menu, to define the strings you will be looking for. In this example, we will use a pattern to match RDP packets.

```
/ip firewall layer7-protocol
add name=rdp regexp="rdpdr.*cliprdr.*rdpsnd"
```

Then, use the defined protocols in the firewall.

```
/ip firewall filter

# add few known protocols to reduce mem usage
add action=accept chain=forward comment="" disabled=no port=80 protocol=tcp
add action=accept chain=forward comment="" disabled=no port=443 protocol=tcp

# add l7 matcher
add action=accept chain=forward comment="" disabled=no layer7-protocol=\
    rdp protocol=tcp
```

As you can see before the l7 rule we added several regular rules that will match known traffic thus reducing memory usage.

L7 in the input chain

In this example, we will try to match the telnet protocol connecting to our router.

```
/ip firewall layer7-protocol add comment="" name=telnet regexp="^\\xff[\\xfb-\\xfe].\\xff[\\xfb-\\xfe].\\xff
[\\xfb-\\xfe]"
```

Note that we need both directions that is why we need also the l7 rule in the output chain that sees outgoing packets.

```
/ip firewall filter

add action=accept chain=input comment="" disabled=no layer7-protocol=telnet \
    protocol=tcp

add action=passthrough chain=output comment="" disabled=no layer7-protocol=telnet \
    protocol=tcp
```

Youtube Matcher

> ⚠ When a user is logged in youtube will use HTTPS, meaning that L7 will not be able to match this traffic. Only unencrypted HTTP can be matched.

```
/ip firewall layer7-protocol
add name=youtube regexp="(GET \\/videoplayback\\\?|GET \\/crossdomain\\.xml)"
```

IP packing

Overview

IP Packing provides packet packaging service on network links. It allows simple packet aggregation into larger packets and compression of contents of packets.

Requirements

Packet packing is part of the system package and has to have discovery protocol enabled on an interface.

Configuration

```
/ip packing
```

It required to have a configuration in two places, both routers should be set up symmetrically:

- *ip packing* - to enable packet aggregation and/or compression on an interface
- */ip neighbor discovery*- to enable discovery protocol on the interface

Packing configuration

Property	Description
aggregated-size (*20 .. 16384 default:**1500***)	size of an aggregated packet that packing will try to achieve before sending a packet over the network
disabled (*yes|no*)	state of packing rule, if a value is *yes* it will be ignored and will not be part of the active configuration
interface (*interface name*)	packing will try to aggregate and/or compress packets from this interface
packing (*simple|compress-all|compress-headers|none*)	the action it should perform when a packet is leaving the interface packing rule is configured: • simple - do just aggregate packets • compress-all - do aggregation and attempt to compress headers and payload of a packet • compress-headers - do aggregation and attempt to compress headers and leave payload of a packet as is • none - send packets as is
unpacking (*simple|compress-all|compress-headers|none*)	the action should perform when a packet is received on the interface packing rule is configured on: • simple - unpack received packets from aggregated packets received from the interface • compress-all - unpack aggregated packet and uncompress headers and payload of a packet • compress-headers - unpack aggregated packets and decompress headers of a packet • none - do nothing with a received packet

The router should be seen as a neighbor of the router over the interface you want to enable packing on. If in the neighbor list there is no entry indicating packing, packing is not working!

Packing may increase latency on the link it is configured on.

Example

Router-A and Router-B are connected with cable with interface ether1 on Router-A and ether3 on Router-B. This example will aggregate packets coming from Router-A, but will leave packets from Router-B intact On Router-A:

Make sure discovery is enabled:

```
/ip neighbor discovery set ether1 discover=yes
```

Add packing rule for the interface:

```
/ip packing add interface=ether1 aggregated-size=1500 packing=simple unpacking=none
```

On Router-B:

Make sure discovery is enabled:

```
/ip neighbor discovery set ether3 discover=yes
```

Add packing rule for the interface:

```
/ip packing add interface=ether3 aggregated-size=1500 packing=none unpacking=simple
```

NAT-PMP

Introduction

MikroTik RouterOS supports NAT Port Mapping Protocol - NAT-PMP for transparent peer-to-peer network connectivity of personal computers and network-enabled intelligent devices or appliances.

Included in the protocol is a method for retrieving the external IPv4 address of a NAT gateway, thus allowing a client to make its external IPv4 address and port known to peers that may wish to communicate with it.

NAT-PMP uses UDP port number 5350 - on client, and 5351 on server side.

There are two interface types for PMP: **internal** (the one local clients are connected to) and **external** (the one the Internet is connected to). **A router may only have one active external interface with a 'public' IP address on it**, and as many internal interfaces as needed, all with source-NATted 'internal' IP addresses. The protocol works by creating dynamic NAT entries.

For more details on NAT PMP see RFC6886

Configuration

General properties

```
/ip nat-pmp
```

Property	Description
enabled (*yes* | *no* ; Default: **no**)	Enable NAT-PMP service

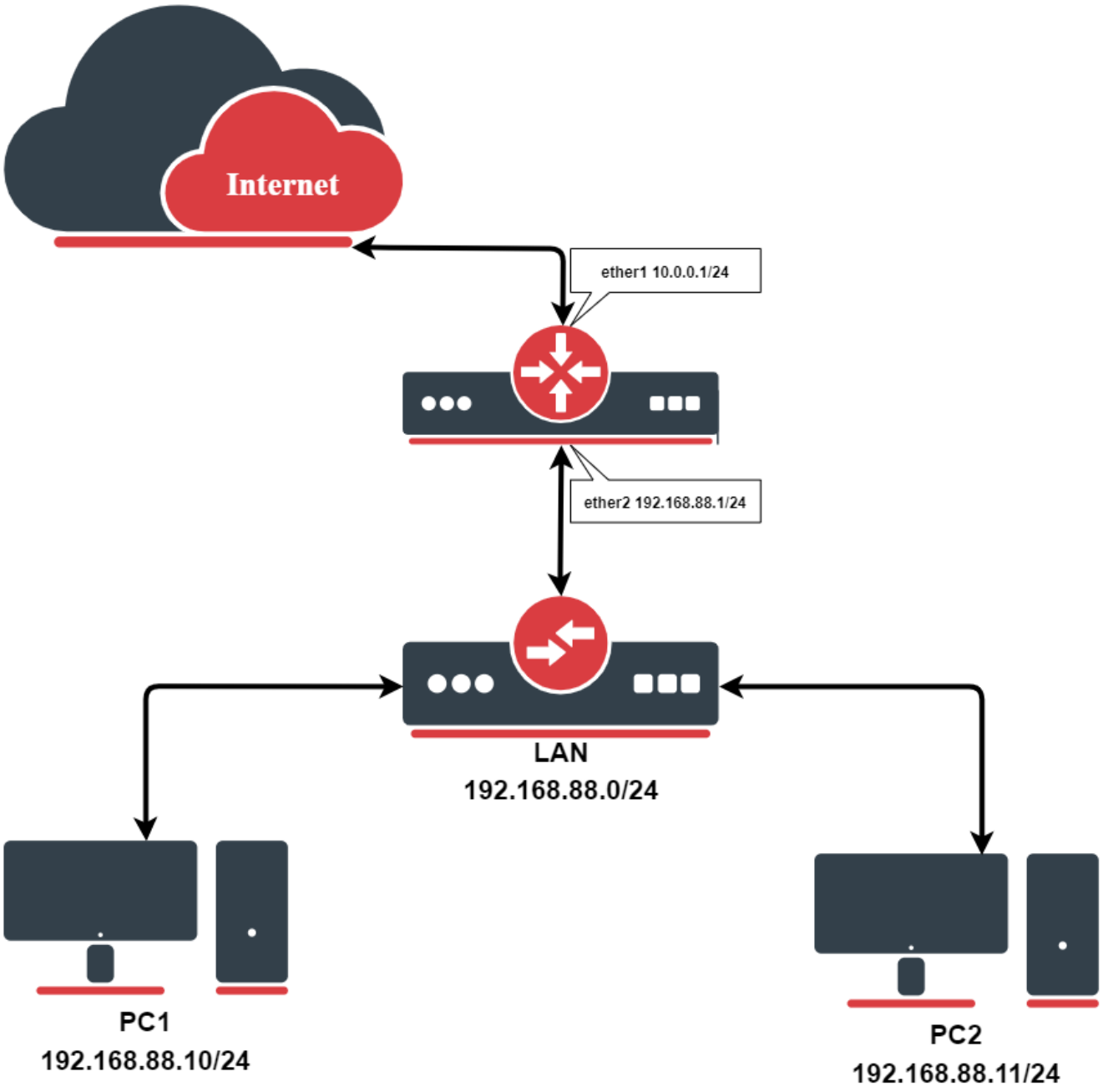
NAT PMP Interfaces

```
/ip nat-pmp interfaces
```

Property	Description
interface (*string*; Default:)	Interface name on which PMP will be running on
type (*external* | *internal*; Default: **no**)	PMP interface type: • external - the interface a global IP address is assigned to • internal - router's local interface the clients are connected to
forced-ip (*Ip*; Default:)	Allow specifying what public IP to use if the external interface has more than one IP available.

⚠ In more complex setups with VLANs, where the VLAN interface is considered as the LAN interface, the VLAN interface itself should be specified as the internal interface for PMP to work properly.

Configuration Example



We have masquerading already enabled on our router:

```
/ip firewall nat
add action=masquerade chain=srcnat out-interface=ether1
```

To enable the PMP feature:

```
[admin@MikroTik] ip nat-pmp> set enable=yes
[admin@MikroTik] ip nat-pmp> print
                             enabled: yes
```

Now, all we have to do is to add interfaces:

```
[admin@MikroTik] ip nat-pmp interfaces> add interface=ether1 type=external
[admin@MikroTik] ip nat-pmp interfaces> add interface=ether2 type=internal
[admin@MikroTik] ip nat-pmp interfaces> print
Flags: X - disabled
  #   INTERFACE TYPE
  0 X ether1    external
  1 X ether2    internal

[admin@MikroTik] ip nat-pmp interfaces> enable 0,1
```

Now once the client from the internal interface side will send PMP request dynamic NAT rules will be created on the router, example rules could look something similar to these:

```
[admin@MikroTik] > ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic

0 chain=srcnat action=masquerade out-interface=ether1

1 D ;;; nat-pmp 192.168.88.10: ApplicationX
chain=dstnat action=dst-nat to-addresses=192.168.88.10 to-ports=55000 protocol=tcp
dst-address=10.0.0.1 in-interface=ether1 dst-port=55000

2 D ;;; nat-pmp 192.168.88.10: ApplicationX
chain=dstnat action=dst-nat to-addresses=192.168.88.10 to-ports=55000 protocol=udp
dst-address=10.0.0.1 in-interface=ether1 dst-port=55000
```

High Availability Solutions

In This Section:

Bonding

Summary

Bonding is a technology that allows aggregation of multiple ethernet-like interfaces into a single virtual link, thus getting higher data rates and providing failover.

Interface bonding does not create an interface with a larger link speed. Interface bonding creates a virtual interface that can load balance traffic over multiple interfaces. More details can be found in the LAG interfaces and load balancing page.

CRS3xx, CRS5xx series switches, CCR2116, CCR2216 routers and 88E6393X, 88E6191X, 88E6190 switch chips support bridge hardware offloading with bonding interfaces. Only `802.3ad` and `balance-xor` bonding modes are hardware offloaded, other bonding modes will use the CPU's resources. The built-in switch chip will always use Layer2+Layer3+Layer4 for a transmit hash policy, changing the transmit hash policy manually will have no effect. See more details on CRS3xx, CRS5xx, CCR2116, CCR2216 switch chip features.

Quick Setup Guide

Let us assume that we have two Ethernet interfaces on each router (Router1 and Router2) and want to get the maximum data rate between these two routers. To make this possible, follow these steps:

1. Make sure that you do not have IP addresses on interfaces that will be enslaved for bonding interface.
2. Add bonding interface and IP address on the Router1:

```
/interface bonding add slaves=ether1,ether2 name=bond1
/ip address add address=172.16.0.1/24 interface=bond1
```

3. Do the same thing on the Router2:

```
/interface bonding add slaves=ether1,ether2 name=bond1
/ip address add address=172.16.0.2/24 interface=bond1
```

4. Test the link from Router1:

```
[admin@Router1] > ping 172.16.0.2
  SEQ HOST                                     SIZE TTL TIME  STATUS
    0 172.16.0.2                                 56  64 0ms
    1 172.16.0.2                                 56  64 0ms
    2 172.16.0.2                                 56  64 0ms
    sent=3 received=3 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```


The bonding interface needs a couple of seconds to get connectivity with its peers.

Link monitoring

It is critical that one of the available link monitoring options is enabled. In the above example, if one of the bonded links were to fail, the bonding driver will still continue to send packets over the failed link which will lead to network degradation. Bonding in RouterOS currently supports two schemes for monitoring a link state of slave devices: MII and ARP monitoring. It is not possible to use both methods at the same time due to restrictions in the bonding driver.

ARP Monitoring

ARP monitoring sends ARP queries and uses the response as an indication that the link is operational. The ARP replies are not validated, any received packet by the slave interface will result in the slave interface considered as active. This gives assurance that traffic is actually flowing over the links. If balance-rr and balance-xor modes are set, then the switch should be configured to evenly distribute packets across all links. Otherwise, all replies from the ARP targets will be received on the same link which could cause other links to fail. ARP monitoring is enabled by setting three properties - link-monitoring, a rp-ip-targets and arp-interval. The meaning of each option is described later in this article. It is possible to specify multiple ARP targets that can be useful in High Availability setups. If only one target is set, the target itself may go down. Having additional targets increases the reliability of the ARP monitoring.

To enable ARP monitoring on Router1:

```
/interface bonding set [find name=bond1] link-monitoring=arp arp-ip-targets=172.16.0.2
```

and Router2:

```
/interface bonding set [find name=bond1] link-monitoring=arp arp-ip-targets=172.16.0.1
```

We will not change the arp-interval value in our example, RouterOS sets arp-interval to 100ms by default. Unplug one of the cables to test if the link monitoring works correctly, you might notice some ping timeouts until arp monitoring detects link failure.

```
[admin@MikroTik] > ping 172.16.0.2
  SEQ HOST                                     SIZE TTL TIME  STATUS
    0 172.16.0.2                                 56  64 0ms
    1 172.16.0.2                                 56  64 0ms
    2 172.16.0.2                                 56  64 0ms
    3 172.16.0.2                                 56  64 0ms
    4 172.16.0.2                                             timeout
    5 172.16.0.2                                 56  64 0ms
    6 172.16.0.2                                 56  64 0ms
    sent=7 received=6 packet-loss=14% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```


For ARP monitoring to work properly it is not required to have any IP address on the device, ARP monitoring will work regardless of the IP address that is set on any interface.

When ARP monitoring is used, bonding slaves will send out ARP requests without a VLAN tag, even if an IP address is set on a VLAN interface in the same subnet as the arp-ip-targets

MII monitoring

MII monitoring monitors only the state of the local interface. *MII Type 1* - a device driver determines whether a link is up or down. If the device driver does not support this option then the link will appear as always up. The main disadvantage is that MII monitoring can't tell if the link can actually pass packets or not, even if the link is detected as being up. MII monitoring is configured by setting the variables - link-monitoring and mii-interval.

To enable MII Type1 monitoring on Router1 and Router2:

```
/interface bonding set [find name=bond1] link-monitoring=mii
```

We will leave mii-interval to its default value (100ms). When unplugging one of the cables, the failure will be detected almost instantly compared to ARP link monitoring.

Bonding modes

802.3ad

802.3ad mode is an IEEE standard also called LACP (Link Aggregation Control Protocol). It includes automatic configuration of the aggregates, so minimal configuration of the switch is needed. This standard also mandates that frames will be delivered in order and connections should not see misordering of packets. The standard also mandates that all devices in the aggregate must operate at the same speed and duplex mode.

LACP balances outgoing traffic across the active ports based on hashed protocol header information and accepts incoming traffic from any active port. The hash includes the Ethernet source and destination address and if available, the VLAN tag, and the IPv4/IPv6 source and destination address. How this is calculated depends on transmit-hash-policy parameter. The ARP link monitoring is not recommended, because the ARP replies might arrive only on one slave port due to transmit hash policy on the LACP peer device. This can result in unbalanced transmitted traffic, so MII link monitoring is the recommended option.

The layer-3-and-4 transmit hash mode is not fully compatible with LACP. More details can be found in https://www.kernel.org/doc/Documentation/networking/bonding.txt

balance-xor

This mode balances outgoing traffic across the active ports based on the hashed protocol header information and accepts incoming traffic from any active port. The mode is very similar to LACP except that it is not standardized and works with **layer-3-and-4** hash policy. The mode can work together with static Link Aggregation Group (LAG) interfaces.

balance-rr

If this mode is set, packets are transmitted in sequential order from the first available slave to the last. The balance-rr is the only mode that will send packets across multiple interfaces that belong to the same TCP/IP connection. When utilizing multiple sending and multiple receiving links, packets are often received out of order, which results in segment retransmission, for other protocols such as UDP it is not a problem if a client software can tolerate out-of-order packets. If a switch is used to aggregate links together, then appropriate switch port configuration is required, however many switches do not support balance-rr. Quick setup guide demonstrates the usage of the balance-rr bonding mode. As you can see, it is quite simple to set up. Balance-rr is also useful for bonding several wireless links, however, it requires equal bandwidth for all bonded links. If the bandwidth of one bonded link drops, then the total bandwidth of bond will be equal to the bandwidth of the slowest bonded link.

active-backup

This mode uses only one active slave to transmit packets. The additional slave only becomes active if the primary slave fails. The MAC address of the bonding interface is presented onto the active port to avoid confusing the switch. Active-backup is the best choice in high availability setups with multiple switches that are interconnected.

⚠ The ARP monitoring in this mode will not work correctly if both routers are directly connected. In such setups, MII monitoring must be used or a switch should be put between routers.

broadcast

When ports are configured with broadcast mode, all slave ports transmit the same packets to the destination to provide fault tolerance. This mode does not provide load balancing.

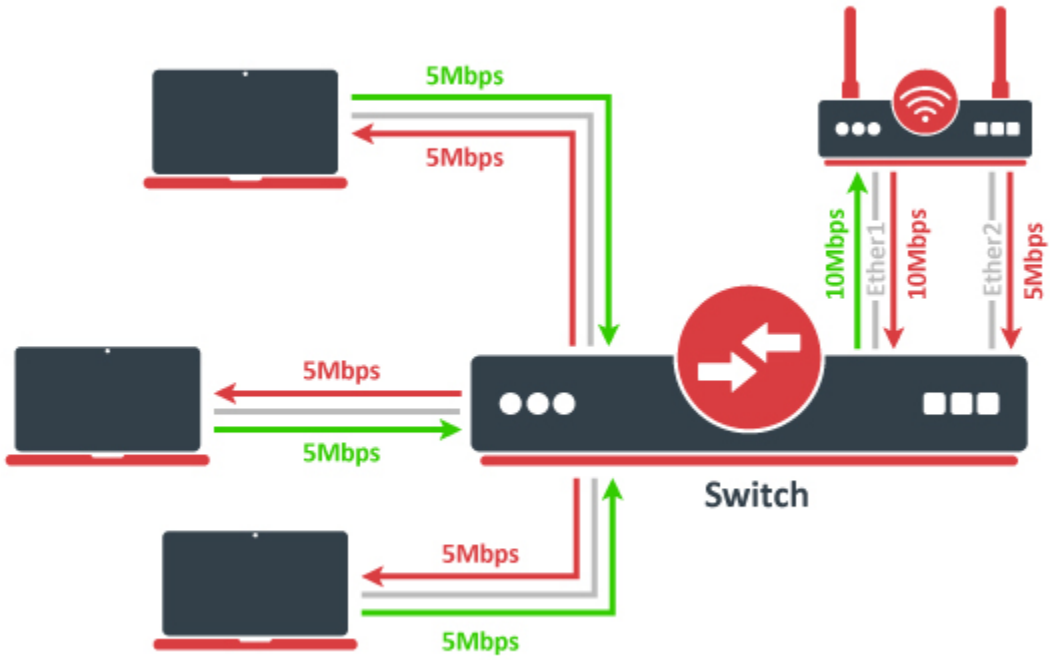
balance-tlb

This mode balances outgoing traffic by peer. Each link can be a different speed and duplex mode and no specific switch configuration is required as for the other modes. The downside of this mode is that only MII link monitoring is supported (ARP link monitoring is ignored when configured) and incoming traffic is not balanced. Incoming traffic will use the link that is configured as "primary".

Configuration example

Let's assume that the router has two links - **ether1** max bandwidth is 10Mbps and **ether2** max bandwidth is 5Mbps. The first link has more bandwidth so we set it as a primary link:

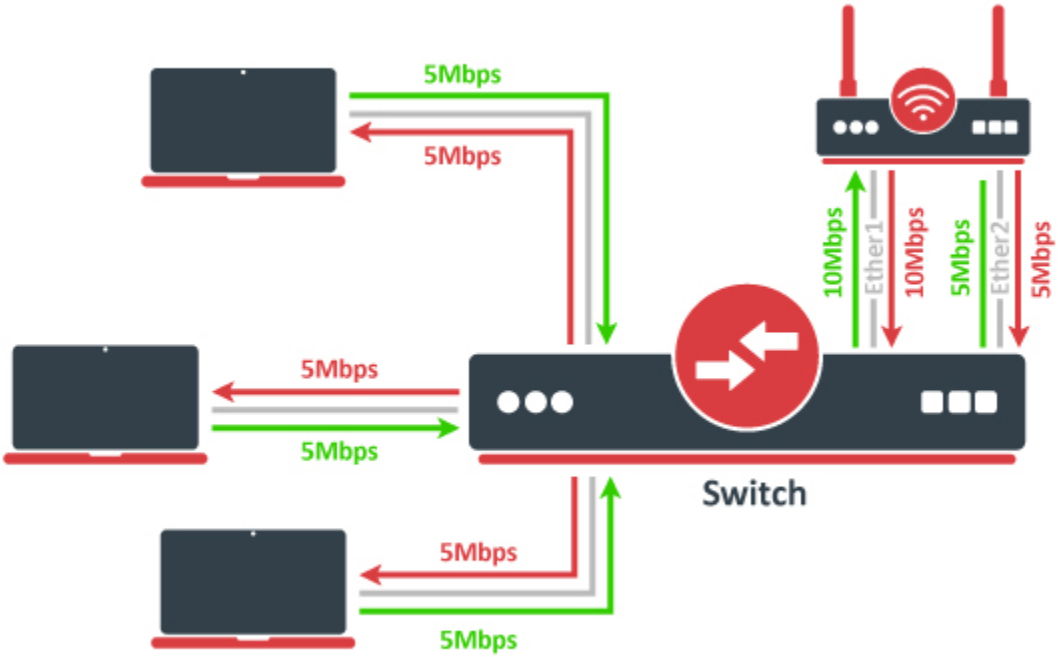
```
/interface bonding add mode=balance-tlb slaves=ether1,ether2 primary=ether1
```



No additional configuration is required for the switch. The image above illustrates how balance-tlb mode works. As you can see router can communicate to all the clients connected to the switch with a total bandwidth of both links (15Mbps). But as you already know, balance-tlb is not balancing incoming traffic. In our example, clients can communicate to the router with a total bandwidth of primary link which is 10Mbps in our configuration.

balance-alb

The mode is basically the same as balance-tlb but incoming IPv4 traffic is also balanced. The receive load balancing is achieved by ARP negotiation. The bonding driver intercepts locally generated ARP messages on their way out and overwrites the source hardware address with the unique address of one of the slaves in the bond such that different peers use different hardware addresses. Only MII link monitoring is supported (ARP link monitoring is ignored when configured), the additional downside of this mode is that it requires device driver capability to change MAC address. The mode is not compatible with local-proxy-arp setting.



The image above illustrates how balance-alb mode works. Compared to balance-tlb mode, traffic from clients can also use the secondary link to communicate with the router.

Bonding monitoring

Since RouterOS 6.48 version, it is possible to monitor the bonding interface and bonding ports. For the `802.3ad` bonding mode, more detailed monitoring options are available.

```
/interface bonding monitor [find]
                     mode: 802.3ad           active-backup
             active-ports: ether4            ether6
                           ether5
           inactive-ports:                   ether7
           lacp-system-id: CC:2D:E0:11:22:33
     lacp-system-priority: 65535
   lacp-partner-system-id: B8:69:F4:44:55:66
```

Property	Description
mode (*802.3ad* / *active-backup* / *balance-alb* / *balance-rr* / *balance-tlb* / *balance-xor* / *broadcast*)	Used bonding mode
active-ports (*interface*)	Shows the active bonding ports
inactive-ports (*interface*)	Shows the inactive bonding ports (e.g. a disabled or backup interface)
lacp-system-id (*MAC address*)	Shows the local LACP system ID

lacp-system-priority (*integer*)	Shows the local LACP priority
lacp-partner-system-id (*MAC address*)	Shows the partner LACP system ID

To monitor individual bonding ports, use a `monitor-slaves` command.

```
/interface bonding monitor-slaves bond1
Flags: A - active, P - partner
 AP port=ether4 key=17 flags="A-GSCD--" partner-sys-id=D4:CA:6D:12:06:65 partner-sys-priority=65535 partner-
key=9 partner-flags="A-GSCD--"

 AP port=ether5 key=17 flags="A-GSCD--" partner-sys-id=D4:CA:6D:12:06:65 partner-sys-priority=65535 partner-
key=9 partner-flags="A-GSCD--"
```

Property	Description
port (*interface*)	Used bonding port
key (*integer*)	Shows the local LACP aggregation key. The lower 6 bits are automatically assigned based on individual port link speed and duplex. The upper 10 bits can be manually specified using the `lacp-user-key` setting (available only since RouterOS v7.3).
flags (*string*)	Shows the local LACP flags: A - activity (link is active, otherwise passive) T - timeout (link is using short 1-second timeout, otherwise using 30-second timeout) G - aggregation (link can be aggregatable) S - synchronization (link is synchronized) C - collecting (link is able to collect incoming frames) D - distributing (link is able to distribute outgoing frames) F - defaulted (link is using defaulted partner information, indicated that no LACPDU has been received from the partner) E - expired (link has expired state)
partner-sys-id (*MAC address*)	Shows the partner LACP system ID
partner-sys-priority (*integer*)	Shows the partner LACP priority
partner-key (*integer*)	Shows the partner LACP aggregation key
partner-flags (*string*)	Shows the partner LACP flags

Property Description

This section describes the available bonding settings.

Property	Description
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	Address Resolution Protocol for the interface. • disabled - the interface will not use ARP • enabled - the interface will use ARP • proxy-arp - the interface will use the ARP proxy feature • reply-only - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the "/ip arp" table. No dynamic entries will be automatically stored in the "/ip arp" table. Therefore for communications to be successful, a valid static entry must already exist.
arp-interval (*time*; Default: **00:00:00.100**)	Time in milliseconds defines how often to monitor ARP requests

arp-ip-targets (*IP address*; Default:)	IP target address which will be monitored if link-monitoring is set to arp. You can specify multiple IP addresses, separated by a comma
comment (*string*; Default:)	Short description of the interface
disabled (*yes* | *no*; Default: **no**)	Changes whether the bonding interface is disabled
down-delay (*time*; Default: **00:00:00**)	If a link failure has been detected, the bonding interface is disabled for a down-delay time. The value should be a multiple of mi i-interval, otherwise, it will be rounded down to the nearest value. This property only has an effect when `link-monitoring` is set to `mii`.
forced-mac-address (*MAC address*; Default: **none**)	By default, the bonding interface will use the MAC address of the first selected slave interface. This property allows to configure static MAC address for the bond interface (all zeros, broadcast or multicast addresses will not apply). RouterOS will automatically change the MAC address for slave interfaces and it will be visible in `/interface ethernet` configuration export
lacp-rate (*1sec* | *30secs*; Default: **30secs**)	Link Aggregation Control Protocol rate specifies how often to exchange with LACPDUs between bonding peers. Used to determine whether a link is up or other changes have occurred in the network. LACP tries to adapt to these changes providing failover.
lacp-user-key (*integer: 0..1023*; Default: **0**)	Specifies the upper 10 bits of the port key. The lower 6 bits are automatically assigned based on individual port link speed and duplex. The setting is available only since RouterOS v7.3.
link-monitoring (*arp* | *mii* | *none*; Default: **mii**)	Method to use for monitoring the link (whether it is up or down) • arp - uses Address Resolution Protocol to determine whether the remote interface is reachable • mii - uses Media Independent Interface to determine link status. Link status determination relies on the device driver. • none - no method for link monitoring is used. **Note:** some bonding modes require specific link monitoring to work properly.
min-links (*integer: 0..4294967295*; Default: **0**)	How many active slave links needed for bonding to become active
mii-interval (*time*; Default: **00:00:00.100**)	How often to monitor the link for failures (the parameter used only if link-monitoring is mii)
mlag-id (*integer: 0..4294967295;* Default:)	Changes MLAG ID for bonding interface. The same MLAG ID should be used on both peer devices to successfully create a single MLAG. See more details on MLAG.
mode (*802.3ad* | *active-backup* | *balance-alb* | *balance-rr* | *balance-tlb* | *balance-xor* | *broadcast*; Default: **balance-rr**)	Specifies one of the bonding policies • 802.3ad - IEEE 802.3ad dynamic link aggregation. In this mode, the interfaces are aggregated in a group where each slave shares the same speed. It provides fault tolerance and load balancing. Slave selection for outgoing traffic is done according to the transmit-hash-policy more>• active-backup - provides link backup. Only one slave can be active at a time. Another slave only becomes active, if the first one fails. more>• balance-alb - adaptive load balancing. The same as balance-tlb but received traffic is also balanced. The device driver should have support for changing it's MAC address. more>• balance-rr - round-robin load balancing. Slaves in a bonding interface will transmit and receive data in sequential order. It provides load balancing and fault tolerance. more>• balance-tlb - Outgoing traffic is distributed according to the current load on each slave. Incoming traffic is not balanced and is received by the current slave. If receiving slave fails, then another slave takes the MAC address of the failed slave. more>• balance-xor - Transmit based on the selected transmit-hash-policy. This mode provides load balancing and fault tolerance. more>• broadcast - Broadcasts the same data on all interfaces at once. This provides fault tolerance but slows down traffic throughput on some slow machines. more>
mtu (*integer*; Default: **1500**)	Maximum Transmit Unit in bytes. Must be smaller or equal to the smallest L2MTU value of a bonding slave. L2MTU of a bonding interface is determined by the lowest L2MTU value among its slave interfaces
name (*string*; Default:)	Name of the bonding interface

primary (*string*; Default: **none**)	Controls the primary interface between active slave ports, works only for active-backup, balance-tlb and balance-alb modes. For active-backup mode, it controls which running interface is supposed to send and receive the traffic. For balance-tlb mode, it controls which running interface is supposed to receive all the traffic, but for balance-alb mode, it controls which interface is supposed to receive the unbalanced traffic (the non-IPv4 traffic). When none of the interfaces are selected as primary, device will automatically select the interface that is configured as the first one.
slaves (*string*; Default: **none**)	At least two ethernet-like interfaces separated by a comma, which will be used for bonding
up-delay (*time*; Default: **00:00:00**)	If a link has been brought up, the bonding interface is disabled for up-delay time and after this time it is enabled. The value should be a multiple of mii-interval, otherwise, it will be rounded down to the nearest value. This property only has an effect when `link-monitoring` is set to `mii`.
transmit-hash-policy (*layer-2 | layer-2-and-3 | layer-3-and-4*; Default: **layer-2**)	Selects the transmit hash policy to use for slave selection in balance-xor and 802.3ad modes • layer-2 - Uses XOR of hardware MAC addresses to generate the hash. This algorithm will place all traffic to a particular network peer on the same slave. This algorithm is 802.3ad compliant. • layer-2-and-3 - This policy uses a combination of layer2 and layer3 protocol information to generate the hash. Uses XOR of hardware MAC addresses and IP addresses to generate the hash. This algorithm will place all traffic to a particular network peer on the same slave. For non-IP traffic, the formula is the same as for the layer2 transmit hash policy. This policy is intended to provide a more balanced distribution of traffic than layer2 alone, especially in environments where a layer3 gateway device is required to reach most destinations. This algorithm is 802.3ad compliant. • layer-3-and-4 - This policy uses upper layer protocol information, when available, to generate the hash. This allows for traffic to a particular network peer to span multiple slaves, although a single connection will not span multiple slaves. For fragmented TCP or UDP packets and all other IP protocol traffic, the source and destination port information is omitted. For non-IP traffic, the formula is the same as for the layer2 transmit hash policy. This algorithm is not fully 802.3ad compliant.

See also

- Bonding presentation at the MUM
- Bonding Examples

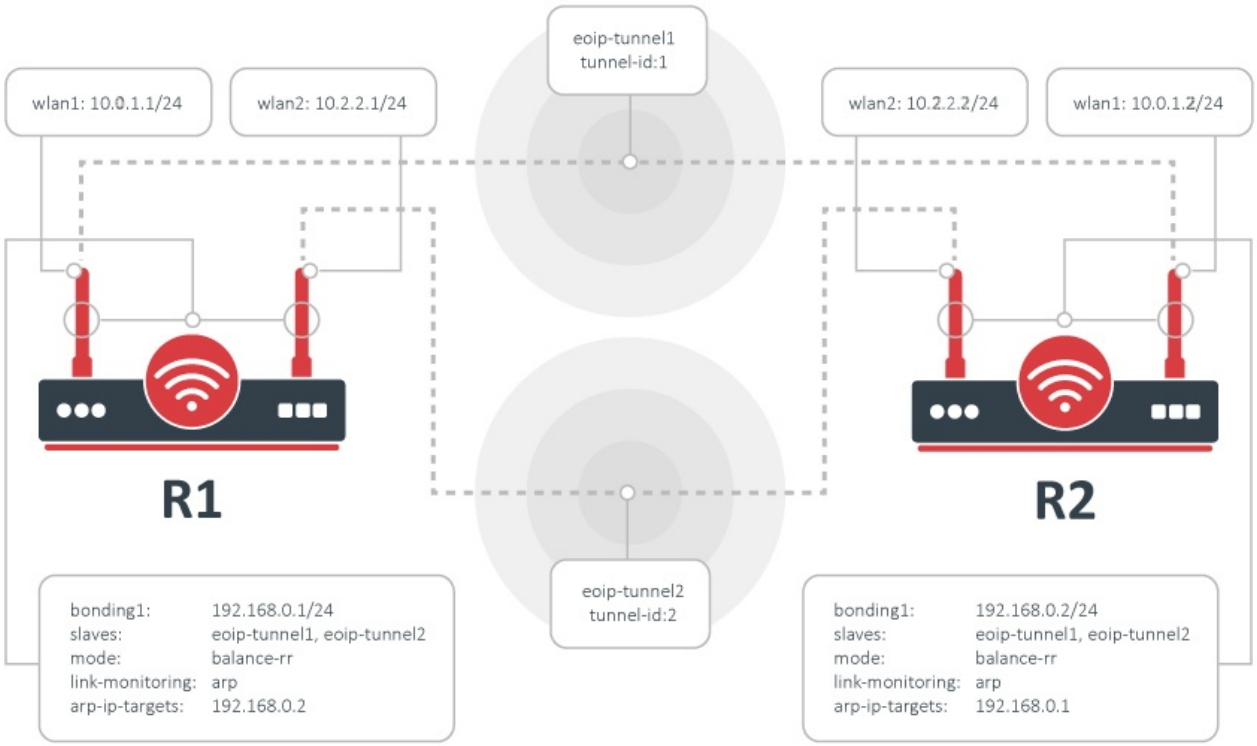
Bonding Examples

Bonding EoIP tunnels over two wireless links

This is an example of aggregating multiple network interfaces into a single pipe. In particular, it is shown how to aggregate multiple virtual (EoIP) interfaces to get maximum throughput (MT) with emphasis on availability.

Network Diagram

Two routers R1 and R2 are interconnected via wireless links. Wireless interfaces on both sides have assigned IP addresses.



Configuration

Bonding could be used only on OSI layer 2 (Ethernet level) connections. Thus we need to create EoIP interfaces on each of the wireless links. This is done as follows:

on router R1:

```
/interface eoip add remote-address=10.0.1.1/24 tunnel-id=1
/interface eoip add remote-address=10.2.2.1/24 tunnel-id=2
```

and on router R2:

```
/interface eoip add remote-address=10.0.1.2/24 tunnel-id=1
/interface eoip add remote-address=10.2.2.2/24 tunnel-id=2
```

The second step is to add a bonding interface and specify EoIP interfaces as slaves:

R1:

```
/interface bonding add slaves=eoip-tunnel1,eoip-tunnel2 mode=balance-rr
```

R2:

```
/interface bonding add slaves=eoip-tunnel1,eoip-tunnel2 mode=balance-rr
```

The last step is to add IP addresses to the bonding interfaces:

R1:

```
/ip address add address 192.168.0.1/24 interface=bonding1
```

R2:

```
/ip address add address 192.168.0.2/24 interface=bonding1
```

Test the configuration

Now two routers are able to reach each other using addresses from the 192.168.0.0/24 network. To verify bonding interface functionality, do the following:

R1:

```
/interface monitor-traffic eoip-tunnel1,eoip-tunnel2
```

R2:

```
/tool bandwidth-test 192.168.0.1 direction=transmit
```

You should see that traffic is distributed equally across both EoIP interfaces:

```
/int monitor-traffic eoip-tunnel1,eoip-tunnel2
    received-packets-per-second: 685      685
       received-bits-per-second: 8.0Mbps  8.0Mbps
        sent-packets-per-second: 21       20
           sent-bits-per-second: 11.9kbps 11.0kbps
    received-packets-per-second: 898      899
       received-bits-per-second: 10.6Mbps 10.6Mbps
        sent-packets-per-second: 20       21
           sent-bits-per-second: 11.0kbps 11.9kbps
    received-packets-per-second: 975      975
       received-bits-per-second: 11.5Mbps 11.5Mbps
        sent-packets-per-second: 22       22
           sent-bits-per-second: 12.4kbps 12.3kbps
    received-packets-per-second: 980      980
       received-bits-per-second: 11.6Mbps 11.6Mbps
        sent-packets-per-second: 21       21
           sent-bits-per-second: 11.9kbps 11.8kbps
    received-packets-per-second: 977      977
       received-bits-per-second: 11.6Mbps 11.5Mbps
        sent-packets-per-second: 21       21
           sent-bits-per-second: 11.9kbps 11.8kbps
-- [Q quit|D dump|C-z pause]
```

Link Monitoring

It is easy to notice that with the configuration above as soon as any individual link fails, the bonding interface throughput collapses. That's because no link monitoring is performed, consequently, the bonding driver is unaware of problems with the underlying links. Enabling link monitoring is a must in most bonding configurations. To enable ARP link monitoring, do the following:

R1:

```
/interface bonding set bonding1 link-monitoring=arp arp-ip-targets=192.168.0.2
```

R2:

```
/interface bonding set bonding1 link-monitoring=arp arp-ip-targets=192.168.0.1
```

HA Case Studies

Multi-chassis Link Aggregation Group

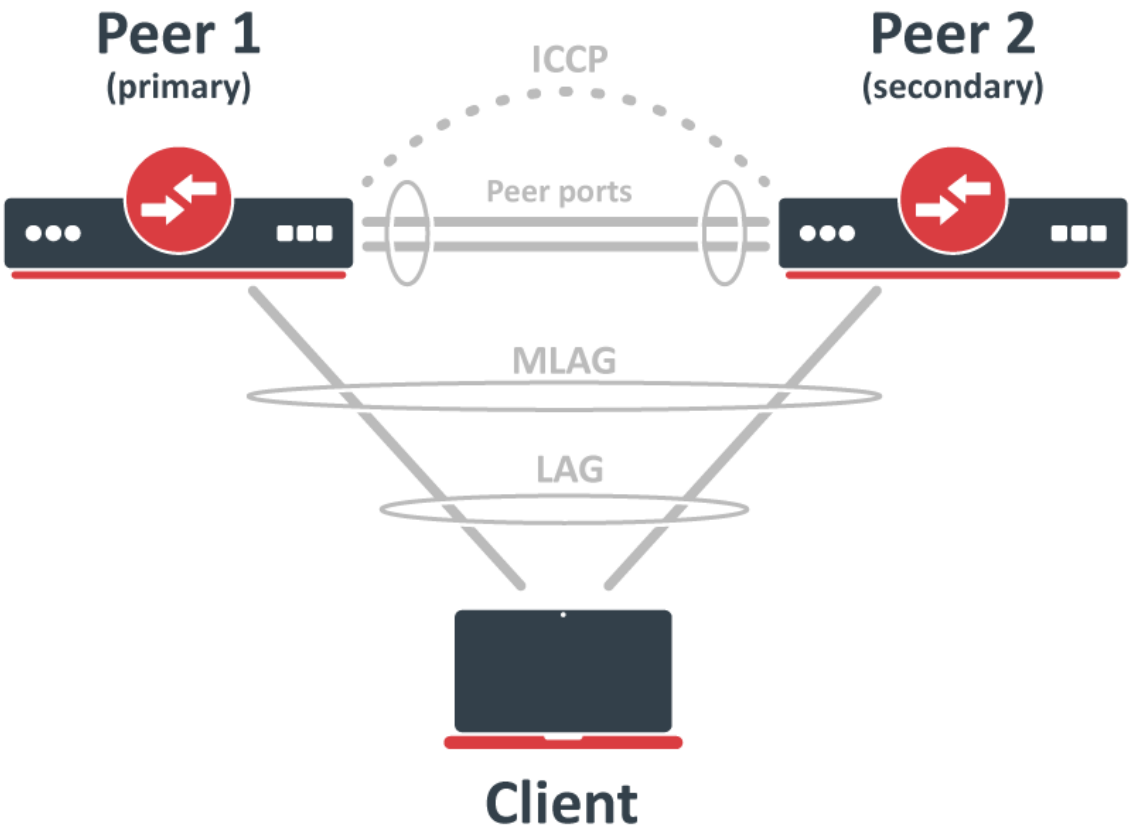
Introduction

MLAG (Multi-chassis Link Aggregation Group) implementation in RouterOS allows configuring LACP bonds on two separate devices, while the client device believes to be connected to the same machine. This provides a physical redundancy in case of switch failure. All CRS3xx, CRS5xx series switches, and CCR2116, CCR2216 devices can be configured with MLAG using RouterOS version 7.

Both peers establish the MLAG interfaces and update the bridge host table over `peer-port` using ICCP (Inter Chassis Control Protocol). RouterOS ICCP does not require an IP configuration, but it should be isolated from the rest of the network using a dedicated untagged VLAN. This untagged VLAN can be configured with `vlan-filtering` and `pvid`. Peer ports can also be configured as LACP bonding interfaces.

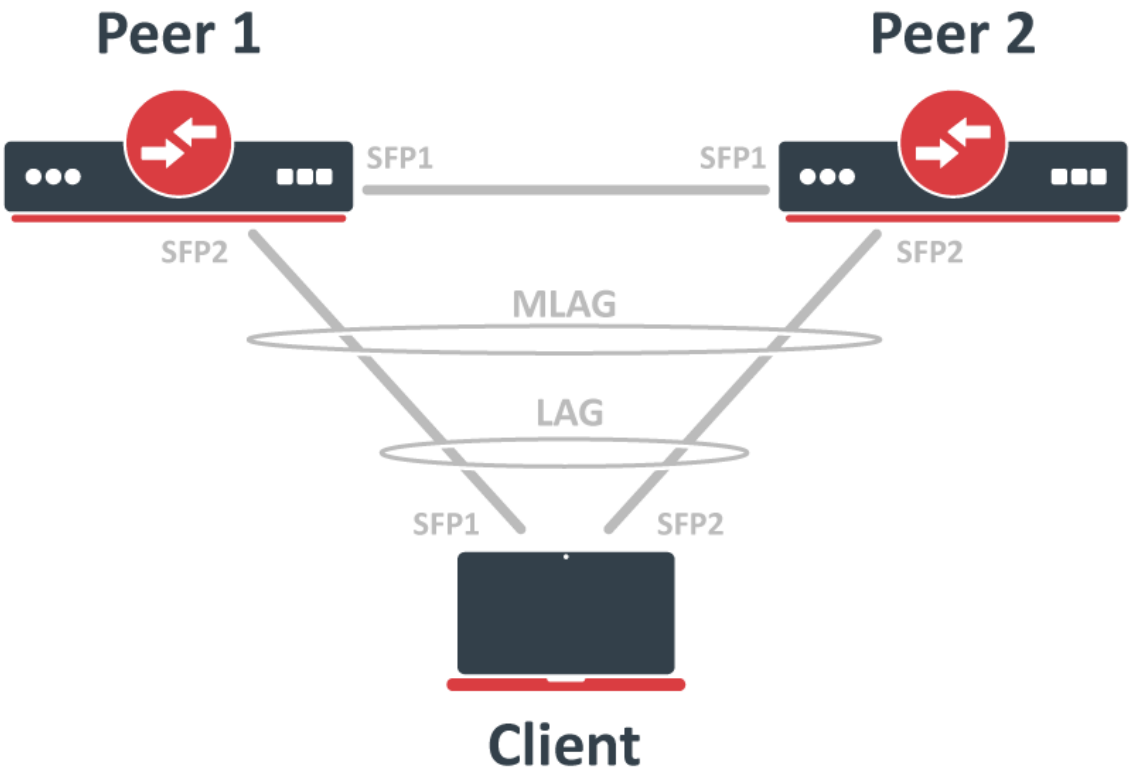
When `peer-port` is running and ICCP is established, the primary device election happens. The peer with the lowest bridge MAC address will be acting as a primary device and `system-id` will be selected. This `system-id` is used for STP BPDU bridge identifier and LACP system ID. The MLAG requires enabled STP, RSTP or MSTP protocol. Use the same STP priority and the same STP configuration on dual-connected bridge ports on both nodes. When MLAG bridges are elected as STP root, then both devices will show as root bridges under the bridge monitor.

The MLAG is not compatible with L3 hardware offloading. When using MLAG, the L3 hardware offloading must be disabled.



Quick setup

in this example, CRS317 and CRS309 devices are used as MLAG peers and any device with two SFP+ interfaces can be used as an LACP client. The SFP+1 interface is used on both peer nodes to create `peer-port`, and it is used for ICCP, see a network scheme below.



Below are configuration commands to create a regular LACP bonding in RouterOS for the Client device:

```
/interface bonding
add mode=802.3ad name=bond1 slaves=sfp-sfpplus1,sfp-sfpplus2
```

Next, configure bonding interfaces for MLAG on Peer1 and Peer2 devices, use a matching `mlag-id` setting on both peer devices:

```
# Peer1
/interface bonding
add mlag-id=10 mode=802.3ad name=client-bond slaves=sfp-sfpplus2

# Peer2
/interface bonding
add mlag-id=10 mode=802.3ad name=client-bond slaves=sfp-sfpplus2
```

Configure bridge with enabled `vlan-filtering`, and add needed interfaces as bridge ports. A dedicated untagged VLAN should be applied for the inter-chassis communication on a peer port, thus a different `pvid` setting is used. Below are configuration commands for Peer1 and Peer2 devices:

```
# Peer1
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=sfp-sfpplus1 pvid=99
add bridge=bridge1 interface=client-bond

# Peer2
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=sfp-sfpplus1 pvid=99
add bridge=bridge1 interface=client-bond
```

The MLAG requires enabled STP, RSTP or MSTP protocol. Use the same STP priority and the same STP configuration on dual-connected bridge ports on both nodes.

In this example, client-bond interfaces are using the default untagged VLAN 1 (the default `pvid=1` is set). In order to send these packets over peer ports, we need to add them as tagged VLAN 1 members. Notice that the default `pvid` value for the peer ports was changed in the previous step, it is important to include the peer ports in all the VLANs that are used on other bridge ports, this includes the untagged and tagged VLANs. Below are configuration commands for both peer devices:

```
# Peer1
/interface bridge vlan
add bridge=bridge1 tagged=sfp-sfpplus1 vlan-ids=1

# Peer2
/interface bridge vlan
add bridge=bridge1 tagged=sfp-sfpplus1 vlan-ids=1
```

All VLANs used for bridge slave ports must be also configured as tagged VLANs for peer-port, so that peer-port is a member of those VLANs and can forward data.

Last, specify `bridge` and `peer-port` to enable MLAG. Below are configuration commands for both peer devices:

```
# Peer1
/interface bridge mlag
set bridge=bridge1 peer-port=sfp-sfpplus1

# Peer2
/interface bridge mlag
set bridge=bridge1 peer-port=sfp-sfpplus1
```

Additionally, check MLAG status on peer devices and make sure that Client LACP has both interfaces active.

```
# Peer1
[admin@Peer1] > /interface/bridge/mlag/monitor
       status: connected
    system-id: 74:4D:28:11:70:6B
  active-role: primary

# Peer2
[admin@Peer2] > /interface/bridge/mlag/monitor
       status: connected
    system-id: 74:4D:28:11:70:6B
  active-role: secondary

# Client
[admin@Client] > /interface bonding monitor bond1
                    mode: 802.3ad
            active-ports: sfp-sfpplus1,sfp-sfpplus2
          inactive-ports:
          lacp-system-id: 74:4D:28:7B:7F:96
    lacp-system-priority: 65535
  lacp-partner-system-id: 74:4D:28:11:70:6C
```

MLAG settings and monitoring

This section describes the available MLAG settings and monitoring options.

Sub-menu: `/interface bridge mlag`

Property	Description
bridge (*interface;* Default: **none**)	The bridge interface where MLAG is being created.
peer-port (*interface;* Default: **none**)	An interface that will be used as a peer port. Both peer devices are using inter-chassis communication over these peer ports to establish MLAG and update the host table. Peer port should be isolated on a different untagged VLAN using a `pvid` setting. Peer port can be configured as a bonding interface.

Use the `monitor` commands to see the current MLAG status.

```
[admin@Peer1] > /interface/bridge/mlag/monitor
       status: connected
    system-id: 74:4D:28:11:70:6B
  active-role: primary
```

Property	Description
status (*connected | connecting | disabled*)	The MLAG status.
system-id (*MAC address*)	The lowest MAC address between both peer bridges will be used as the `system-id`. This `system-id` is used for (R)STP BPDU bridge identifier and LACP system ID.
active-role (*primary | secondary*)	The peer with the lowest bridge MAC address will be acting as a primary device. The `system-id` of the primary device is used for sending the (R)STP BPDU bridge identifier and LACP system ID.

Sub-menu: `/interface bonding`

Property	Description
mlag-id (*integer: 0..4294967295;* Default:)	Changes MLAG ID for bonding interface. The same MLAG ID should be used on both peer devices to successfully create a single LAG for the client device. The `peer-port` should not be configured with the MLAG ID.

LACP bonding interface and bonding slave ports can be monitored with `monitor` and `monitor-slaves` commands. See more details on Bonding monitoring.

VRRP

Summary

This chapter describes the Virtual Router Redundancy Protocol (VRRP) support in RouterOS.

Mostly on larger LANs dynamic routing protocols (OSPF or RIP) are used, however, there are a number of factors that may make it undesirable to use dynamic routing protocols. One alternative is to use static routing, but if the statically configured first hop fails, then the host will not be able to communicate with other hosts.

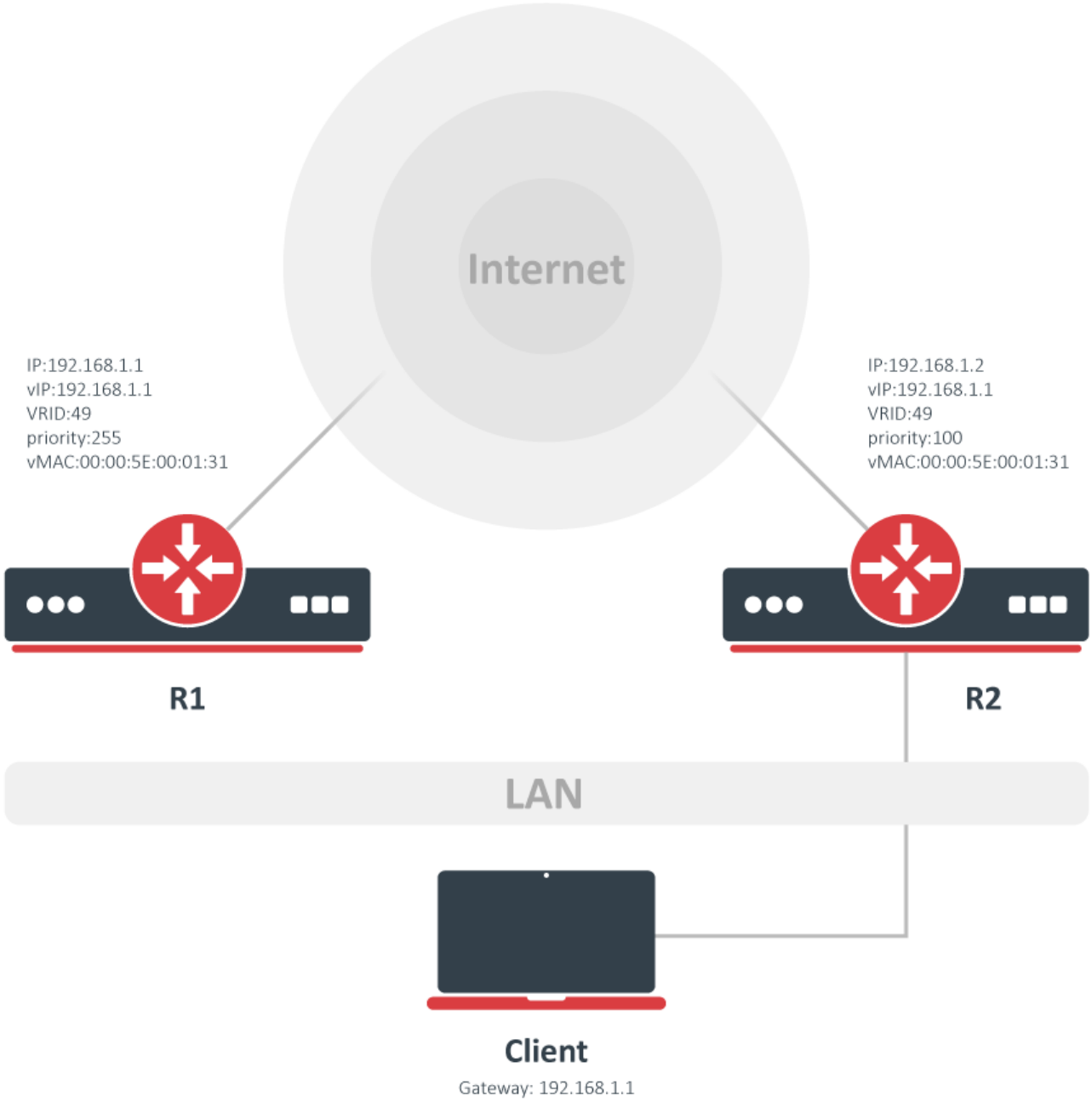
In IPv6 networks, hosts learn about routers by receiving Router Advertisements used by the Neighbor Discovery (ND) protocol. ND already has a built-in mechanism to determine unreachable routers. However, it can take up to 38 seconds to detect an unreachable router. It is possible to change parameters and make detection faster, but it will increase the overhead of ND traffic especially if there are a lot of hosts. VRRP allows the detection of unreachable routers within 3 seconds without additional traffic overhead.

Virtual Router Redundancy Protocol (VRRP) provides a solution by combining a number of routers into a logical group called *Virtual Router* (VR). VRRP implementation in RouterOS is based on VRRPv2 RFC 3768 and VRRPv3 RFC 5798.

It is recommended to use the same version of RouterOS for all devices with the same VRID used to implement VRRP.

According to RFC authentication is deprecated for VRRP v3

Protocol Overview



The purpose of the VRRP is to communicate to all VRRP routers associated with the Virtual Router ID and support router redundancy through a prioritized election process among them.

All messaging is done by IPv4 or IPv6 multicast packets using protocol 112 (VRRP). The destination address of an IPv4 packet is *224.0.0.18* and for IPv6 it is *FF02:0:0:0:0:0:0:12*. The source address of the packet is always the primary IP address of an interface from which the packet is being sent. In IPv6 networks, the source address is the link-local address of an interface.

These packets are always sent with TTL=255 and are not forwarded by the router. If for any reason the router receives a packet with lower TTL, a packet is discarded.

Each VR node has a single assigned MAC address. This MAC address is used as a source for all periodic messages sent by Master.

Virtual Router is defined by VRID and mapped set of IPv4 or IPv6 addresses. The master router is said to be the **owner** of mapped IPv4/IPv6 addresses. There are no limits to using the same VRID for IPv4 and IPv6, however, these will be two different Virtual Routers.

Only the Master router is sending periodic Advertisement messages to minimize the traffic. A backup will try to preempt the Master only if it has the higher priority and preemption is not prohibited.

All VRRP routers belonging to the same VR must be configured with the same advertisement interval. If the interval does not match router will discard the received advertisement packet.

Virtual Router (VR)

A Virtual Router (VR) consists of one Owner router and one or more backup routers belonging to the same network.

VR includes:

- VRID configured on each VRRP router
- the same virtual IP on each router
- Owner and Backup configured on each router. On a given VR there can be only one Owner.

Virtual MAC address

VRRP automatically assigns MAC address to VRRP interface based on standard MAC prefix for VRRP packets and VRID number. The first five octets are 00:00:5E:00:01 and the last octet is configured VRID. For example, if Virtual Routers VRID is 49, then the virtual MAC address will be *00:00:5E:00:01:31*.

Virtual mac addresses can not be manually set or edited.

Owner

An Owner router for a VR is the default Master router and operates as the Owner for all subnets included in the VR. Priority on an owner router must be the highest value (255) and virtual IP is the same as real IP (owns the virtual IP address).

RouterOS can not be configured as Owner. The Pure virtual IP configuration is the only valid configuration unless a non-RouterOS device is set as the owner.

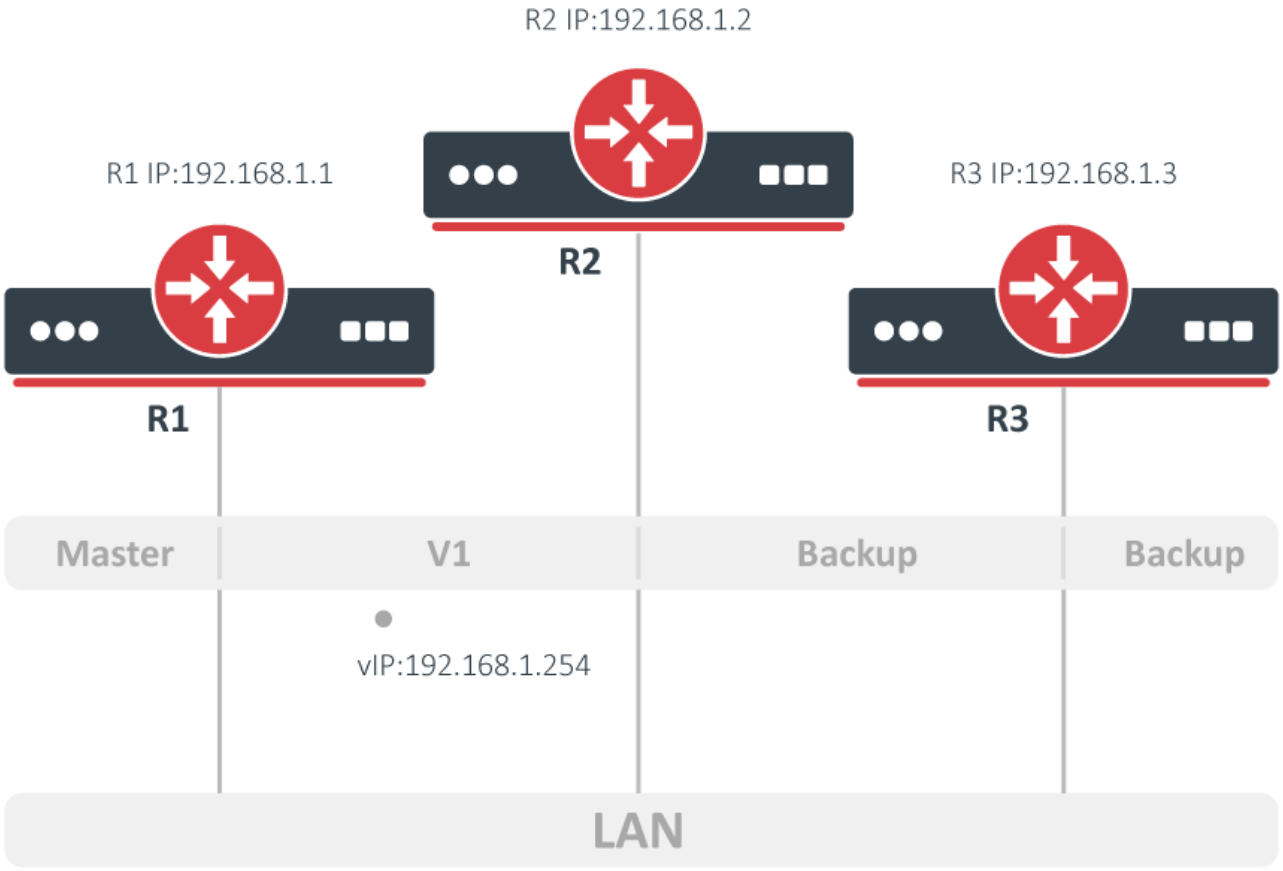
Master

A master router in a VR operates as the physical gateway for the network for which it is configured. The selection of the Master is controlled by priority value. The Master state describes the behavior of the Master router. For example network, **R1** is the Master router. When R1 is no longer available R2 becomes master.

Backup

VR must contain at least one Backup router. A backup router must be configured with the same virtual IP as the Master for that VR. The default priority for Backup routers is 100. When the current master router is no longer available, a backup router with the highest priority will become a current master. Every time when a router with higher priority becomes available it is switched to master. Sometimes this behavior is not necessary. To override it preemption mode should be disabled.

Virtual Address



Virtual IP associated with VR must be identical and set on all VR nodes. All virtual and real addresses should be from the same network.

⚠ RouterOS can not be configured as Owner. VRRP address and real IP address should not be the same.

If the Master of VR is associated with multiple IP addresses, then Backup routers belonging to the same VR must also be associated with the same set of virtual IP addresses. If the virtual address on the Master is not also on Backup a misconfiguration exists and VRRP advertisement packets will be discarded.

All Virtual Router members can be configured so that virtual IP is not the same as physical IP. Such a virtual address can be called a floating or pure virtual IP address. The advantage of this setup is the flexibility given to the administrator. Since the virtual IP address is not the real address of any one of the participant routers, the administrator can change these physical routers or their addresses without any need to reconfigure the virtual router itself.

In IPv6 networks, the first address is always a link-local address associated with VR. If multiple IPv6 addresses are configured, then they are added to the advertisement packet after the link-local address.

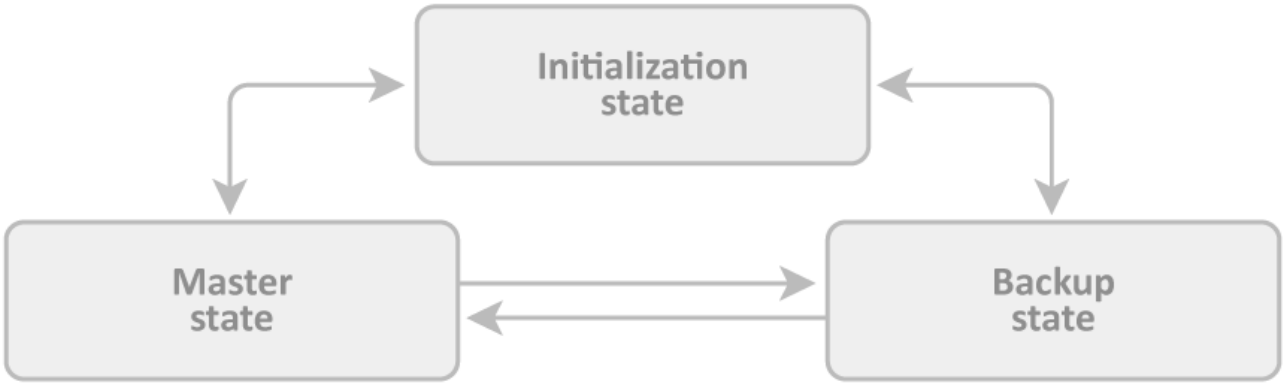
IPv4 ARP

The Master for a given VR responds to ARP requests with the VR's assigned MAC address. The virtual MAC address is also used as the source MAC address for advertisement packets sent by the Master. To ARP requests for non-virtual IP, addresses router responds with the system MAC address. Backup routers are not responding to ARP requests for Virtual IPs.

IPv6 ND

As you may know, in IPv6 networks, the Neighbor Discovery protocol is used instead of ARP. When a router becomes the Master, an unsolicited ND Neighbor Advertisement with the Router Flag is sent for each IPv6 address associated with the virtual router.

VRRP state machine



As you can see from the diagram, each VRRP node can be in one of three states:

- Init state
- Backup state
- Master state

Init state

The purpose of this state is to wait for a Startup event. When this event is received, the following actions are taken:

- if priority is 255,
- * for IPv4 send advertisement packet and broadcast ARP requests
- * for IPv6 send an unsolicited ND Neighbor Advertisement for each IPv6 address associated with the virtual router and set target address to link-local address associated with VR.
- * transit to MASTER state;
- else transit to BACKUP state.

Backup state

When in the backup state,

- in IPv4 networks, a node is not responding to ARP requests and is not forwarding traffic for the IP associated with the VR.
- in IPv6 networks, a node is not responding to ND Neighbor Solicitation messages and is not sending ND Router Advertisement messages for VR-associated IPv6 addresses.

Routers' main task is to receive advertisement packets and check if the master node is available.

The backup router will transmit itself to the master state in two cases:

- If priority in advertisement packet is 0;
- When Preemption_Mode is set to yes and Priority in the ADVERTISEMENT is lower than the local Priority

After the transition to Master state node is:

- in IPv4 broadcasts gratuitous ARP request;
- in IPv6 sends an unsolicited ND Neighbor Advertisement for every associated IPv6 address.

In other cases, advertisement packets will be discarded. When the shutdown event is received, transit to Init state.

> ⚠ Preemption mode is ignored if the Owner router becomes available.

Master state

When the MASTER state is set, the node functions as a forwarding router for IPv4/IPv6 addresses associated with the VR.

In IPv4 networks, the Master node responds to ARP requests for the IPv4 address associated with the VR. In IPv6 networks Master node:

- responds to ND Neighbor Solicitation message for the associated IPv6 address;

- sends ND Router Advertisements for the associated IPv6 addresses.

If the advertisement packet is received by master node:

- If priority is 0, send advertisement immediately;
- If priority in advertisement packet is greater than nodes priority then transit to the backup state
- If priority in advertisement packet is equal to nodes priority and primary IP Address of the sender is greater than the local primary IP Address, then transit to the backup state
- Ignore advertisement in other cases

When the shutdown event is received, send the advertisement packet with priority=0 and transit to Init state.

Connection tracking synchronization

Similar to different High availability features, RouterOS v7 supports VRRP connection tracking synchronization.

The VRRP connection tracking synchronization requires that RouterOS connection tracking is running. By default, connection tracking is working in `auto` mode. If VRRP devices do not contain any firewall rules, you need to manually enable connection tracking:

```
/ip/firewall/connection/tracking/set enabled=yes
```

To sync connection tracking entries configure the device as follows:

```
/interface/vrrp/set vrrp1 sync-connection-tracking=yes
```

Verify configuration in the logging section:

```
16:14:06 vrrp,info vrrp1 now MASTER, master down timer
16:14:06 vrrp,info vrrp1 stop CONNTRACK
16:14:06 vrrp,info vrrp1 starting CONNTRACK MASTER
```

Connection tracking entries are synchronized only from the Master to the Backup device.

When both **`sync-connection-tracking`** and **`preemption-mode`** are enabled, and a router with higher VRRP priority becomes online, the connections get synchronized first, and only then the router with higher priority becomes the VRRP master.

If multiple VRRP interfaces are configured between two units, then it is enough to enable sync-connection-tracking=yes on one (preferably master) VRRP interface.

Configuring VRRP

IPv4

Setting up Virtual Router is quite easy, only two actions are required - create VRRP interface and set Virtual Routers IP address.

For example, add VRRP to ether1 and set VRs address to 192.168.1.1

```
/interface vrrp add name=vrrp1 interface=ether1
/ip address add address=192.168.1.2/24 interface=ether1
/ip address add address=192.168.1.1/32 interface=vrrp1
```

Notice that only the 'interface' parameter was specified when adding VRRP. It is the only parameter required to be set manually, other parameters if not specified will be set to their defaults: `vrid=1, priority=100` and `authentication=none`.

⚠ Address on the VRRP interface must have /32 netmask if the address configured on VRRP is from the same subnet as on the router's any other interface.

Before VRRP can operate correctly correct IP address is required on ether1. In this example, it is 192.168.1.2/24

IPV6

To make VRRP work in IPv6 networks, several additional options must be enabled - v3 support is required and the protocol type should be set to IPv6:

```
/interface vrrp add name=vrrp1 interface=ether1 version=3 v3-protocol=ipv6
```

Now when the VRRP interface is set, we can add a global address and enable ND advertisement:

```
/ipv6 address add address=FEC0:0:0:FFFF::1/64 advertise=yes interface=vrrp1
```

No additional address configuration is required as it is in the IPv4 case. IPv6 uses link-local addresses to communicate between nodes.

Parameters

Property	Description
arp (*disabled | enabled | proxy-arp | reply-only*; Default: **enabled**)	ARP resolution protocol mode
arp-timeout *(integer; Default: auto)*	
authentication (*ah | none | simple*; Default: **none**)	Authentication method to use for VRRP advertisement packets. • none - should be used only in low-security networks (e.g., two VRRP nodes on LAN). • ah - IP Authentication Header. This algorithm provides strong protection against configuration errors, replay attacks, and packet corruption/modification. Recommended when there is limited control over the administration of nodes on a LAN. HMAC-MD5 is used. • simple - uses a clear-text password. Protects against accidental misconfiguration of routers on a local network.

group-authority (*interface;* Default: **none**)	Allows combining multiple VRRP interfaces to maintain the same VRRP status within the group. For example, VRRP instances run on LAN and WAN networks with NAT in-between. If one VRRP instance is Master and the other is Backup on the same device, the entire network malfunctions due to NAT failure. Grouping LAN and WAN VRRP interfaces ensures that both are either VRRP Master or Backup.
	In a VRRP group, VRRP control traffic gets sent only by the group authority. That's why in a typical WAN+LAN setup, it is recommended to use the LAN network as the group master to keep VRRP control traffic in the internal network.
	`/interface vrrp`
	`add name=vrrp-wan interface=sfp-sfpplus1 vrid=1 priority=100`
	`add name=vrrp-lan interface=bridge1 vrid=2 priority=100`
	`set [find] group-authority=vrrp-lan`
	Group-authority was previously called "group-master", "group-master" is kept for compatibility with scripts, but if both are set only "group-authority" will be taken into account.

interface (*string*; Default:)	Interface name on which VRRP instance will be running

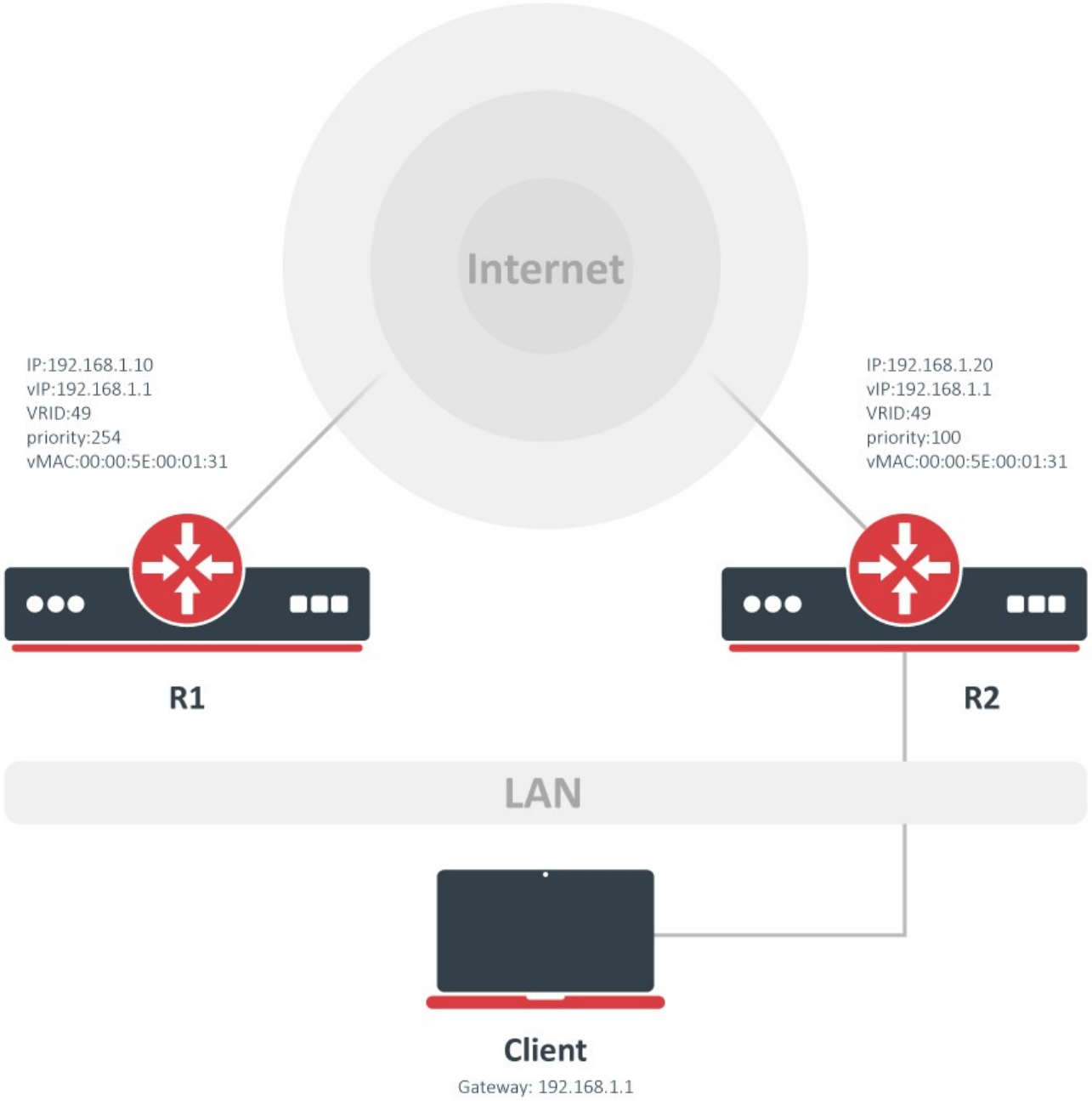
interval (*time [10ms..4m15s]*; Default: **1s**)	VRRP update interval in seconds. Defines how often the master sends advertisement packets.
mtu (*integer*; Default: **1500**)	Layer3 MTU size. Since RouterOS v7.7, the VRRP interface always uses slave interface MTU
name (*string*; Default:)	VRRP interface name
on-backup (*string*; Default:)	Script to execute when the node is switched to the backup state
on-master (*string*; Default:)	Script to execute when the node is switched to master state
on-fail (*string*; Default:)	Script to execute when the node fails
password (*string*; Default:)	Password required for authentication. Can be ignored if authentication is not used.
preemption-mode (*yes* | *no*; Default: **yes**)	Whether the master node always has the priority. When set to 'no' the backup node will not be elected to be a master until the current master fails, even if the backup node has higher priority than the current master. This setting is ignored if the owner router becomes available
priority (*integer: 1..254*; Default: **100**)	Priority of VRRP node used in Master election algorithm. A higher number means higher priority. '255' is reserved for the router that owns VR IP and '0' is reserved for the Master router to indicate that it is releasing responsibility.
remote-address (*IPv4;* Default:)	Specifies the remote address of the other VRRP router for syncing connection tracking. If not set, the system autodetects the remote address via VRRP. The remote address is used only if sync-connection-tracking=yes. Explicitly setting a remote address has the following benefits: • Connection syncing starts faster since there is no need to wait for VRRP's initial message exchange to detect the remote address. • Faster VRRP Master election. • Allows sending connection tracking data via a different network interface (e.g., a dedicated secure line between two routers). Sync connection tracking uses UDP port 8275.
v3-protocol (*ipv4 | ipv6*; Default: **ipv4**)	A protocol that will be used by VRRPv3. Valid only if the **version** is 3.
version (*integer [2, 3]*; Default: **3**)	Which VRRP version to use.
vrid (*integer: 1..255*; Default: **1**)	Virtual Router identifier. Each Virtual router must have a unique id number
sync-connection-tracking (*string*; Default: **no**)	Synchronize connection tracking entries from Master to Backup device. The VRRP connection tracking synchronization requires that RouterOS connection tracking is running.

VRRP Configuration Examples

Basic Setup

This is the basic VRRP configuration example.

> **Note**
>
> It is recommended to use the same version of RouterOS for all devices with the same VRID used to implement VRRP.



According to this configuration, as long as the master, R1, is functional, all traffic destined to the external network gets directed to R1. But as soon as R1 fails, R2 takes over as the master and starts handling packets forwarded to the interface associated with IP(R1). In this setup router "R2" is completely idle during the Backup period.

Configuration

R1 configuration:

```
/ip address add address=192.168.1.10/24 interface=ether1
/interface vrrp add interface=ether1 vrid=49 priority=254
/ip address add address=192.168.1.1/32 interface=vrrp1
```

R2 configuration:

```
/ip address add address=192.168.1.20/24 interface=ether1
/interface vrrp add interface=ether1 vrid=49
/ip address add address=192.168.1.1/32 interface=vrrp1
```

Testing

First of all, check if both routers have correct flags at VRRP interfaces. On router R1 it should look like this

```
/interface vrrp print detail
 0   RM name="vrrp1" mtu=1500 mac-address=00:00:5E:00:01:31 arp=enabled interface=ether1 vrid=49
       priority=254 interval=1 preemption-mode=yes authentication=none password="" on-backup=""
       on-master="" version=3 v3-protocol=ipv4
```

and on router R2:

```
/interface vrrp print detail
 0    B name="vrrp1" mtu=1500 mac-address=00:00:5E:00:01:31 arp=enabled interface=ether1 vrid=49
       priority=100 interval=1 preemption-mode=yes authentication=none password=""
       on-backup="" on-master=" version=3 v3-protocol=ipv4
```

As you can see VRRP interface MAC addresses are identical on both routers. Now to check if VRRP is working correctly, try to ping the virtual address from a client and check ARP entries:

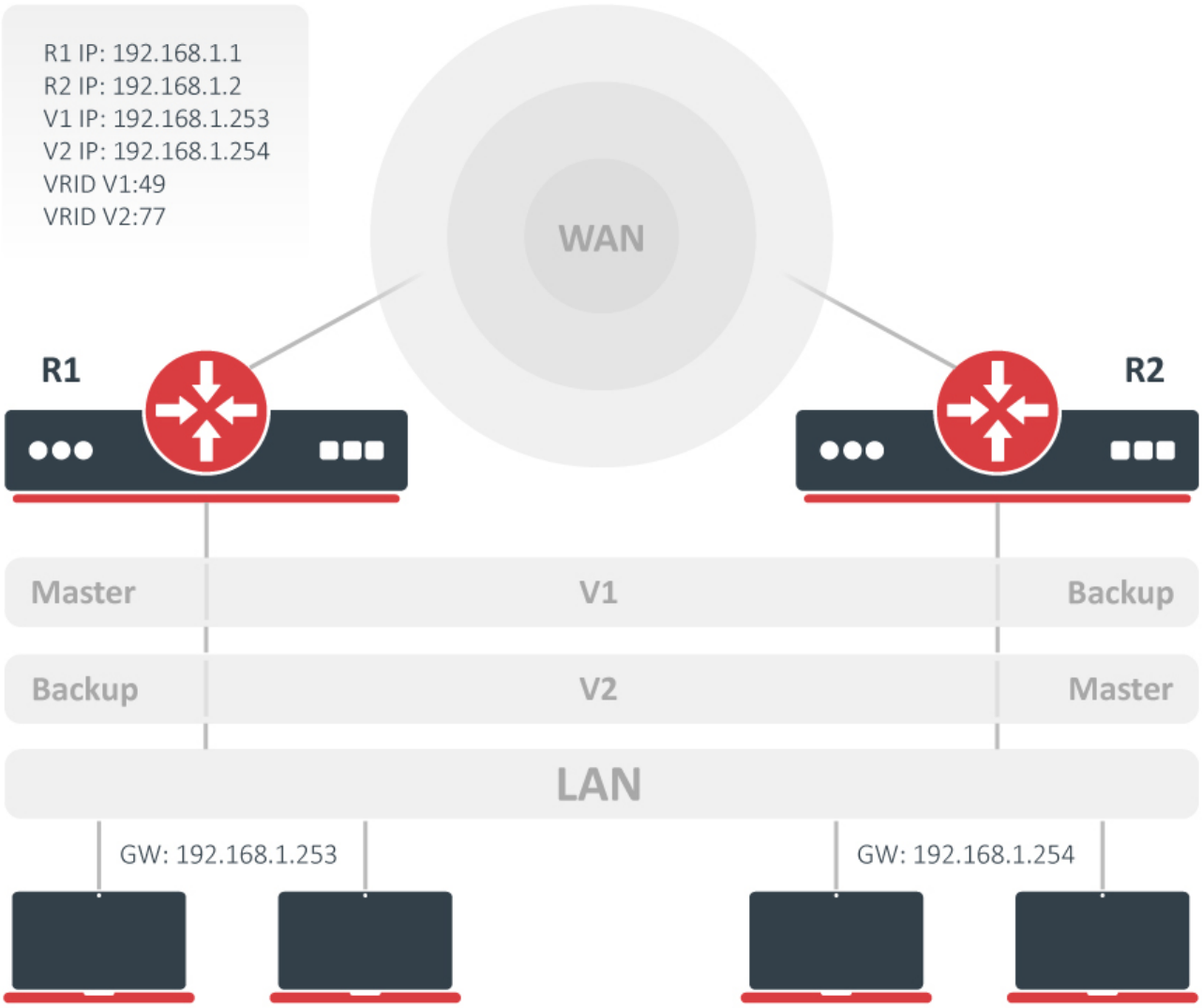
```
[admin@client] > /ping 192.168.1.1
192.168.1.254 64 byte ping: ttl=64 time=10 ms
192.168.1.254 64 byte ping: ttl=64 time=8 ms
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 8/9.0/10 ms
[admin@client] /ip arp> print
Flags: X - disabled, I - invalid, H - DHCP, D - dynamic
 #   ADDRESS         MAC-ADDRESS       INTERFACE
 ...
 1 D 192.168.1.1   00:00:5E:00:01:31 bridge1
```

Now unplug the ether1 cable on router R1. R2 will become VRRP master, and the ARP table on a client will not change but traffic will start to flow over the R2 router.

> In case VRRP is used with Reverse Path Filtering, then it is recommended that `rp-filter` is set to `loose`, otherwise, the VRRP interface might not be reachable.

Load sharing

In the basic configuration example, R2 is completely idle during the Backup state. This behavior may be considered a waste of valuable resources. In such circumstances, the R2 router can be set as the gateway for some clients.
The obvious advantage of this configuration is the establishment of a load-sharing scheme. But by doing so R2 router is not protected by the current VRRP setup.
To make this setup work we need two virtual routers.



Configuration for V1 virtual router will be identical to a configuration in basic example - R1 is the Master and R2 is the Backup router. In V2 Master is R2 and Backup is R1.
With this configuration, we establish load-sharing between R1 and R2; moreover, we create a protection setup by having two routers acting as backups for each other.

Configuration

R1 configuration:

```
/ip address add address=192.168.1.1/24 interface=ether1
/interface vrrp add interface=ether1 vrid=49 priority=254
/interface vrrp add interface=ether1 vrid=77
/ip address add address=192.168.1.253/32 interface=vrrp1
/ip address add address=192.168.1.254/32 interface=vrrp2
```

R2 configuration:

```
/ip address add address=192.168.1.2/24 interface=ether1
/interface vrrp add interface=ether1 vrid=49
/interface vrrp add interface=ether1 vrid=77 priority=254
/ip address add address=192.168.1.253/32 interface=vrrp1
/ip address add address=192.168.1.254/32 interface=vrrp2
```

VRRP without Preemption

Each time when the router with a higher priority becomes available it becomes the Master router. Sometimes this is not the desired behavior and can be turned off by setting `preemption-mode=no` in VRRP configuration.

Configuration

We will be using the same setup as in the basic example. The only difference is during configuration set preemption-mode=no. It can be done easily by modifying the existing configuration:

```
/interface vrrp set [find] preemption-mode=no
```

Testing

Try turning off the R1 router, R2 will become the Master router because it has the highest priority among available routers.

Now turn the R1 router on and you will see that the R2 router continues to be the Master even if R1 has the higher priority.

Mobile Networking

In This Section:

GPS

Summary

Package requirement: `gps`
Sub-menu: `/system gps`
Standards: `GPS, NMEA 0183, Simple Text Output Protocol`

Global Positioning System (GPS) is used for determining precise location of a GPS receiver.

Configuration Properties

Property	Description
channel (*integer [0..4294967295]*; Default: **0**)	Port channel used by the device
coordinate-format (*dd* / *dms* / *ddmm*; Default: **no**)	Which coordinate format to use, "Decimal Degrees", "Degrees Minutes Seconds" or "NMEA format DDDMM.MM[MM]"
enabled (*yes* / *no*; Default: **no**)	Whether GPS is enabled
gps-antenna-select (*external* / *internal*; Default: **internal**)	Depending on the model. Internal antenna can be selected, if the device has one installed.
init-channel (*integer [0..4294967295]*; Default:)	Channel for init-string execution
init-string (*string*; Default:)	AT init string for GPS initialization
port (*string*; Default:)	Name of the USB/Serial port where GPS receiver is connected
set-system-time (*yes* / *no*; Default: **no**)	Whether to set router's date and time to one received from GPS.

Monitoring Status

Command: `/system gps monitor`

This command is used for monitoring the data received from a GPS receiver.

Parameters:

⚠ Starting with the 7.1rc3 firmware release, a new parameter was added, called "data-age" (measured in seconds). This parameter displays the time that has passed since the device received last NMEA message.

Property	Description
date-and-time (*date*)	Date and time received from GPS
latitude (*none* / *string*)	Latitude in DM (Degrees Minute decimal) format
longitude (*none* / *string*)	Longitude in DM (Degrees Minute decimal) format
altitude (*none* / *string*)	Altitude based on GPS data
speed (*none* / *string*)	Current moving speed of the GPS unit
destination-bearing (*none* / *string*)	The direction toward which a GPS is moving

true-bearing (*none* | *string*)	The direction toward which a GPS is moving
magnetic-bearing (*none* | *string*)	The direction toward which a GPS is moving
valid (*yes* | *no*)	
satellites (*integer*)	Number of satellites seen by the device.
fix-quality (*integer*)	Quality of the signal
horizontal-dilution (*integer*)	Horizontal dilution of precision (HDOP);
data-age (integer)	The time that has passed since the device received last NMEA message

Basic examples

Check port usage, as only one instance can use serial port simultaneously:

```
[admin@MikroTik] /port print
Flags: I - inactive
 #   DEVICE NAME                    CHANNELS USED-BY                  BAUD-RATE
 0          serial0                        1 Serial Console            auto
```

In case there is one port and it is used by console, release it from the console menu:

```
[admin@MikroTik] > /system console print
Flags: X - disabled, U - used, F - free
 #   PORT
TERM
 0 U serial0                                                                vt102

[admin@MikroTik] > /system console disable 0
```

Adjust port settings specifically for your device (leave "auto" for LtAP mini):

```
[admin@MikroTik] /port> set 0 baud-rate=4800 parity=odd
[admin@MikroTik] /port> print detail
Flags: I - inactive
 0   name="usb1" used-by="" channels=1 baud-rate=4800 data-bits=8 parity=odd stop-bits=1 flow-control=none
```

Enable GPS:

```
[admin@MikroTik] /system gps> set enable=yes port=usb1
[admin@MikroTik] /system gps> print
          enabled: yes
             port: usb1
          channel: 0
     init-channel: 0
      init-string:
  set-system-time: no
```

Monitor status:

```
[admin@MikroTik] /system gps> monitor
        date-and-time: sep/07/2021 08:26:26
             latitude: 56.969689
            longitude: 24.162471
             altitude: 25.799999m
                speed: 0.759320 km/h
  destination-bearing: none
        true-bearing: 185.500000 deg. True
    magnetic-bearing: 0.000000 deg. Mag
                valid: yes
           satellites: 6
         fix-quality: 1
  horizontal-dilution: 1.3
```

Port and GPS settings for **LtAP**

```
/port set serial1 baud-rate=115200
```

```
/system gps set port=serial1 channel=0 enabled=yes
```

We have also created an in-depth article about live GPS tracking, using scripting and a web server: Manual:GPS-tracking.

Troubleshooting

Note that sometimes in order to make GPS module to be recognized under RouterOS you need to change the baud-rate setting in the '/port' menu.

LtAP mini has a low gain GPS antenna built-in and for a better experience, we suggest using an additional external antenna.

Switch between internal and external antennas under GPS menu:

```
[admin@MikroTik] > /system gps set gps-antenna-select=external
```

On some modems with GPS support you need to send multiple init commands for the continuous GPS monitoring, for example, for Huawei cards you need to send "AT^WPDST=1,AT^WPDGP" init string to get continuous monitoring.

GPS-tracking using HTTP POST

The following article explains how to create a simple vehicle tracking system using the RouterOS GPS function and scripting.

Method

This approach uses HTTP POST capability of RouterOS Fetch tool. It allows you to POST any kind of data to a webserver, right from RouterOS command line. Of course, you can use scripting, to fill the POST data with variables. The posted data will be written to an SQLITE3 database (file is created, if it doesn't exist) and then, read from the database and fead into a Leaflet.js PolyLine array. This is a proof of concept example, there is no authentication, security or error handling.

Requirements

- Webserver of your choice
- PHP
- SQLite3 module for PHP
- RouterOS device with a working GPS module
- RouterOS v6.40rc30 or above
- Set GPS format in RouterOS to **dd**

RouterOS script

You can run this script in the Scheduler tool, with an interval of 1s, to have your coordinates sent every 1 seconds.

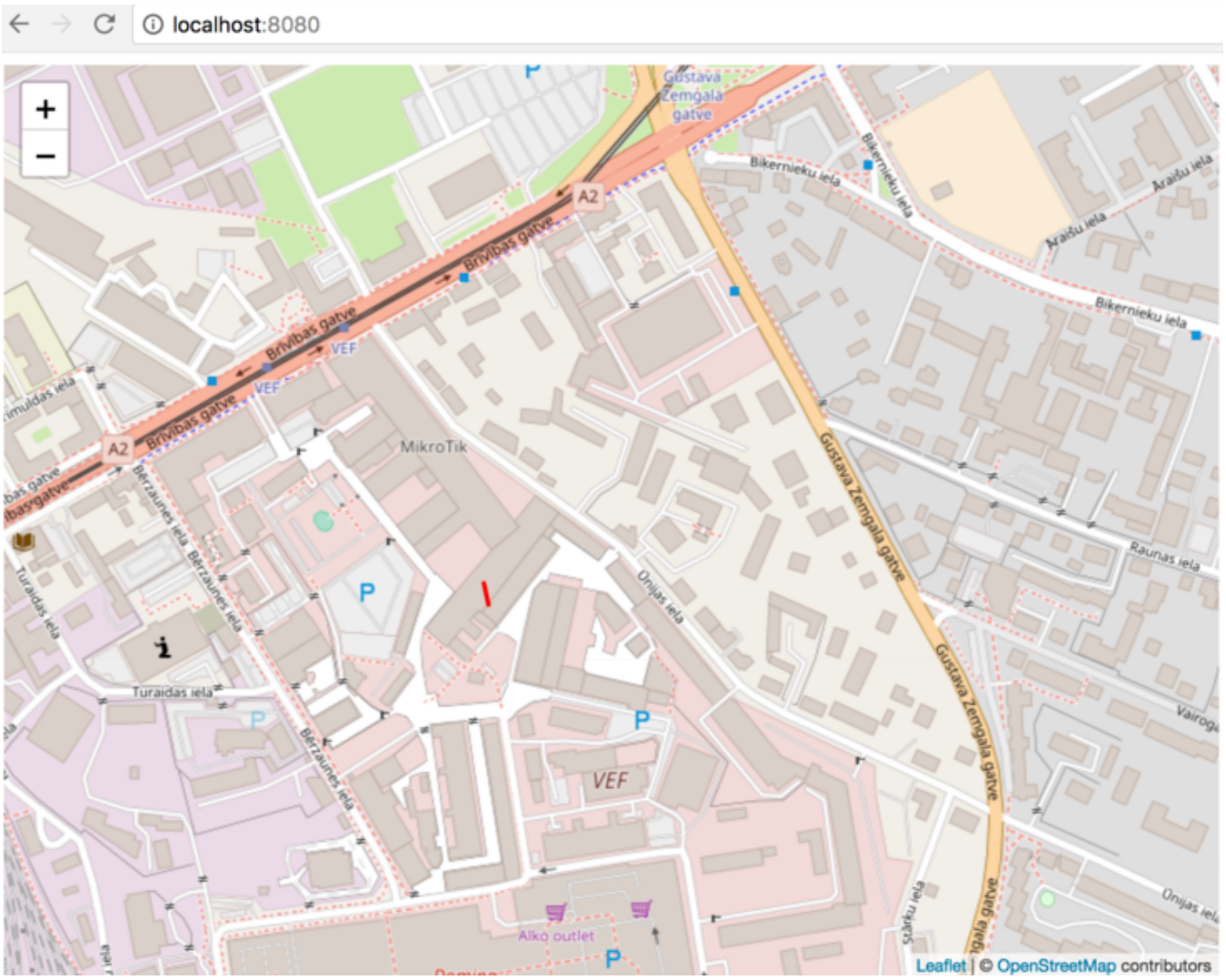
```
{
:global lat
:global lon
/system gps monitor once do={
:set $lat $("latitude")
:set $lon $("longitude")
}
tool fetch mode=http url="http://YOURSERVER.com/index.php" port=80 http-method=post http-data=("{\"lat\":\"" .
$lat . "\",\"lon\":\"" . $lon . "\"}") http-header-field="Content-Type: application/json"
:put ("{\"lat\":\"" . $lat . "\",\"lon\":\"" . $lon . "\"}")
}
```

index.php file

Create an empty directory called **sqlite_db** next to the index.php file. Make sure that directory and files are writable by the group with **chmod -R a+w sqlite_db/**

```
<?php
$loc = dirname(__FILE__).'/sqlite_db/coord.db';
$db = new SQLite3($loc,SQLITE3_OPEN_READWRITE | SQLITE3_OPEN_CREATE);
$raw = file_get_contents('php://input');
$raw = preg_replace('/\\x00/','',$raw);
$data = json_decode($raw);

if (!empty($data) && is_object($data) && property_exists($data,'lat') && property_exists($data,'lon')){
    if(file_exists($loc)) echo 'exists!'.chr(0xa);
    $src = 'SELECT name FROM sqlite_master WHERE type=\'table\' AND name=\'coordinates\'';
    $res = $db->querySingle($src);
    if (count($res)==0){
            $db->exec('CREATE TABLE coordinates (latitude TEXT, longitude TEXT, time TIMESTAMP DEFAULT
CURRENT_TIMESTAMP, added TIMESTAMP DEFAULT CURRENT_TIMESTAMP ) ');
    }

$regex = '/^(|\-)([0-9]{2,3}\.[0-9]{0,8})$/';

if (preg_match($regex,$data->lat) && preg_match($regex,$data->lon) )
        {
                $lat = $data->lat;
                $lon = $data->lon;
        }
        $ins = 'INSERT INTO coordinates (latitude,longitude) VALUES (\''.SQLite3::escapeString($lat).'\',\''.
SQLite3::escapeString($lon).'\')';
        $db->exec($ins);
        die();
}
?>

<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.1/dist/leaflet.css" integrity="sha512-
Rksm5RenBEKSKFjgI3a41vrjkw4EVPlJ3+OiI65vTjIdo9brlAacEuKOiQ5OFh7cOI1bkDwLqdLw3Zg0cRJAAQ==" crossorigin=""/>
  <script src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js" integrity="sha512-
/Nsx9X4HebavoBvEBuyp3I7od5tA0UzAxs+j83KgC8PU0kgB4XiK4Lfe4y4cgBtaRJQEIFCW+oC506aPT2L1zw==" crossorigin=""><
/script>
</head>
<body>
<div id="map" style="width: 800px; height: 600px;"></div>
<script>
var map = L.map('map').setView([0,0], 4);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {attribution: '<a href="http://osm.org/copyright">OSM<
/a>'}).addTo(map);

<?php
    if($result = $db->query('SELECT latitude,longitude FROM coordinates')){
    echo ' var latlngs = [ ';
    while($obj = $result->fetchArray()){
            if (!is_array($obj) || !isset($obj['latitude']) || !isset($obj['longitude']) || empty($obj
['latitude']) || empty($obj['longitude'])) continue;
            echo '["'. $obj['latitude'].'","'.$obj['longitude'].'"],';
    }
    echo ']; ';
    } else
     echo('//'.$db->lastErrorMsg().chr(0xa));
         echo($data);
?>
var polyline = L.polyline(latlngs, {color: 'red'}).addTo(map);
map.fitBounds(polyline.getBounds());
</script>
</body>
</html>
```

Result


localhost:8080
Gustava Zemgala gatve
Bikernieku iela
Araišu iela
A2
Brīvības gatve
VEF
MikroTik
Gustava Zemgala gatve
Raunas iela
Berzaunes iela
Turaidas iela
Onijas iela
Alko outlet
Leaflet | © OpenStreetMap contributors

GPS-tracking using MQTT and ThingsBoard

Introduction

Many RouterOS devices have GPS support. It allows RouterOS to determine the precise location of its GPS receiver. GPS coordinates will indicate the latitude and the longitude values (among other parameters) of the current position.

Let's say, you have LTAP (or any other RouterOS device with GPS support) and you wish to track its location. You want the router to send this data to a server, where the data will be stored and integrated into a map, as it is more convenient to monitor. In this guide, we will showcase how you can do that. This scenario will utilize MQTT protocol communication with a platform called ThingsBoard.

ThingsBoard has a cloud solution and different local installation options (on different OS).

Since we've added a container feature, it became possible to also run the platform within the RouterOS. Meaning, you can build this scenario, solely on RouterOS units → devices with GPS support that you wish to track (for example, cars equipped with LTAPs → RouterOS devices that act as **MQTT publishers**), and a ThingsBoard server run within a more powerful RouterOS device (like a CHR machine → RouterOS device that acts as an **MQTT broker**).

If you want to choose this route (container route), make sure to pick the devices that you plan on using as a "server" carefully, because this implementation can be heavy on RAM usage (it is suggested to have a device that has at least **2 GB RAM** or **1 GB RAM with minimal load** and is either **ARM64** or **AMD64** architecture).

Configuration

In this guide, we will demonstrate how to configure a GPS receiver (MQTT publisher) and how to set up ThingsBoard.

In case you want to use the container feature to run the ThingsBoard instance (MQTT broker), check the guide over here. General guidelines on ThingsBoard and MQTT configuration can be found in the guide over here. Make sure to explore both guides as they will have additional useful information.

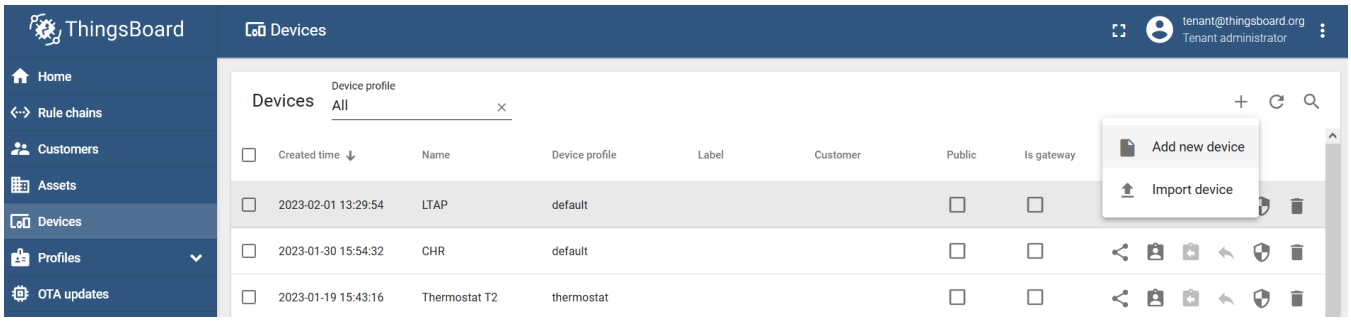
Before proceeding, make sure that the ThingsBoard is up and running and that you are able to access its WEB management portal. Confirm that the MQTT port is open or/and port-forwarded properly.

> **Package requirement:** `gps, iot`

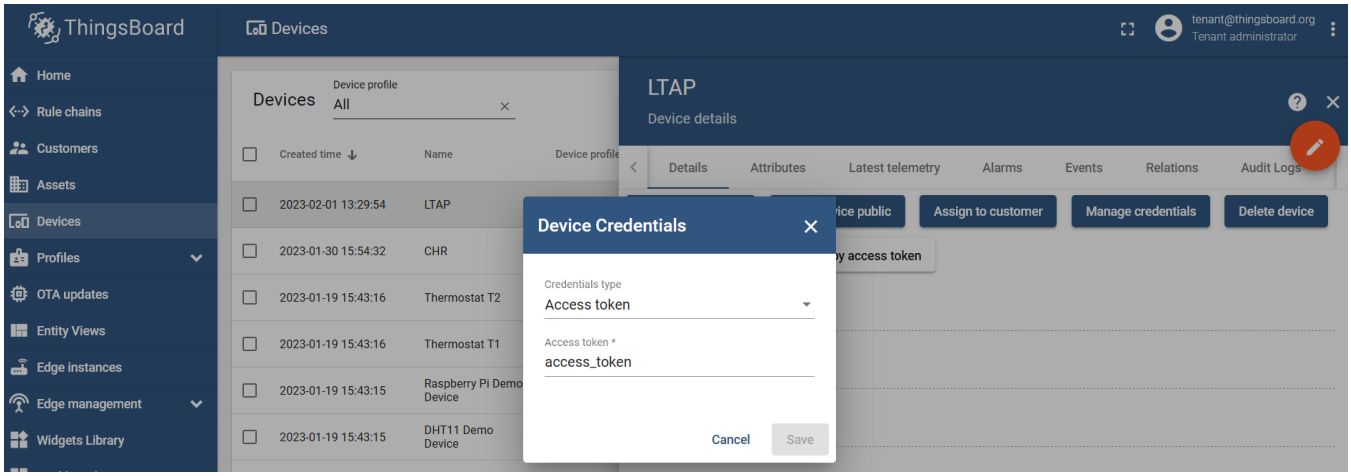
ThingsBoard preparation

> This example will showcase access-token and one-way SSL communication via access-token scenarios for simplicity reasons, but you can use other available options as well.

Navigate to the "**Devices**" menu and add a new device via the "**Add new device**" button → name it and create it (for example, LTAP):



Click on the device you've just added, go to the "**Details**" section, and generate an access token under the "**Manage credentials/Device Credentials**" setting:



MQTT broker configuration

In case it is a local test or the broker is available through the VPN, you can use non-SSL MQTT:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=1883 username=access_token
```

Where:

- `name` is the name that you wish to give to the broker and this name will be used later in the script;
- `address` is the IP address of the broker;
- `port` is the TCP port that the broker is listening for → for non-SSL it is typically TCP 1883;
- `username` is dictated by the MQTT broker and, in our case, it is an "access token" that was generated in the ThingsBoard management portal.

In case it is public access (when you want to access the broker via its public IP address), **we advise you to use SSL MQTT**:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=8883 username=access_token ssl=yes
```

Where:

- `name` is the name that you wish to give to the broker and this name will be used later in the script;
- `address` is the IP address of the broker;
- `port` is the TCP port that the broker is listening for → for SSL it is typically TCP 8883;
- `username` is dictated by the MQTT broker, and, in our case, it is an "access token" that was generated in the ThingsBoard management portal;
- `ssl` enables SSL MQTT communication.

MQTT publish

You can test MQTT publish with a static message by using the command:

```
/iot/mqtt/publish broker="tb" topic="v1/devices/me/telemetry" message="{\"test\":\"123\"}"
```

To post GPS coordinates, import the script shown below:

```
/system/script/add dont-require-permissions=no name=mqttgps owner=admin policy="ftp,re\
    boot,read,write,policy,test,password,sniff,sensitive,romon" \
    source="    ###Configuration###\r\
    \n    #Enter pre-configured broker's name within \"\":\r\
    \n    :local broker \"tb\"\r\
    \n    #Enter the topic name within \"\", per the broker's config\
    uration:\r\
    \n    :local topic \"v1/devices/me/telemetry\"\r\
    \n\r\
    \n    ###Variables####\r\
    \n    :global lat\r\
    \n    :global lon\r\
    \n    :global alt1\r\
    \n    :global alt2\r\
    \n\r\
    \n    ###GPS####\r\
    \n    :put (\"[*] Capturing GPS coordinates...\")\r\
    \n    /system gps monitor once do={\r\
    \n    :set \$lat \$(\"latitude\");\r\
    \n    :set \$lon \$(\"longitude\");\r\
    \n    :set \$alt1 \$(\"altitude\")}\r\
    \n    ###remove \"meters\" from the value because JSON format wi\
    ll not understand it###\r\
    \n    :set \$alt2 [:pick \$alt1 0 [find \$alt1 \" m\"]]\r\
    \n\r\
    \n    :local message \\\r\
    \n    \"{\\\"latitude\\\":\$lat,\\\r\
    \n    \\\"longitude\\\":\$lon,\\\r\
    \n    \\\"altitude\\\":\$alt2}\"\r\
    \n\r\
    \n    ###MQTT###\r\
    \n    :if (\$lat != \"none\") do={\\\r\
    \n    :put (\"[*] Sending message to MQTT broker...\");\r\
    \n    /iot mqtt publish broker=\$broker topic=\$topic message=\$\
    message} else={:put (\"[*] Lattitude=none, not posting anything!\
    \");:log info \"Latitude=none, not posting anything!\"}"
```

In short, the script captures GPS information, specifically the latitude, longitude, and altitude values. Then it structures a JSON message out of them. In case, at the moment when the script is initiated, the latitude value equals anything other than "none" (equals any actual value-number) → it sends the JSON message via MQTT to the broker named "**tb**". In case, the GPS data can not be captured → "latitude" is recognized as "none" → the script just logs that nothing could be captured and does nothing else.

This is a very basic example. Feel free to alter the script and add your own "if" (maybe an email notification if there is no GPS signal) and additional parameters (any other RouterOS captured value, like, maybe, its firmware version) per your requirements.

Run the script with the command:

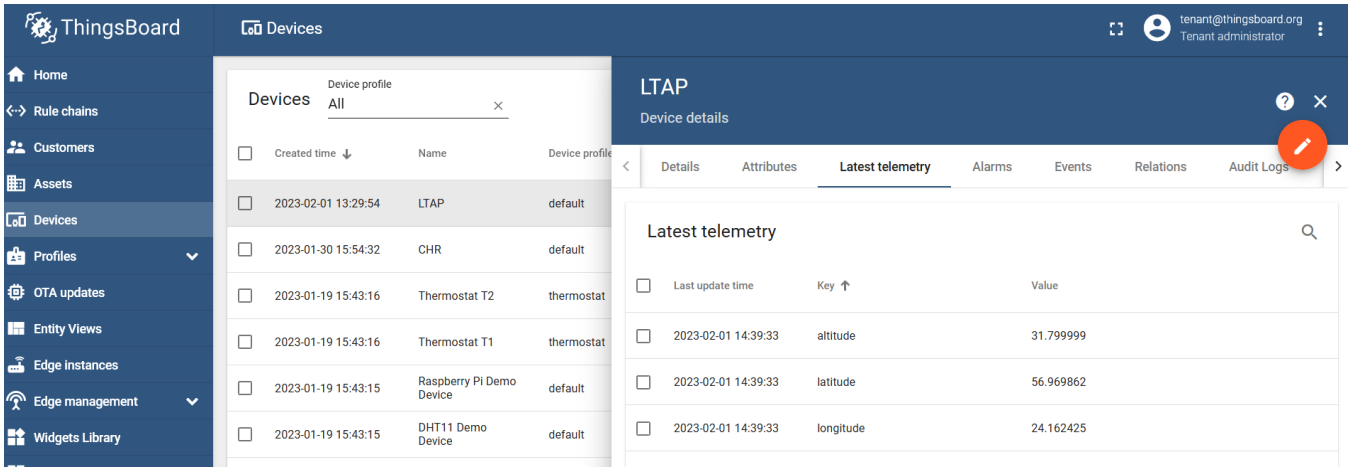
```
/system/script/run mqttgps
[*] Capturing GPS coordinates...
        date-and-time: feb/01/2023 10:39:37
             latitude: 56.969862
            longitude: 24.162425
             altitude: 31.799999 m
                speed: 1.000080 km/h
  destination-bearing: none
         true-bearing: 153.089996 deg. True
    magnetic-bearing: 0.000000 deg. Mag
                valid: yes
           satellites: 6
          fix-quality: 1
  horizontal-dilution: 1.42
             data-age: 0s
[*] Sending message to MQTT broker...
```

To automate the process, add a scheduler (to run the script, for example, every 30 seconds):

```
/system/scheduler/add name=mqttgpsscheduler interval=30s on-event="/system/script/run mqttgps"
```

Result verification

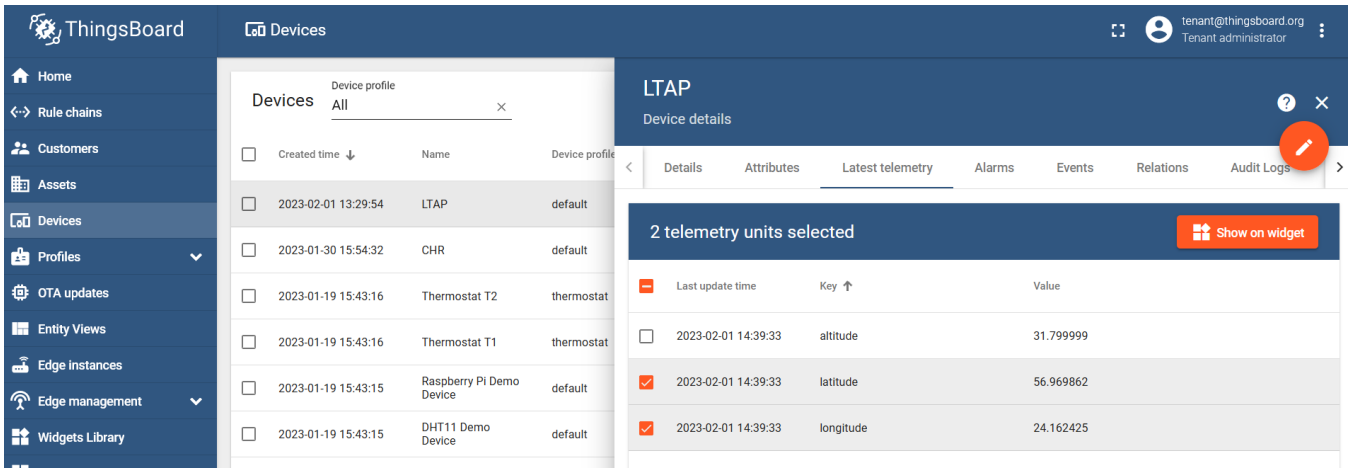
Go the the "Latest telemetry" section under your created device and confirm that the data was posted:



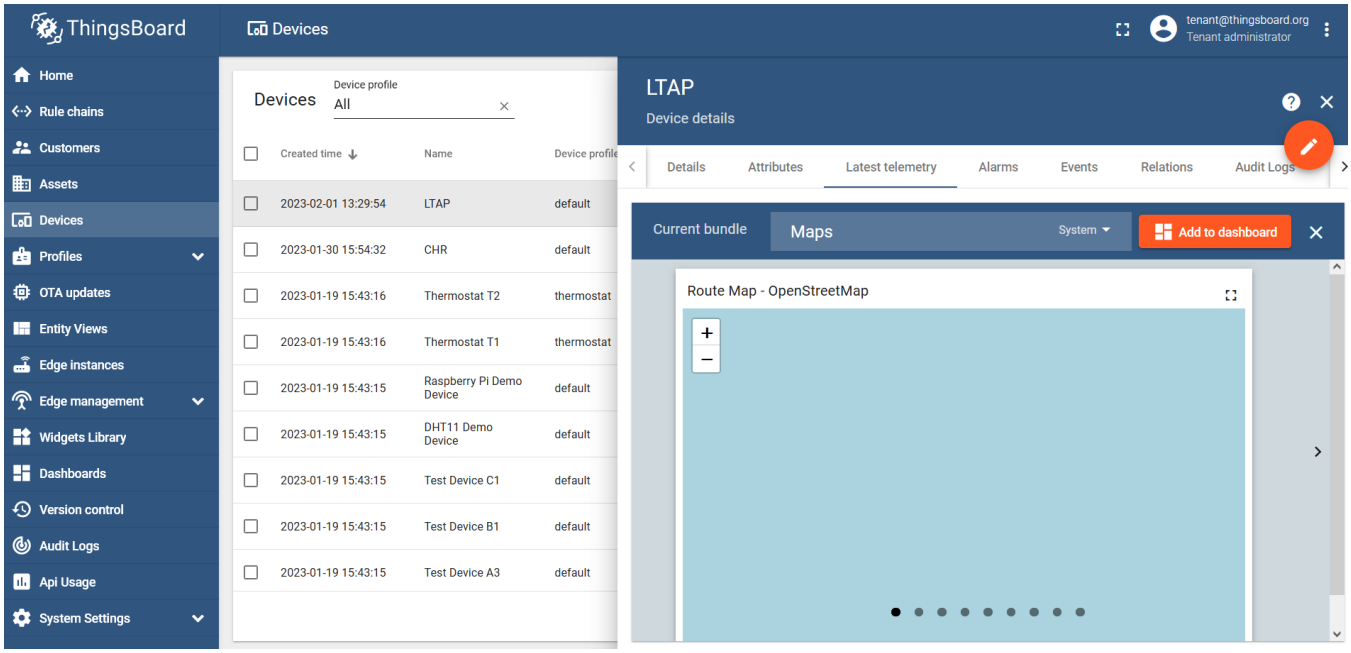
Data visualization using maps

ThingsBoard allows you to use Widgets to create visually appealing dashboards. In our case, we want to track our LTAP GPS coordinates, so we will need a map widget.

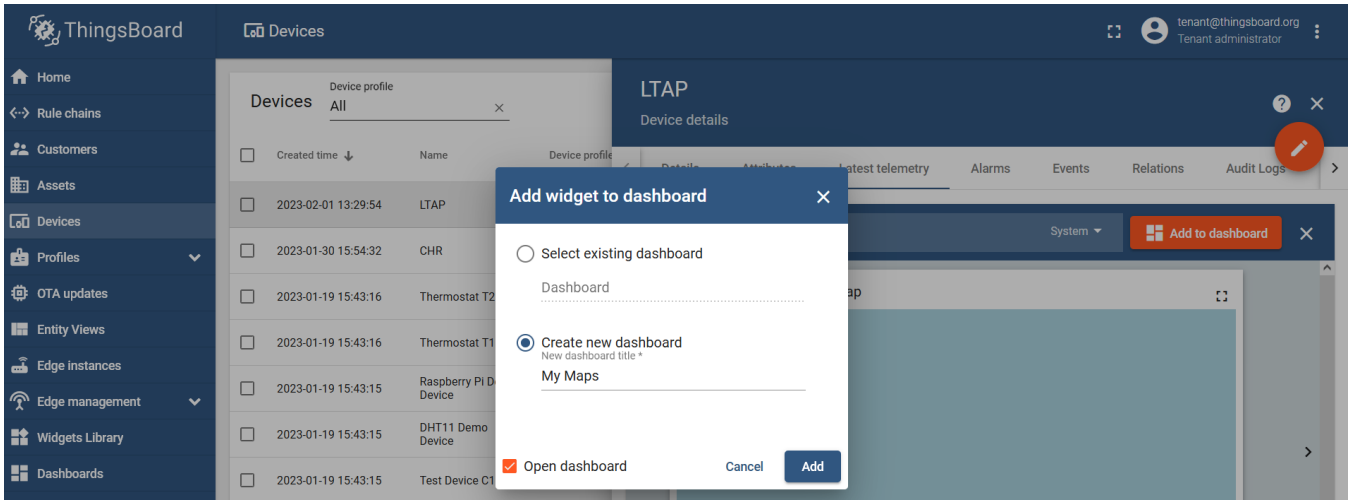
Select the latitude and longitude values and click on the "**Show on widget**" button:



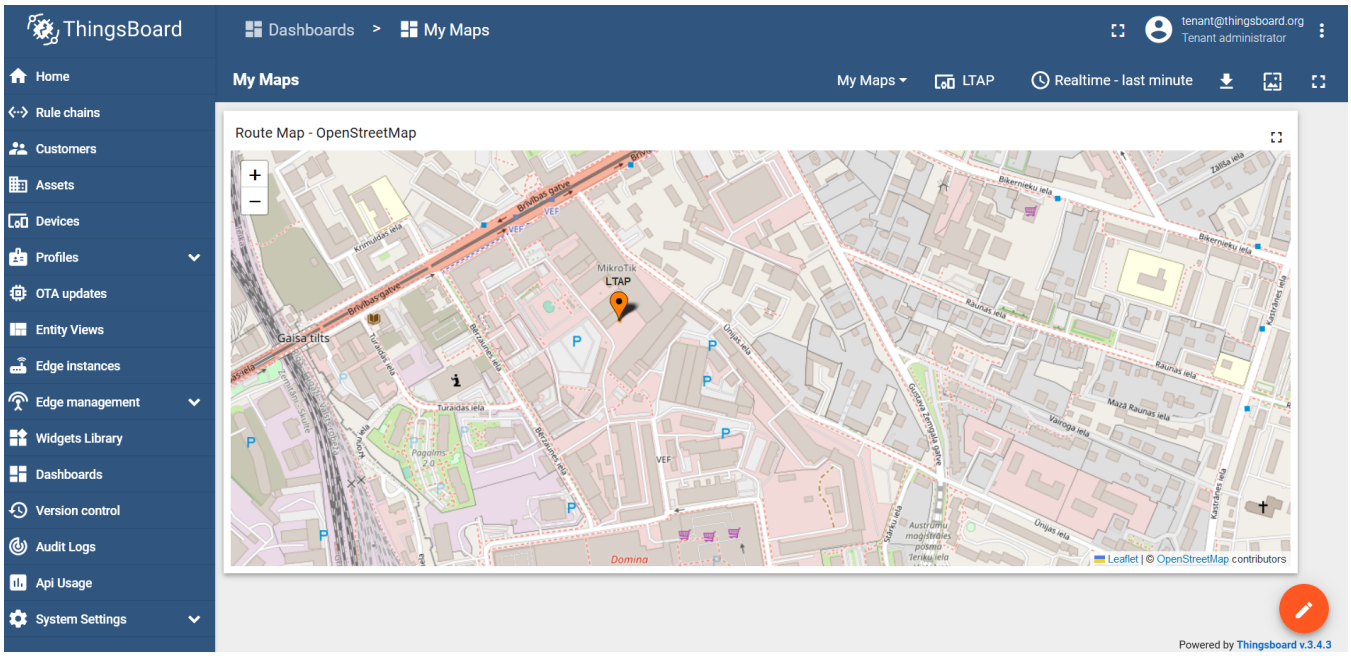
Find the "**Maps**" bundle and click on the "**Add to dashboard**":



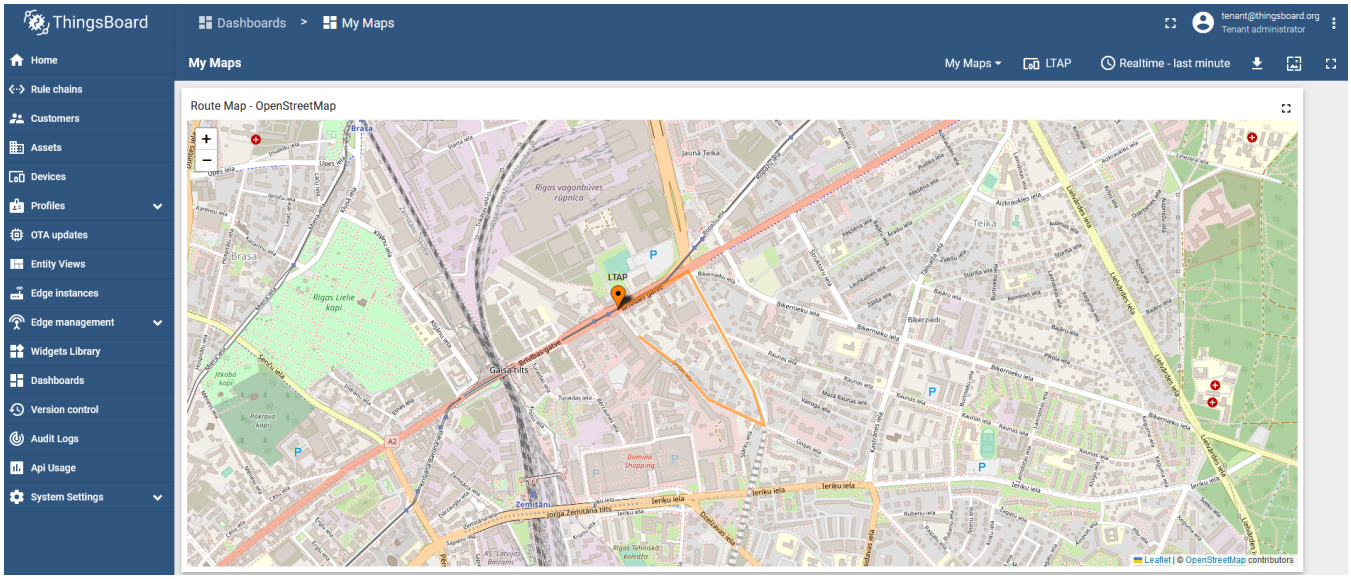
Select an existing dashboard or create a new one and name it however you like:



Run the script via the scheduler or manually and check the result:



Now, we can install it on a moving target and track its location:



LTE

Summary

```
Package: system
```

Support for Direct-IP mode type cards only. MBIM support is available in RouterOS v7 releases and MBIM driver is loaded automatically. If modem is not recognized in RouterOS v6 - Please test it in v7 releases before asking for support in RouterOS v6.

To enable access via a PPP interface instead of a LTE Interface, change direct IP mode with `/port firmware set ignore-directip-modem=yes` command and a reboot. Note that using PPP emulation mode you may not get the same throughput speeds as using the LTE interface emulation type.

For RouterOS v7 ignore-direct-modem parameter renamed to "mode" and moved to `/interface lte settings` menu.

LTE Client

```
Sub-menu: /interface lte
```

Properties

Property	Description
allow-roaming (*yes* | *no*; Default: **no**)	Enable data roaming for connecting to other countries data-providers. Not all LTE modems support this feature. Some modems, that do not fully support this feature, will connect to the network but will not establish an IP data connection with allow-roaming set to no.

apn-profiles (*string*; Default: **default**)	Which APN profile to use for this interface
band (*integer list*; Default: **""**)	LTE Frequency band used in communication `LTE Bands and bandwidths`
nr-band (*integer list*; Default: "")	5G NR Frequency band used in communication `5G NR Bands and bandwidths`
comment (*string*; Default: **""**)	Descriptive name of an item
disabled (*yes | no*; Default: **yes**)	Whether interface is disabled or not. By default it is disabled.
modem-init (*string*; Default: **""**)	Modem init string (AT command that will be executed at modem startup)
mtu (*integer*; Default: **1500**)	Maximum Transmission Unit. Max packet size that LTE interface will be able to send without packet fragmentation.
name (*string*; Default: **""**)	Descriptive name of the interface.
network-mode (*3g | gsm | lte | 5g*)	Select/force mode for LTE interface to operate with
operator (*integer*; Default: **""**)	used to lock device to specific operator full PLMN number is used for lock consisting from MCC+MNC. PLMN codes
pin (*integer*; Default: **""**)	SIM Card's PIN code.

APN profiles

All network related settings are moved under profiles, starting from RouterOS 6.41

```
Sub-menu: /interface lte apn
```

Property	Description
add-default-route (*yes | no*)	Whether to add default route to forward all traffic over the LTE interface.
apn (*string*)	Service Provider's Access Point Name
authentication (*pap | chap | none*; Default: **none**)	Allowed protocol to use for authentication
default-route-distance (*integer*; Default: **2**)	Sets distance value applied to auto created default route, if add-default-route is also selected. LTE route by default is with distance 2 to prefer wired routes over LTE
ip-type (*ipv4 | auto | ipv6*; Default: **auto**)	Requested PDN type
ipv6-interface (; Default:)	Interface on which to advertise IPv6 prefix
name (*string*; Default:)	APN profile name
number (*integer*; Default:)	APN profile number

passthrough-interface (; Default:)	Interface to passthrough IP configuration (activates passthrough)
passthrough-mac (*MAC*; Default: **auto**)	If set to auto, then will learn MAC from first packet
passthrough-subnet-selection (*auto / p2p*; Default: **auto**)	"auto" selects the smallest possible subnet to be used for the passthrough interface. "p2p" sets the passthrough interface subnet as /32 and picks gateway address from 10.177.0.0/16 range. The gateway address stays the same until the apn configuration is changed.
password (*string*; Default:)	Password used if any of the authentication protocols are active
use-network-apn (*yes* | *no*; Default: **yes**)	Parameter is available starting from RouterOS v7 and used only for MBIM modems. If set to yes, uses network provided APN.
use-peer-dns (*yes* | *no*; Default: **yes**)	If set to yes, uses DNS recieved from LTE interface
user (*integer*)	Username used if any of the authentication protocols are active

LTE settings

LTE and router-specific LTE settings. The menu is available starting from RouterOS v7.

```
Sub-menu: /interface lte settings
```

Property	Description
mode (*auto* | *mbim* | *serial*; *Default:* ***auto***)	Operation mode setting. • auto - automatically select the operation mode. • serial - provide only serial ports • mbim - switch modem into MBIM mode if possible
firmware-path (*string*)	Firmware path in host OS. Modem gobi firmware
external-antenna (*auto* | *both* | *div* | *main* | *none*; Default: **auto**)	This setting is only available for "Chateau" routers, except for Chateau 5G versions. • auto - measures the signal levels on both internal and external antennas and selects the antennas with the best signal(RSRP). • both - both antennas are set to external • div - diversity antenna set to external • main - main antenna set to external • none - no external antenna selected(using internal antennas)
external-antenna-selected ()	This setting is only available for "Chateau" routers, except for Chateau 5G versions. Shows the currently selected antenna if "**external-antenna**" is set to "auto"
sim-slot ()	This setting is available for routers that have switchable SIM slots (LtAP, SXT). Selection options differ between products.

Scanner

It is possible to scan LTE interfaces with `/interface lte scan` command. Example:

```
[admin@MikroTik] > /interface lte scan duration=60 number=0
Columns: OPERATOR, MCC-MNC, RSSI, RSRP, RSRQ
OPERATOR  MCC-MNC  RSSI    RSRP    RSRQ
LMT         24701  -36dBm  -63dBm  -7dB
```

Available properties:

Property	Description
duration (*integer*)	Duration of scan in seconds
freeze-frame-interval (*integer*)	time between data printout
number (*integer*)	Interface number or name

User Info command

It is possible to send special "info" command to LTE interface with `/interface lte info` command. In RouterOS v7 this command is moved to `/interface lte monitor` menu.

Properties (Up to 6.40)

Property	Description
user-command (*string*; Default: **""**)	send a command to LTE card to extract useful information, e.g. with AT commands
user-command-only (*yes* | *no*; Default:)	

User at-chat command

It is possible to send user defined "at-chat" command to LTE interface with `/interface lte at-chat` command.

```
[admin@MikroTik] > /interface lte at-chat lte1 input="AT"
  output: OK
```

It is also possible to use the "wait" parameter *wait=yes* with the command to make "at-chat" wait for 5 seconds and return all the output instead of returning only the first received data, this is useful for some commands that return multiline output or a large block of data.

```
[admin@MikroTik] > interface lte at-chat lte1 input="at+qcfg=?"
  output:

[admin@MikroTik] > interface lte at-chat lte1 input="at+qcfg=?" wait=yes
  output: +QCFG: "rrc",(0-5)
          +QCFG: "hsdpacat",(6,8,10-24)
          +QCFG: "hsupacat",(5,6)
          +QCFG: "pdp/duplicatechk",(0,1)
          +QCFG: "risignaltype",("respective","physical")
          +QCFG: "lte/bandprior",(1-43),(1-43),(1-43)
          +QCFG: "volte_disable",(0,1)
          +QCFG: "diversity/config",(4,6),(1-4),(0)
          +QCFG: "div_test_mode",(0,1)
          +QCFG: "usbspeed",("20","30")
          +QCFG: "data_interface",(0,1),(0,1)
          +QCFG: "pcie/mode",(0,1)
          +QCFG: "pcie_mbim",(0,1)
          +QCFG: "sms_control",(0,1),(0,1)
          +QCFG: "call_control",(0,1),(0,1)
          +QCFG: "usb/maxpower",(0-900)
          +QCFG: "efratctl",(0,1)
          +QCFG: "netmaskset",(0,1)[,<netmask>]
          +QCFG: "mmwave",ant_chip,ant_type
          +QCFG: "gatewayset",(0,1)[,<gateway>]
          +QCFG: "clat",(0,1),(0,1),<prefix>,(0,32,40,48,56,64,96),<fqdn>,(0,1),(0,1,2,4,8),(0,1),(0,1),(0,1,2),
(0,1,2)
          +QCFG: "usage/apmem"
          +QCFG: "enable_gea1"[,(0,1)]
          +QCFG: "dhcppktfltr",(0,1)
          OK
```

You can also use "at-chat" function in scripts and assign command output to variable.

```
[admin@MikroTik] > :global "lte_command" [/interface lte at-chat lte1 input="AT+CEREG?" as-value ]
[admin@MikroTik] > :put $"lte_command"
output=+CEREG: 0,1
OK
```

Quick setup example

Start with network settings -

> ⚠ This guide is for RouterOS versions starting from 6.41

Start with network settings - Add new connection parameters under LTE apn profile (provided by network provider):

```
/interface lte apn add name=profile1 apn=phoneprovider.net authentication=chap password=web user=web
```

Select newly created profile for LTE connection:

```
/interface lte set [find] apn-profiles=profile1
```

LTE interface should appear with running (R) flag:

```
[admin@MikroTik] > /interface lte print
Flags: X - disabled, R - running
0 R name="lte1" mtu=1500 mac-address=AA:AA:AA:AA:AA:AA
```

If required, add NAT Masquerade for LTE Interface to get internet to the local network:

```
/ip firewall nat add action=masquerade chain=srcnat out-interface=lte1
```

After interface is added, you can use "info" command to see what parameters client acquired (parameters returned depends on LTE hardware device):

```
[admin@MikroTik] > interface/lte/monitor
lte1
            status: connected
             model: EG18-EA
          revision: EG18EAPAR01A12M4G
  current-operator: LMT
    current-cellid: 3103242
            enb-id: 12122
         sector-id: 10
        phy-cellid: 480
        data-class: LTE
    session-uptime: 15m54s
              imei: 86981604098XXXX
              imsi: 24701060267XXXX
              uicc: 8937101122102057XXXX
      primary-band: B3@20Mhz earfcn: 1300 phy-cellid: 480
     dl-modulation: qpsk
               cqi: 7
                ri: 2
               mcs: 1
              rssi: -68dBm
              rsrp: -97dBm
              rsrq: -9dB
              sinr: 6dB
```

Passthrough Example

Starting from RouterOS v6.41 some LTE interfaces support LTE Passthrough feature where the IP configuration is applied directly to the client device. In this case modem firmware is responsible for the IP configuration and router is used only to configure modem settings - APN, Network Technologies and IP-Type. In this configuration the router will not get IP configuration from the modem. The LTE Passthrough modem can pass both IPv4 and IPv6 addresses if that is supported by modem. Some modems support multiple APN where you can pass the traffic from each APN to a specific router interface.

Passthrough will only work for one host. Router will automatically detect MAC address of the first received packet and use it for the Passthrough. If there are multiple hosts on the network it is possible to lock the Passthrough to a specific MAC. On the host on the network where the Passthrough is providing the IP a DHCP-Client should be enabled on that interface to. Note, that it will not be possible to connect to the LTE router via public lte ip address or from the host which is used by the passthrough. It is suggested to create additional connection from the LTE router to the host for configuration purposes. For example vlan interface between the LTE router and host.

To enable the Passthrough a new entry is required or the default entry should be changed in the '/interface lte apn' menu

Passthrough is not supported by all chipsets.

Examples.

To configure the Passthrough on ether1:

```
[admin@MikroTik] > /interface lte apn add apn=apn1 passthrough-interface=ether1
[admin@MikroTik] > /interface lte set lte1 apn-profiles=apn1
```

To configure the Passthrough on ether1 host 00:0C:42:03:06:AB:

```
[admin@MikroTik] > /interface lte apn add apn=apn1 passthrough-interface=ether1 passthrough-mac=00:0C:42:03:06:
AB
[admin@MikroTik] > /interface lte set lte1 apn-profiles=apn1
```

To configure multiple APNs on ether1 and ether2:

```
[admin@MikroTik] > /interface lte apn add apn=apn1 passthrough-interface=ether1
[admin@MikroTik] > /interface lte apn add apn=apn2 passthrough-interface=ether2
[admin@MikroTik] > /interface lte set lte1 apn-profiles=apn1,apn2
```

To configure multiple APNs with the same APN for different interfaces:

```
[admin@MikroTik] > /interface lte apn add name=interface1 apn=apn1
[admin@MikroTik] > /interface lte apn add name=interface2 apn=apn1 passthrough-interface=ether1
[admin@MikroTik] > /interface lte set lte1 apn-profiles=interface1
[admin@MikroTik] > /interface lte set lte2 apn-profiles=interface2
```

Dual SIM

Boards with switchable SIM slots

RouterBoard	Modem slot	SIM slots	Switchable
LtAP	lower	2 | 3	Y
	upper	1	N
LtAP mini		up | down	Y
SXT R		a | b	Y

SIM slots switching commands

- RouterOS v7

```
/interface lte settings set sim-slot=down
```

- RouterOS v6 after 6.45.1

```
/system routerboard modem set sim-slot=down
```

- RouterOS v6 pre 6.45.1:

```
/system routerboard sim set sim-slot=down
```

For more reference please see board block diagram, Quick Guide and User manual.

Usage Example

Follow this link - Dual SIM Application, to see examples of how to change SIM slot based on roaming status and in case the interface status is down with help of RouterOS scripts and scheduler.

Tips and Tricks

This paragraph contains information for additional features and usage cases.

Find device location using Cell information

On devices using R11e-LTE International version card (wAP LTE kit) some extra information is provided under info command (from 6.41rc61)

```
  current-operator: 24701
               lac: 40
    current-cellid: 2514442
```

Property	Description
current-operator (*integer*; Default:)	Contains MCC and MNC. For example: current-operator: 24701 breaks to: MCC=247 MNC=01
lac (*integer*; Default:)	location area code (LAC)
current-cellid (*integer*; Default:)	Station identification number

Values can be used to find location in databases: Cell Id Finder

Using Cell lock

It is possible to lock R11e-LTE, R11e-LTE6 and R11e-4G modems and equipped devices to exact LTE tower. LTE info command provides currently used cellular tower information:

```
        phy-cellid: 384
            earfcn: 1300 (band 3, bandwidth 20Mhz)
```

Property	Description
phy-cellid (*integer*; Default:)	Physical Cell Identification (PCI) of currently used cell tower.
earfcn (*integer*; Default:)	Absolute Radio Frequency Channel Number

Exact tower location as well as available bands and other information can be acquired from mobile carrier or by using online services:

CellMapper

By using those acquired variables it's possible to send AT command to modem for locking to tower in current format:

for R11e-LTE and R11e-LTE6

```
AT*Cell=<mode>,<NetworkMode>,<band>,<EARFCN>,<PCI>

where

<mode> :
0 – Cell/Frequency disabled
1 – Frequency lock enabled
2 – Cell lock enabled

<NetworkMode>
0 – GSM
1 – UMTS_TD
2 – UMTS_WB
3 – LTE

<band>
Not in use, leave this blank

<EARFCN>
earfcn from lte info

<PCI>
phy-cellid from lte info
```

To lock modem at previously used tower at-chat can be used:

```
/interface lte at-chat lte1 input="AT*Cell=2,3,,1300,384"
```

For R11e-LTE all set on locks are lost after reboot or modem reset. Cell data can be also gathered from "cell-monitor".

For R11e-LTE6 cell lock works only for the primary band, this can be useful if you have multiple channels on the same band and you want to lock it to a specific earfcn. Note, that cell lock is not band-specific and for ca-band it can also use other frequency bands, unless you use band lock.

Use cell lock to set the primary band to the 1300 earfcn and use the second channel for the ca-band:

```
/interface lte at-chat lte1 input="AT*Cell=2,3,,1300,138"
```

Now it uses the earfcn: 1300 for the primary channel:

```
         primary-band: B3@20Mhz earfcn: 1300 phy-cellid: 138
              ca-band: B3@5Mhz earfcn: 1417 phy-cellid: 138
```

You can also set it the other way around:

```
/interface lte at-chat lte1 input="AT*Cell=2,3,,1417,138"
```

Now it uses the earfcn: 1417 for the primary channel:

```
         primary-band: B3@5Mhz earfcn: 1417 phy-cellid: 138
              ca-band: B3@20Mhz earfcn: 1300 phy-cellid: 138
```

For R11e-LTE6 modem cell lock information will not be lost after reboot or modem reset. To remove cell lock use at-chat command:

```
/interface lte at-chat lte1 input="AT*Cell=0"
```

for R11e-4G

```
AT%CLCMD=<mode>,<mode2>,<EARFCN>,<PCI>,<PLMN>
AT%CLCMD=1,1,3250,244,\"24705\"

where

<mode> :
0 – Cell/Frequency disabled
1 – Cell lock enabled

<mode2> :
0 - Save lock for first scan
1 - Always use lock
(after each reset modem will clear out previous settings no matter what is used here)

<EARFCN>
earfcn from lte info

<PCI>
phy-cellid from lte info

<PLMN>
Mobile operator code
```

All PLMN codes available here this variable can be also left blank

To lock modem to the cell - modem needs to be in non operating state, easiest way for **R11e-4G** modem is to add CellLock line to "modem-init" string:

```
/interface lte set lte1 modem-init="AT%CLCMD=1,1,3250,244,\"24705\""
```

Multiple cells can also be added by providing list instead of one tower information in following format:

```
AT%CLCMD=<mode>,<mode2>,<EARFCN_1>,<PCI_1>,<PLMN_1>,<EARFCN_2>,<PCI_2>,<PLMN_2>
```

For example to lock to two different PCIs within same band and operator:

```
/interface lte set lte1 modem-init="AT%CLCMD=1,1,6300,384,\"24701\",6300,385,\"24701\""
```

for Chateau LTE12, Chateau 5G, LHG LTE18 and ATL LTE18

```
AT+QNWLOCK="common/4g",<num of cells>,[[<freq>,<pci>],...]
AT+QNWLOCK=\"common/4g\",1,6300,384

where

<num of cells>
number of cells to cell lock

<freq>
earfcn from lte info

<pci>
phy-cellid from lte info
```

Single cell lock example:

```
/interface lte at-chat lte1 input="AT+QNWLOCK=\"common/4g\",1,3050,448"
```

Query current configuration:

```
/interface lte at-chat lte1 input="AT+QNWLOCK=\"common/4g\""
```

Multiple cells can also be added to the cell lock. For example to lock to two different cells:

```
/interface lte at-chat lte1 input="AT+QNWLOCK=\"common/4g\",2,3050,448,1574,474"
```

To remove the cell lock use this at-chat command:

```
/interface lte at-chat lte1 input="at+qnwlock=\"common/4g\",0"
```

1. Cell lock information will not be saved after a reboot or modem reset. 2. AT+QNWLOCK command can lock the cell and frequency. Therefore, the module can be given priority to register to the locked cell, however, according to the 3gpp protocol, the module will be redirected or handover to a cell with better signal instructions, even if it is not within the lock of the command. This phenomenon is normal.

for Fibocom FG621

AT+GTCELLLOCK=<mode>[,<rat>,<type>,<earfcn>[,<PCI>]]

where

< mode >: integer type; 0 Disable this function 1 Enable this function 2 Add new cell to be locked

<rat>: integer type; 0 LTE 1 WCDMA

<type>: integer type; 0 Lock PCI 1 Lock frequency

<earfcn>: integer type; the range is 0-65535.

<PCI>: integer type; If second parameter value is 0, the range is 0-503 for LTE If second parameter value is 1, the range is 0-512 for WCDMA

Example:

```
/interface lte at-chat lte1 input="AT+GTCELLLOCK=1,0,0,6175,176"
```

Cell Monitor

Cell monitor allows to scan available nearby mobile network cells:

```
[admin@MikroTik] > /interface lte cell-monitor lte1
PHY-CELLID BAND         PSC EARFCN                RSRP          RSRQ          RSSI         SINR
        49 B20              6300               -110dBm       -19.5dB
       272 B20              6300               -116dBm       -19.5dB
       374 B20              6300               -108dBm         -16dB
       384 B1               150                -105dBm       -13.5dB
       384 B3               1300               -106dBm         -12dB
       384 B7               2850               -107dBm       -11.5dB
       432 B7               2850               -119dBm       -19.5dB
```

Gathered data can be used for more precise location detection or for Cell lock.

⚠ Not all modems support this feature

Troubleshooting

Enable LTE logging:

```
[admin@MikroTik] > /system logging add topics=lte
```

Check for errors in log:

```
[admin@MikroTik] > /log print

11:08:59 lte,async lte1: sent AT+CPIN?
11:08:59 lte,async lte1: rcvd +CME ERROR: 10
```

search for CME error description online,

in this case: CME error 10 - SIM not inserted

Locking band on Huawei and other modems

To lock band for Huawei modems `/interface lte set lte1 band=""` option can't be used.

It is possible to use AT commands to lock to desired band manually.

To check all supported bands run at-chat command:

```
[admin@MikroTik] /interface lte at-chat lte1 input="AT^SYSCFGEX=\?"

output: ^SYSCFGEX: ("00","03","02","01","99"),((2000004e80380,"GSM850/GSM900/GSM1800/GSM1900/WCDMA BCI/WCDMA
BCII/WCDMA BCV/WCDMA BCVIII"),
(3fffffff,"All Bands")),(0-2),(0-4),((800d7,"LTE BC1/LTE BC2/LTE
BC3/LTE BC5/LTE BC7/LTE BC8/LTE BC20"),(7fffffffffffffff,"All Bands"))
OK
```

Example to lock to LTE band 7:

```
[admin@MikroTik] /interface lte set lte1 modem-init="AT^SYSCFGEX=\"03\",3FFFFFFF,2,4,40,,"
```

Change last part **40** to desired band specified hexadecimal value where:

```
4 LTE BC3
40 LTE BC7
80000 LTE BC20
7FFFFFFFFFFFFFFF  All bands
etc
```

All band HEX values and AT commands can be found in Huawei AT Command Interface Specification guide

Check if band is locked:

```
[admin@MikroTik] /interface lte at-chat lte1 input="AT^SYSCFGEX\?"

output: ^SYSCFGEX: "03",3FFFFFFF,0,2,40
OK
```

For more information check modem manufacturers AT command reference manuals.

mPCIe modems with RB9xx series devices

In case your modem is not being recognized after a soft reboot, then you might need to add a delay before the USB port is being initialized. This can be done using the following command:

```
/system routerboard settings set init-delay=5s
```

Boards with USB-A port and mPCIe

Some devices such as specific RB9xx's and the RBLtAP-2HnD share the same USB lines between a single mPCIe slot and a USB-A port. If auto switch is not taking place and a modem is not getting detected, you might need to switch manually to either use the USB-A or mini-PCIe:

```
/system routerboard usb set type=mini-PCIe
```

Modem firmware upgrade

⚠ Before attempting LTE modem firmware upgrade - upgrade RouterOS version to latest releases How To Upgrade RouterOS

Starting from RouterOS version 6.44beta20 it is possible to upgrade modems firmware. The firmware upgrade is also possible for the Chateau series products starting from 7.1beta1 version.

Firmware update is available only as FOTA Firmware Over The Air - firmware upgrade can only be done through working mobile connection for:

-)R11e-LTE
-)R11e-LTE-US

Firmware update available as FOTA and as well as upgrade from file for:

-)R11e-4G
-)R11e-LTE6

Firmware update available as FOTA with access to the internet over any interface:

-)EG12-EA (Chateau LTE12)
-)RG502Q-EA (Chateau 5G)
-)EG18-EA (LHG LTE18)

Firmware updates usually includes small improvements in stability or small bug fixes that can't be included into RouterOS.

Check currently used firmware version by running:

```
[admin@MikroTik] > /interface lte info lte1 once

-----
revision: "MikroTik_CP_2.160.000_v008"
-----
```

Check if new firmware is available:

```
[admin@MikroTik] > /interface lte firmware-upgrade lte1
  installed: MikroTik_CP_2.160.000_v008
     latest: MikroTik_CP_2.160.000_v010
```

Upgrade firmware:

```
[admin@MikroTik] > /interface lte firmware-upgrade lte1 upgrade=yes
  status: downloading via LTE connection (>2min)
```

Whole upgrade process may take up to 10 minutes, depending on mobile connection speed.

After successful upgrade issue USB power-reset, reboot device or run AT+reset command, to update modem version readout under info command:

```
[admin@MikroTik] > /interface lte at-chat lte1 input="AT+reset"
```

if modem has issues connecting to cells after update, or there are any other unrelated issues - wipe old configuration with:

```
/interface lte at-chat lte1 input="AT+RSTSET"
```

Avoiding tethering speed throttling

Some operators (TMobile, YOTA etc.) allows unlimited data only for device SIM card is used on, all other data coming from mobile hotspots or tethering is highly limited by volume or by throughput speed. Some sources have found out that this limitation is done by monitoring TTL (Time To Live) values from packets to determinate if limitations need to be applied (TTL is decreased by 1 for each "hop" made). RouterOS allows changing the TTL parameter for packets going from the router to allow hiding sub networks. Keep in mind that this may conflict with fair use policy.

```
IPv4 mangle rule:
/ip firewall mangle
add action=change-ttl chain=postrouting new-ttl=set:65 out-interface=lte1 passthrough=yes
IPv6 mangle rule:
/ipv6 firewall mangle
add action=change-hop-limit chain=postrouting new-hop-limit=set:65 passthrough=yes
```

More information: YOTA, TMobile

Unlocking SIM card after multiple wrong PIN code attempts

After locking SIM card, unlock can be done through "at-chat"

Check current PIN code status:

```
/interface lte at-chat lte1 input="at+cpin\?"
```

If card is locked - unlock it by providing:

```
/interface lte at-chat lte1 input="AT+CPIN=\"PUK_code\",\"NEW_PIN\""
```

Replace PUK_code and NEW_PIN with matching values.

The command for sim slot selection changes in v6.45.1 and again in v7. Some device models like SXT, have SIM slots named "a" and "b" instead of "up" and down"

PPP

Overview

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP in RouterOS is based on RFC 1661 standard.

Introduction

The basic purpose of PPP at this point is to transport Layer-3 packets across a Data Link layer point-to-point link. Packets between both peers are assumed to deliver in order.

PPP is comprised of three main components:

1. A method for encapsulating multi-protocol datagrams.
2. A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection.
3. A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

Detailed PPP packet processing in RouterOS you can see in the Packet Flow Diagram.

PPP Client

```
/interface ppp-client
```

PPP Client example

This is an example of how to add a client using an exposed serial port from an LTE modem.

```
/interface ppp-client add apn=yourapn dial-on-demand=no disabled=no port=usb2
```

The dial-on-demand should to be set to 'no' for a continuous connection.

PPP Server

```
/interface ppp-server
```

SMS

Summary

It is possible to connect the GSM modem to the RouterOS device and use it to send and receive SMS messages. RouterOS lists such modem as a serial port that appears in the '*/port print*' listing. GSM standard defines AT commands for sending SMS messages and defines how messages should be encoded in these commands.

'advanced tools package provides command '`/tool sms send`' that uses standard GSM AT commands to send SMS.

Sending

```
/tool sms send
```

Example

Sending command for ppp interface:

```
/tool sms send usb3 "20000000" \ message="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@#\$%^&*(){}[]
\"'~"
```

For LTE interface use LTE interface name in the port field:

```
/tool sms send lte1 "20000000" \ message="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@#\$%^&*(){}[]
\"'~"
```

Parameter	Description
port (*string*)	Name of port from */port* list that GSM modem is attached to.
phone-number (*string*)	Recepient phone number. Allowed characters are "0123456789*#abc". If first character is "+" then phone number type is set to *international*, otherwise it is set to *unknown*.
channel (*integer*)	Which modem channel to use for sending.
message (*string*)	Message contents. It is encoded using GSM 7 encoding (UCS2 currently is not supported), so message length is limited to 160 characters (characters ^{}\[]~
smsc (*string*)	
type (*string*)	If set to *class-0*, then send class 0 SMS message. It is displayed immedeately and not stored in phone.
sms-storage (*string*)	Select storage where to save received SMS (modem/sim)

USSD messages

USSD (Unstructured Supplementary Service Data) messages can be used to communicate with mobile network provider to receive additional information, enabling additional services or adding funds to prepaid cards. USSD messages can be processed by using AT commands (commands can differ or even may be blocked on some modems).

3G or GSM network modes must be activated to use this functionality, as it's not supported under LTE only mode (**R11e-LTE** modem auto switches to 3G mode to send out USSD message).

PDU (Protocol Data Unit) message and its decrypted version is printed under LTE debug logging.

Example

Check if LTE debug logging is active:

```
/system logging print
Flags: X - disabled, I - invalid, * - default
# TOPICS ACTION PREFIX
0 * info memory
1 * error memory
2 * warning memory
3 * critical echo
```

If there is no logging entry add it by running this command:

```
/system logging add topics=lte,!raw

/system logging print
Flags: X - disabled, I - invalid, * - default
# TOPICS ACTION PREFIX
0 * info memory
1 * error memory
2 * warning memory
3 * critical echo
4 lte,!raw memory
```

To recieve account status from ***245#**

```
/interface lte at-chat lte1 input="AT+CUSD=1,\"*245#\",15"
output: OK
/log print
11:51:20 lte,async lte1: sent AT+CUSD=1,"*245#",15
11:51:20 lte,async lte1: rcvd OK
11:51:23 lte,async,event +CUSD: 0,"EBB79B1E0685E9ECF4BADE9E03", 0
11:51:23 gsm,info USSD: konta atlikums
```

Receiving

Since v3.24 RouterOS also supports receiving of SMS messages, and can execute scripts, and even respond to the sender.

Before router can receive SMS, relevant configuration is required in general **/tool sms** menu. Following parameters are configurable:

Parameter	Description
allowed-number (*string*; Default: "")	Sender number that will be allowed to run commands, must specify country code ie. +371XXXXXXX
channel (*integer*; Default: **0**)	Which modem channel to use for receiving.

keep-max-sms (*integer*; Default: **0**)	Maximum number of messages that will be saved. If you set this bigger than SIM supports, new messages will not be received! Replaced with **auto-erase** parameter starting from RouterOS v6.44.6
auto-erase (*yes* | *no*; Default: **no**)	SIM storage size is read automatically. When **auto-erase=no** new SMS will not be received if storage is full. Set **auto-erase=yes** to delete the oldest received SMS to free space for new ones automatically. Available starting from v6.44.6
port (*string*; Default: (**unknown**))	Modem port (modem can be used only by one process "/port> print")
receive-enabled (*yes* | *no*; Default: **no**)	Must be turned on to receive messages
secret (*string*; Default: "")	the secret password, mandatory

Basic Example configuration to be able to view received messages:

```
/tool sms set receive-enabled=yes port=lte1

/tool/sms/print
           status: running
  receive-enabled: yes
             port: lte1
          channel: 0
           secret:
   allowed-number:
       auto-erase: no
          sim-pin:
        last-ussd:
```

Inbox

```
/tool sms inbox
```

If you have enabled the reader, you will see incoming messages in this submenu:

Read-only properties:

Property	Description
phone (*string*)	Senders phone number.
message (*string*)	Message body
timestamp (*time*)	Time when message was received. It is the time sent by operator, not the router's local time.
type (*string*)	Message type

Syntax

```
:cmd SECRET script NAME [[ VAR[=VAL] ] ... ]
```

- **SECRET** - the password
- **NAME** - name of the script that's available in "/system script"
- **VAR** - variables that will be passed to the script (can be passed as VAR or as VAR=value), separated by spaces.

Other things to remember:

- *Parameters can be put into quotes "VAR"="VAL" if necessary.
- *Escaping of values is not supported (VAR="\"").
- *Combined SMS are not supported, every SMS will be treated separately
- * 16Bit unicode messages are not supported

- * SMS are decoded with the standard GSM7 alphabet, so you can't send in other encodings, otherwise it will be decoded incorrectly

Examples

Wrong:

```
:cmd script mans_skripts
:cmd slepens script mans skripts
:cmd slepens script mans_skripts var=
:cmd slepens script mans_skripts var= a
:cmd slepens script mans_skripts var=a a
```

Right:

```
:cmd slepens script mans_skripts
:cmd slepens script "mans skripts"
:cmd slepens script mans_skripts var
:cmd slepens script mans_skripts var=a
:cmd slepens script mans_skripts var="a a"
```

Debugging

/tool sms send command is logging data that is written and read. It is logged with tags *gsm,debug,write* and *gsm,debug,read* For more information see syst em logging.

Implementation details

AT+CMGS and *AT+CMGF* commands are used. Port is acquired for the duration of the command and cannot be used concurently by another RouterOS component. Message sending process can take a long time, it times out after a minute and after two seconds during initial AT command exchange.

Dual SIM Application

Summary

The first script example shows how to switch between SIM slots in case mobile roaming is detected for LtAP mini devices. This could be useful for mobile vehicle applications, where cars, buses or trains could drive abroad and should use two SIM cards (one for a home network, other for a roaming network). Since RouterOS has a roaming status in the LTE monitor (displayed only when roaming) we can use this in RouterOS scripts to change SIM cards accordingly.

The second script example shows how to switch between the SIM cards in case mobile connection is lost on the currently selected one.

Note: Keep in mind that these are just examples of how to utilize dual SIM slots. For real-life production environments, a proper testing should be carried out, so try to optimize them and add new features according to your needs.

Initial settings

First, make sure you have correctly set up LTE network parameters (provided by the mobile network operator) for each SIM card. You can use default APN profile or create two separate ones, follow this link - LTE#Quicksetupexample. This example uses default APN profile.

After that, enable data roaming for connecting to other countries data-providers with the following command. This allows to keep track of roaming status.

```
/interface lte set [find name=lte1] allow-roaming=yes
```

Then, choose which SIM slots will be used for home and roaming networks. In this example, we use slot "down" for home and slot "up" for roaming network. Use the following command to switch between active slots.
Note: command for sim slot selection changes in v6.45.1. And some device models like SXT, have SIM slots named "a" and "b" instead of "up" and down"

Command for pre 6.45.1:

```
/system routerboard sim set sim-slot=down
```

Command after 6.45.1:

```
/system routerboard modem set sim-slot=down
```

Command in RouterOS v7 version:

```
/interface lte settings set sim-slot=down
```

After changing SIM slots, LTE modem will be restarted. It can take some time (depending on modem and board around 30 seconds) to fully initialize it, so make sure you test your modem.

Roaming script example

Now create a script that will run with a scheduler. This script example is going through few key points:

- Check if LTE interface is initialized (shows in `/interface lte` list), otherwise try a power reset
- Check if LTE connection is established (interface is in "running" state), otherwise create a log entry and simply wait for next scheduler
- Read currently used LTE slot and make a decision whether to change SIM slots based on roaming status

Let's call this script "roamingScript", and see below the source:

```
{
# Setup and read current values, "up" SIM slot will be used for roaming, "down" for home network
:global simSlot [/system routerboard sim get sim-slot]
:global timeoutLTE 60
:global timeoutConnect 60

# Wait for LTE to initialize for maximum "timeoutLTE" seconds
:local i 0
:local isLTEinit false
:while ($i<$timeoutLTE) do={
    :foreach n in=[/interface lte find] do={:set $isLTEinit true}
    :if ($isLTEinit=true) do={
        :set $i $timeoutLTE
    }
    :set $i ($i+1)
    :delay 1s
}

# Check if LTE is initialized, or try power-reset the modem
:if ($isLTEinit=true) do={
    # Wait for LTE interface to connect to mobile network for maximum "timeoutConnet" seconds
    :local isConnected false
    :set $i 0
    :while ($i<$timeoutConnect) do={
        :if ([/interface lte get [find name="lte1"] running]=true) do={
            :set $isConnected true
            :set $i $timeoutConnect
        }
        :set $i ($i+1)
        :delay 1s
    }
    # Check if LTE is connected
    if ($isConnected=true) do={
        :local Info [/interface lte monitor lte1 once as-value]
        :local isRoaming ($Info->"roaming")
        # Check which SIM slot is used
        :if ($simSlot="down") do={
            # If "down" (home) slot, check roaming status
            :if ($isRoaming=true) do={
                :log info message="Roaming detected, switching to SIM UP (Roaming)"
                /system routerboard sim set sim-slot=up
            }
        } else={
            # Else "up" (roaming) slot, check roaming status
            :if (!$isRoaming=true) do={
                :log info message="Not roaming, switching to SIM DOWN (Home)"
                /interface lte settings set sim-slot=down
            }
        }
    } else={
        :log info message="LTE interface did not connect to network, wait for next scheduler"
    }
} else={
    :log info message="LTE modem did not appear, trying power-reset"
    /system routerboard usb power-reset duration=5s
}
}
```

Failover script example

Now create a script that will run with a scheduler. This script example is going through few key points:

- Check if LTE interface is initialized (shows in `/interface lte` list), otherwise try a power reset
- Check if LTE connection is established (interface is in "running" state), otherwise create a log entry and simply wait for next scheduler
- Read currently used LTE slot and make a decision whether to change SIM slots based on interface status

Note: Keep in mind that the SIM slot will only be changed if the current one is not able to connect to the network if you need to switch back to the main SIM card you need to schedule another action that does it at a certain time. It is not possible to know if the other SIM card is in service without switching back to it.

Let's call this script "failoverScript", and see below the source:

```
{
# Setup and read current values
:global simSlot [/system routerboard modem get sim-slot]
:global timeoutLTE 60
:global timeoutConnect 60

# Wait for LTE to initialize for maximum "timeoutLTE" seconds
:local i 0
:local isLTEinit false
:while ($i<$timeoutLTE) do={
    :foreach n in=[/interface lte find] do={:set $isLTEinit true}
    :if ($isLTEinit=true) do={
        :set $i $timeoutLTE
    }
    :set $i ($i+1)
    :delay 1s
}

# Check if LTE is initialized, or try power-reset the modem
:if ($isLTEinit=true) do={
    # Wait for LTE interface to connect to mobile network for maximum "timeoutConnet" seconds
    :local isConnected false
    :set $i 0
    :while ($i<$timeoutConnect) do={
        :if ([/interface lte get [find name="lte1"] running]=true) do={
            :set $isConnected true
            :set $i $timeoutConnect
        }
        :set $i ($i+1)
        :delay 1s
    }
    # Check if LTE is connected
    if ($isConnected=false) do={
    # Check which SIM slot is used
        :if ($simSlot="down") do={
            # If "down" slot, switch to up
        :log info message="LTE down, switching slot to UP"
            /interface lte settings set sim-slot=up
    }
        :if ($simSlot="up") do={
            # If "up" slot, switch to down
        :log info message="LTE down, switching slot to DOWN"
            /interface lte settings set sim-slot=down
            }
        } else={
            # Else "running"
            :if ($isConnected=true) do={
                :log info message="LTE UP"
            } else={
        :log info message="LTE interface did not connect to network, wait for next scheduler"
        }
    }
    } else={
    :log info message="LTE modem did not appear, trying power-reset"
    /system routerboard usb power-reset duration=5s
}
}
```

Setting up scheduler

Last, create your scheduler that will run the previously created script. Choose a proper scheduler interval, so two or more events do not overlap with each other. For this example above, 3 minutes will be enough.

```
/system scheduler add interval=3m on-event=roamingScript name=Roaming
```

```
/system scheduler add interval=3m on-event=failoverScript name=Failover
```

Keep in mind that "home" SIM card will consume some roaming data because changing SIM slots do not happen instantly.

Multi Protocol Label Switching (MPLS)

In This Section:

Mpls Overview

Overview

MPLS stands for MultiProtocol Label Switching. It kind of replaces IP routing - packet forwarding decision (outgoing interface and next-hop router) is no longer based on fields in IP header (usually destination address) and routing table, but on labels that are attached to packet. This approach speeds up the forwarding process because next-hop lookup becomes very simple compared to routing lookup (finding the longest matching prefix).

The efficiency of the forwarding process is the main benefit of MPLS, but it must be taken into account that MPLS forwarding disables the processing of network layer (e.g. IP) headers, therefore no network layer-based actions like NAT and filtering can be applied to MPLS forwarded packets. Any network-layer-based actions should be taken on ingress or egress of MPLS cloud, with the preferred way being ingress - this way, e.g. traffic that is going to be dropped anyway does not travel through the MPLS backbone.

In the simplest form, MPLS can be thought of as improved routing - labels are distributed by means of LDP protocol for routes that are active and a labeled packet takes the same path it would take if it was not labeled. A router that routes unlabeled packets using some route for which it has received a label from the next hop, imposes a label on the packet, and sends it to the next hop - gets MPLS switched further along its path. A router that receives a packet with a label it has assigned to some route changes the packet label with one received from the next hop of a particular route and sends a packet to the next hop. Label switched path ensures delivery of data to the MPLS cloud egress point. Applications of MPLS are based on this basic MPLS concept of label switched paths.

Another way of establishing label switching paths is traffic engineering tunnels (TE tunnels) by means of the RSVP-TE protocol. Traffic engineering tunnels allow explicitly routed LSPs and constraint-based path selection (where constraints are interface properties and available bandwidth).

Taking into account the complexity, new protocols, and applications that MPLS introduces and the differences of concepts that MPLS adds to routed /bridged networks, it is recommended to have an in-depth understanding of MPLS concepts before implementing MPLS in a production network. Some suggested reading material:

- Multiprotocol Label Switching http://en.wikipedia.org/wiki/Multiprotocol_Label_Switching
- RFC3031 Multiprotocol Label Switching Architecture http://www.ietf.org/rfc/rfc3031.txt
- MPLS Fundamentals by Luc De Ghein http://www.amazon.com/MPLS-Fundamentals-Luc-Ghein/dp/1587051974

Feature is not supported on SMIPS devices (hAP lite, hAP lite TC and hAP mini).

Supported Features

Currently, RouterOS supports the following MPLS related features:

- MPLS switching with penultimate hop popping support
- static local label bindings for IPv4 and IPv6
- static remote label bindings for IPv4 and IPv6
- Label Distribution Protocol (RFC 3036, RFC 5036, and RFC 7552) for IPv4 and IPv6
  - downstream unsolicited label advertisement
  - independent label distribution control
  - liberal label retention
  - targeted session establishment
  - optional loop detection
  - ECMP support
- Virtual Private Lan Service
  - VPLS LDP signaling (RFC 4762)
  - Cisco style static VPLS pseudowires (RFC 4447 FEC type 0x80)
  - VPLS pseudowire fragmentation and reassembly (RFC 4623)
  - VPLS MP-BGP based autodiscovery and signaling (RFC 4761)
  - Cisco VPLS BGP-based auto-discovery (draft-ietf-l2vpn-signaling-08)
  - support for multiple import/export route-target extended communities for BGP based VPLS (both, RFC 4761 and draft-ietf-l2vpn-signaling-08)
- RSVP-TE Tunnels
  - tunnel head-end

    - explicit paths
    - OSPF extensions for TE tunnels
    - CSPF path selection
    - forwarding of VPLS and MPLS IP VPN traffic on TE tunnels
    - Ingress TE tunnel rate limit and automatic reserved bandwidth adjustment, see TE Tunnel Bandwidth Control
    - all tunnel bandwidth settings are specified and displayed in bits per second
- MP-BGP based MPLS IP VPN
- Per-prefix and per-vrf label distribution policies for MP-BGP based MPLS VPN
- OSPF extensions for MPLS TE
- support for OSPF as CE-PE protocol
- ping and traceroute for specified VRF
- control over network-layer TTL propagation in MPLS
- RIP as CE-PE protocol
- per-VRF BGP instance redistribution settings

MPLS features that RouterOS DOES NOT HAVE yet:

- LDP features:
    - downstream on-demand label advertisement
    - ordered label distribution control
    - conservative label retention
- TE features
    - fast-reroute
    - link/node protection
- Support for BGP as label distribution protocol

MPLS MTU, Forwarding and Label Bindings

Label Range and TTL

From the `/mpls settings` menu it is possible to assign specific dynamic label range and TTL propagation. If for some reason static label mapping is used then the dynamic range can be adjusted to exclude statically assigned label numbers from being dynamically assigned by any of the label distribution protocols.

Property	Description
dynamic-label-range (*range of integer[16..1048575]*; Default: **16-1048575**)	Range of Label numbers used for dynamic allocation. The first 16 labels are reserved for special purposes (as defined in RFC). If you intend to configure labels statically then adjust the dynamic default range not to include numbers that will be used in a static configuration.
propagate-ttl (*yes* / *no*; Default: **yes**)	Whether to copy TTL values from IP header to MPLS header. If this option is set to **no** then hops inside the MPLS cloud will be invisible from traceroutes.
allow-fast-path(*yes* / *no*; Default: **yes)**	Enable/disable MPLS fast-path support.

MPLS MTU

Configuration of MPLS MTU (path MTU + MPLS tag size) is useful in cases when there is a large variety of possible MTUs along the path. Configuring MPLS MTU to a minimum value that can pass all the hops will ensure that the MPLS packet will not be silently dropped on the devices that do not support big enough MTU.

MPLS MTUs are configured from the `/mpls interface` menu.

```
[admin@rack1_b35_CCR1036] /mpls/interface> print
Flags: X - disabled; * - builtin
 0    ;;; router-test
      interface=ether1 mpls-mtu=1580 input=yes

 1    ;;; router-test
      interface=ether2 mpls-mtu=1580 input=yes

 2    interface=all mpls-mtu=1500
```

Properties

Property	Description
comment (*string*; Default:)	Short description of the interface
disabled (*yes* / *no*; Default: **no**)	If set to **yes** then this configuration is ignored.
interface (*name*; Default:)	Name of the interface or interface-list to match.
input (*yes* / *no*; Default: **yes**)	Whether to allow MPLS input on the interface
mpls-mtu (*integer [512..65535]*; Default: **1508**)	The option represents how big packets can be carried over the interface with added MPLS labels.

⚠ Listed entries are ordered, and the first entry (iterating from the top to the bottom) that matches the interface will be used.

The order of the entries is important due to the possibility that different interface lists can contain the same interface and in addition, that interface can be referenced directly.

Selection of the MPLS MTU happens in the following manner:

- If the interface matched the entry from this table, then try to use configured MPLS MTU value
- If the interface does not match any entry then consider MPLS MTU equal to L2MTU
- If the interface does not support L2MTU, then consider MPLS MTU equal to L3 MTU

On the MPLS ingress path, MTU is chosen by min(MPLS MTU - tagsize, l3mtu). This means that on interfaces that do not support L2MTU and default L3 MTU is set to 1500, max path MTU will be 1500 - tag size (the interface will not be able to pass full IP frame without fragmentation). In such scenarios, L3MTU must be increased by max observed tag size.

Read more on MTUs in the MTU in RouterOS article.

Forwarding Table

Entries in the `/mpls forwarding-table` menu show label bindings for specific routes that will be used in MPLS label switching. Properties in this menu are read-only.

```
[admin@rack1_b35_CCR1036] /mpls/forwarding-table> print
Flags: L, V - VPLS
Columns: LABEL, VRF, PREFIX, NEXTHOPS
#   LABEL  VRF   PREFIX         NEXTHOPS
0 L    16  main  10.0.0.0/8     { nh=10.155.130.1; interface=ether12 }
1 L    18  main  111.111.111.3  { label=impl-null; nh=111.12.0.1; interface=ether2 }
2 L    17  main  111.111.111.2  { label=impl-null; nh=111.11.0.1; interface=ether1 }
```

Property	Description
prefix (*IP/Mask*)	Destination prefix for which labels are assigned
label (*integer*)	Ingress MPLS label
ldp (*yes* | *no*)	Whether labels are LDP signaled
nexthops ()	An array of the next-hops, each entry in the array represents one ECMP next-hop. Array entry can contain several parameters: • **label** - egress MPLS label • **nh** - out next-hop IP address • **interface** - out the interface
out-label (*integer*)	Label number which is added or switched to for outgoing packet.
packets (*integer*)	Number of packets matched by this entry
te-sender	
te-session	
traffic-eng	Shows whether the entry is signaled by RSVP-TE (Traffic Engineering)
type *(string)*	Type of the entry, for example, "vpls", etc.
vpls (*yes* | *no*)	Shows whether the entry is used for VPLS tunnels.
vpn	
vrf	Name of the VRF table this entry belongs to

EXP bit behaviour

Overview

When the MPLS label is attached to the packet, it increases packet length by 32 bits (4 bytes). These 32 bits are broken down as follows:

- label value itself (20 bits)
- EXP ("experimental") field (3 bits)
- time to live field (8 bits)
- bottom of stack field (1 bit)

The use of "experimental" bits is not specified by MPLS standards, but the most common use is to carry QoS information, similar to 802.1q priority in the VLAN tag. Note that the EXP field is 3 bits only therefore it can carry values from 0 to 7 only, which allows having 8 traffic classes.

EXP field treatment in RouterOS

When RouterOS receives an MPLS packet, it sets the "ingress priority" value for the packet to that carried inside the top label. Note that "ingress priority" **is not** a field inside packet headers - it can be thought of as an additional mark assigned to a packet while being processed by the router. When RouterOS labels an MPLS packet, it sets EXP bits to "priority" (not "ingress priority"!) assigned to the packet. When RouterOS switches MPLS packet, "ingress priority" is automatically copied to "priority", this way regular MPLS switching communicates priority info over the whole label switched path.

Additional info on "ingress priority" and "priority" handling is also in WMM and VLAN priority.

Therefore what happens to the EXP field depends based on what action is taken on the packet:

- if the packet is MPLS switched (by popping the label off the packet and pushing on the new one), the EXP field in the new label will be the same as in the received label, because:
  - RouterOS sets "ingress priority" to EXP bits in the received label
  - Switching automatically sets "priority" to "ingress priority"
  - RouterOS labels the packet with a new label and sets its EXP bits to value in "priority".
- if the packet is MPLS switched by using penultimate-hop-popping (the received label is popped off and no new one is pushed on), the EXP field of received priority stays in the "priority" field of the packet and may be used by some other MAC protocol, e.g. WMM or 802.1q VLAN, for example:
  - RouterOS sets "ingress priority" to EXP bits in the received label
  - Switching automatically sets "priority" to "ingress priority"
  - RouterOS switches the packet to the next hop (without pushing on the label) and that happens over the VLAN interface
  - VLAN interface sets 802.1q priority in the VLAN header to the "priority" value of the packet.

Note that penultimate-hop-popping can therefore lose QoS information carried over label switched path at the last hop. In cases where this is not desirable, penultimate-hop-popping behavior should be disabled by using the Explicit NULL label instead of the Implicit NULL label for the last hop in the label switched path. Using an Explicit NULL label for the last hop is the default behavior for MPLS TE tunnels.

- if a packet is supposed to be sent over label switched path (the first label will get pushed on the packet), EXP bits will be set to value in "priority", which in turn can be set up properly using firewall rules or other means (e.g. from DSCP field in IP header)
- if a packet is received for local processing, "ingress priority" is set to the EXP field of the received packet and can therefore be used to update the DSCP field of the packet or set "priority" from "ingress priority" using firewall rules

LDP

Overview

MikroTik RouterOS implements Label Distribution Protocol (RFC 3036, RFC 5036, and RFC 7552) for IPv4 and IPv6 address families. LDP is a protocol that performs the set of procedures and exchange messages by which Label Switched Routers (LSRs) establish Label Switched Paths (LSPs) through a network by mapping network-layer routing information directly to data-link layer switched paths.

Prerequisites for MPLS

"Loopback" IP address

Although not a strict requirement, it is advisable to configure routers participating in the MPLS network with "loopback" IP addresses (not attached to any real network interface) to be used by LDP to establish sessions.

This serves 2 purposes:

- as there is only one LDP session between any 2 routers, no matter how many links connect them, the loopback IP address ensures that the LDP session is not affected by interface state or address changes
- use of loopback address as LDP transport address ensures proper penultimate hop popping behavior when multiple labels are attached to the packet as in the case of VPLS

In RouterOS "loopback" IP address can be configured by creating a dummy bridge interface without any ports and adding the address to it. For example:

```
/interface bridge add name=lo
/ip address add address=255.255.255.1/32 interface=lo
```

IP connectivity

As LDP distributes labels for active routes, the essential requirement is properly configured IP routing. LDP by default distributes labels for active IGP routes (that is - connected, static, and routing protocol learned routes, except BGP).

For instructions on how to set up properly IGP refer to appropriate documentation sections:

- OSPF
- Static Routing
- etc

LDP supports ECMP routes.

You should be able to reach any loopback address from any location of your network before continuing with the LDP configuration. Connectivity can be verified with the ping tool running from loopback address to loopback address.

Example Setup

Let's consider that we have already existing four routers setup, with working IP connectivity.

```
   (lo:111.111.111.1)       (lo:111.111.111.2)          (lo:111.111.111.3)         (lo:111.111.111.4)
|---------R1-----(111.11.0.0/24)-----R2-----(111.12.0.0/24)-----R3-----(111.13.0.0/24)-----R4---------|
```

Ip Reachability

Not going deep into routing setup here is the quit export of the IP and OSPF configurations:

```
#R1
/interface bridge
add name=loopback
/ip address
add address=111.11.0.1/24 interface=ether2
add address=111.111.111.1 interface=loopback

/routing ospf instance
add name=default_ip4 router-id=111.111.111.1
/routing ospf area
add instance=default_ip4 name=backbone_ip4
/routing ospf interface-template
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.111.111.1
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.11.0.0/24


#R2
/interface bridge
add name=loopback
/ip address
add address=111.11.0.2/24 interface=ether2
add address=111.12.0.1/24 interface=ether3
add address=111.111.111.2 interface=loopback

/routing ospf instance
add name=default_ip4 router-id=111.111.111.2
/routing ospf area
add instance=default_ip4 name=backbone_ip4
/routing ospf interface-template
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.111.111.2
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.11.0.0/24
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.12.0.0/24


#R3
/interface bridge
add name=loopback

/ip address
add address=111.12.0.2/24 interface=ether2
add address=111.13.0.1/24 interface=ether3
add address=111.111.111.3 interface=loopback

/routing ospf instance
add name=default_ip4 router-id=111.111.111.3
/routing ospf area
add instance=default_ip4 name=backbone_ip4
/routing ospf interface-template
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.111.111.3
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.12.0.0/24
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.13.0.0/24


#R4
/interface bridge
add name=loopback
/ip address
add address=111.13.0.2/24 interface=ether2
add address=111.111.111.4 interface=loopback

/routing ospf instance
add name=default_ip4 router-id=111.111.111.4
/routing ospf area
add instance=default_ip4 name=backbone_ip4
/routing ospf interface-template
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.111.111.4
add area=backbone_ip4 dead-interval=10s hello-interval=1s networks=111.13.0.0/24
```

Verify that IP connectivity and routing are working properly

```
[admin@R4] /ip/address> /tool traceroute 111.111.111.1 src-address=111.111.111.4
Columns: ADDRESS, LOSS, SENT, LAST, AVG, BEST, WORST, STD-DEV
#  ADDRESS        LOSS  SENT  LAST   AVG  BEST  WORST  STD-DEV
1  111.13.0.1     0%       4  0.6ms  0.6  0.6   0.6    0
2  111.12.0.1     0%       4  0.5ms  0.6  0.5   0.6    0.1
3  111.111.111.1  0%       4  0.6ms  0.6  0.6   0.6    0
```

LDP Setup

In order to start distributing labels, LDP is enabled on interfaces that connect other LDP routers and not enabled on interfaces that connect customer networks.

On R1 it will look like this:

```
/mpls ldp
add afi=ip lsr-id=111.111.111.1 transport-addresses=111.111.111.1
/mpls ldp interface
add interface=ether2
```

Note that the transport address gets set to 111.111.111.1. This makes the router originate LDP session connections with this address and also advertise this address as a transport address to LDP neighbors.

Other routers are set up similarly.

R2:

```
/mpls ldp
add afi=ip lsr-id=111.111.111.2 transport-addresses=111.111.111.2
/mpls ldp interface
add interface=ether2
add interface=ether3
```

On R3:

```
/mpls ldp
add afi=ip lsr-id=111.111.111.3 transport-addresses=111.111.111.3
/mpls ldp interface
add interface=ether2
add interface=ether3
```

On R4:

```
/mpls ldp
add afi=ip lsr-id=111.111.111.4 transport-addresses=111.111.111.4
/mpls ldp interface
add interface=ether2
```

After LDP sessions are established, R2 should have two LDP neighbors:

```
[admin@R2] /mpls/ldp/neighbor> print
Flags: D, I - INACTIVE; O, T - THROTTLED; p - PASSIVE
Columns: TRANSPORT, LOCAL-TRANSPORT, PEER, ADDRESSES
#     TRANSPORT      LOCAL-TRANSPORT  PEER             ADDRESSES
0 DO  111.111.111.1  111.111.111.2    111.111.111.1:0  111.11.0.1
                                                       111.111.111.1
1 DOp 111.111.111.3  111.111.111.2    111.111.111.3:0  111.12.0.2
                                                       111.13.0.1
                                                       111.111.111.3
```

The local mappings table shows what label is assigned to what route and peers the router have distributed labels to.

```
[admin@R2] /mpls/ldp/local-mapping> print
Flags: I - INACTIVE; D - DYNAMIC; E - EGRESS; G - GATEWAY; L - LOCAL
Columns: VRF, DST-ADDRESS, LABEL, PEERS
#       VRF   DST-ADDRESS      LABEL      PEERS
0  D G  main  10.0.0.0/8       16         111.111.111.1:0
                                          111.111.111.3:0
1 IDE L main  10.155.130.0/25  impl-null  111.111.111.1:0
                                          111.111.111.3:0
2 IDE L main  111.11.0.0/24    impl-null  111.111.111.1:0
                                          111.111.111.3:0
3 IDE L main  111.12.0.0/24    impl-null  111.111.111.1:0
                                          111.111.111.3:0
4 IDE L main  111.111.111.2    impl-null  111.111.111.1:0
                                          111.111.111.3:0
5  D G  main  111.111.111.1    17         111.111.111.1:0
                                          111.111.111.3:0
6  D G  main  111.111.111.3    18         111.111.111.1:0
                                          111.111.111.3:0
7  D G  main  111.111.111.4    19         111.111.111.1:0
                                          111.111.111.3:0
8  D G  main  111.13.0.0/24    20         111.111.111.1:0
                                          111.111.111.3:0
```

Remote mappings table on the other hand shows labels that are allocated for routes by neighboring LDP routers and advertised to this router:

```
[admin@R2] /mpls/ldp/remote-mapping> print
Flags: I - INACTIVE; D - DYNAMIC
Columns: VRF, DST-ADDRESS, NEXTHOP, LABEL, PEER
 #    VRF   DST-ADDRESS      NEXTHOP     LABEL      PEER
 0 ID main  10.0.0.0/8                  16         111.111.111.1:0
 1 ID main  10.155.130.0/25             impl-null  111.111.111.1:0
 2 ID main  111.11.0.0/24               impl-null  111.111.111.1:0
 3 ID main  111.12.0.0/24               17         111.111.111.1:0
 4  D main  111.111.111.1    111.11.0.1  impl-null  111.111.111.1:0
 5 ID main  111.111.111.2               19         111.111.111.1:0
 6 ID main  111.111.111.3               20         111.111.111.1:0
 7 ID main  111.111.111.4               21         111.111.111.1:0
 8 ID main  111.13.0.0/24               18         111.111.111.1:0
 9 ID main  0.0.0.0/0                   impl-null  111.111.111.3:0
10 ID main  111.111.111.2               16         111.111.111.3:0
11 ID main  111.111.111.1               18         111.111.111.3:0
12  D main  111.111.111.3    111.12.0.2  impl-null  111.111.111.3:0
13  D main  111.111.111.4    111.12.0.2  19         111.111.111.3:0
14 ID main  10.155.130.0/25             impl-null  111.111.111.3:0
15 ID main  111.11.0.0/24               17         111.111.111.3:0
16 ID main  111.12.0.0/24               impl-null  111.111.111.3:0
17  D main  111.13.0.0/24    111.12.0.2  impl-null  111.111.111.3:0
```

We can observe that router has received label bindings for all routes from both its neighbors - R1 and R3.

The remote mapping table will have active mappings only for the destinations that have direct next-hop, for example, let's take a closer look at 111.111.111.4 mappings. The routing table indicates that the network 111.111.111.4 is reachable via 111.12.0.2 (R3):

```
[admin@R2] /ip/route> print where dst-address=111.111.111.4
Flags: D - DYNAMIC; A - ACTIVE; o, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
    DST-ADDRESS       GATEWAY            DISTANCE
DAo 111.111.111.4/32  111.12.0.2%ether3       110
```

And if we look again at the remote mapping table, the only active mapping is the one received from R3 with assigned label 19. This implies that when R2 when routing traffic to this network, will impose label 19.

```
17  D main  111.111.111.4    111.12.0.2  19         111.111.111.3:0
```

Label switching rules can be seen in the forwarding table:

```
[admin@R2] /mpls/forwarding-table> print
Flags: L, V - VPLS
Columns: LABEL, VRF, PREFIX, NEXTHOPS
#   LABEL  VRF   PREFIX         NEXTHOPS
0 L    16  main  10.0.0.0/8     { nh=10.155.130.1; interface=ether1 }
1 L    18  main  111.111.111.3  { label=impl-null; nh=111.12.0.2; interface=ether3 }
2 L    19  main  111.111.111.4  { label=19; nh=111.12.0.2; interface=ether3 }
3 L    20  main  111.13.0.0/24  { label=impl-null; nh=111.12.0.2; interface=ether3 }
4 L    17  main  111.111.111.1  { label=impl-null; nh=111.11.0.1; interface=ether2 }
```

If we take a look at rule number 2, the rule says that when R2 received the packet with label 19, it will change the label to new label 19 (assigned by the R3).

As you can see from this example it is not mandatory that labels along the path should be unique.

Now if we look at the forwarding table of R3:

```
[admin@R3] /mpls/forwarding-table> print
Flags: L, V - VPLS
Columns: LABEL, VRF, PREFIX, NEXTHOPS
#   LA  VRF   PREFIX         NEXTHOPS
0 L 19  main  111.111.111.4  { label=impl-null; nh=111.13.0.2; interface=ether3 }
1 L 17  main  111.11.0.0/24  { label=impl-null; nh=111.12.0.1; interface=ether2 }
2 L 16  main  111.111.111.2  { label=impl-null; nh=111.12.0.1; interface=ether2 }
3 L 18  main  111.111.111.1  { label=17; nh=111.12.0.1; interface=ether2 }
```

Rule number 0, shows that the out label is "**impl-null**". The reason for this is that R3 is the last hop before 111.111.111.4 will be reachable and there is no need to swap to any real label. It is known that R4 is the egress point for the 111.111.111.4 network (router is the egress point for directly connected networks because the next hop for traffic is not MPLS router), therefore it advertises the "implicit null" label for this route. This tells R3 to forward traffic for the destination 111.111.111.4/32 to R4 unlabelled, which is exactly what R3 forwarding table entry tells.

> ⚠ Action, when the label is not swapped to any real label, is called **Penultimate hop popping,** it ensures that routers do not have to do unnecessary label lookup when it is known in advance that the router will have to route the packet.

Using traceroute in MPLS networks

RFC4950 introduces extensions to the ICMP protocol for MPLS. The basic idea is that some ICMP messages may carry an MPLS "label stack object" (a list of labels that were on the packet when it caused a particular ICMP message). ICMP messages of interest for MPLS are Time Exceeded and Need Fragment.

MPLS label carries not only label value, but also TTL field. When imposing a label on an IP packet, MPLS TTL is set to value in the IP header, when the last label is removed from the IP packet, IP TTL is set to value in MPLS TTL. Therefore MPLS switching network can be diagnosed by means of a traceroute tool that supports MPLS extension.

For example, the traceroute from R4 to R1 looks like this:

```
[admin@R1] /mpls/ldp/neighbor> /tool traceroute 111.111.111.4 src-address=111.111.111.1
Columns: ADDRESS, LOSS, SENT, LAST, AVG, BEST, WORST, STD-DEV, STATUS
#  ADDRESS         LOSS  SENT  LAST   AVG  BEST  WORST  STD-DEV  STATUS
1  111.11.0.2      0%       2  0.7ms  0.7  0.7   0.7          0  <MPLS:L=19,E=0>
2  111.12.0.2      0%       2  0.4ms  0.4  0.4   0.4          0  <MPLS:L=19,E=0>
3  111.111.111.4   0%       2  0.5ms  0.5  0.5   0.5          0
```

Traceroute results show MPLS labels on the packet when it produced ICMP Time Exceeded. The above means: that when R3 received a packet with MPLS TTL 1, it had label 18 on it. This match advertised label by R3 for 111.111.111.4. In the same way, R2 observed label 17 on the packet on the next traceroute iteration - R3 switched label 17 to label 17, as explained above. R1 received packet without labels - R2 did penultimate hop popping as explained above.

Drawbacks of using traceroute in MPLS network

Label switching ICMP errors

One of the drawbacks of using traceroute in MPLS networks is the way MPLS handles produced ICMP errors. In IP networks ICMP errors are simply routed back to the source of the packet that caused the error. In an MPLS network, it is possible that a router that produces an error message does not even have a route to the source of the IP packet (for example in the case of asymmetric label switching paths or some kind of MPLS tunneling, e.g. to transport MPLS VPN traffic).

Due to this produced ICMP errors are not routed to the source of the packet that caused the error but switched further along the label switching path, assuming that when the label switching path endpoint will receive an ICMP error, it will know how to properly route it back to the source.

This causes the situation, that traceroute in MPLS network can not be used the same way as in IP network - to determine failure point in the network. If the label switched path is broken anywhere in the middle, no ICMP replies will come back, because they will not make it to the far endpoint of the label switching path.

Penultimate hop popping and traceroute source address

A thorough understanding of penultimate hop behavior and routing is necessary to understand and avoid problems that penultimate hop popping causes to traceroute.

In the example setup, a regular traceroute from R5 to R1 would yield the following results:

```
[admin@R5] > /tool traceroute 9.9.9.1
     ADDRESS                                  STATUS
   1         0.0.0.0 timeout timeout timeout
   2         2.2.2.2 37ms 4ms 4ms
                      mpls-label=17
   3         9.9.9.1 4ms 2ms 11ms
```

compared to:

```
[admin@R5] > /tool traceroute 9.9.9.1 src-address=9.9.9.5
     ADDRESS                                  STATUS
   1         4.4.4.3 15ms 5ms 5ms
                      mpls-label=17
   2         2.2.2.2 5ms 3ms 6ms
                      mpls-label=17
   3         9.9.9.1 6ms 3ms 3ms
```

The reason why the first traceroute does not get a response from R3 is that by default traceroute on R5 uses source address 4.4.4.5 for its probes because it is the preferred source for a route over which next-hop to 9.9.9.1/32 is reachable:

```
[admin@R5] > /ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        G GATEWAY         DISTANCE             INTERFACE
 ...
 3 ADC  4.4.4.0/24         4.4.4.5                           0                    ether1
 ...
 5 ADo  9.9.9.1/32                         r 4.4.4.3         110                  ether1
 ...
```

When the first traceroute probe is transmitted (source: 4.4.4.5, destination 9.9.9.1), R3 drops it and produces an ICMP error message (source 4.4.4.3 destination 4.4.4.5) that is switched all the way to R1. R1 then sends ICMP error back - it gets switched along the label switching path to 4.4.4.5.

R2 is the penultimate hop popping router for network 4.4.4.0/24 because 4.4.4.0/24 is directly connected to R3. Therefore R2 removes the last label and sends ICMP error to R3 unlabelled:

```
[admin@R2] > /mpls forwarding-table print
 # IN-LABEL             OUT-LABELS           DESTINATION        INTERFACE            NEXTHOP
 ...
 3 19                                        4.4.4.0/24         ether2               2.2.2.3
 ...
```

R3 drops the received IP packet because it receives a packet with its own address as a source address. ICMP errors produced by following probes come back correctly because R3 receives unlabelled packets with source addresses 2.2.2.2 and 9.9.9.1, which are acceptable to a route.

Command:

```
[admin@R5] > /tool traceroute 9.9.9.1 src-address=9.9.9.5
 ...
```

produces expected results, because the source address of traceroute probes is 9.9.9.5. When ICMP errors are traveling back from R1 to R5, the penultimate hop popping for the 9.9.9.5/32 network happens at R3, therefore it never gets to route packet with its own address as a source address.

Optimizing label distribution

Label binding filtering

During the implementation of the given example setup, it has become clear that not all label bindings are necessary. For example, there is no need to exchange IP route label bindings between R1 and R3 or R2 and R4, as there is no chance they will ever be used. Also, if the given network core is providing connectivity only for mentioned customer ethernet segments, there is no real use to distribute labels for networks that connect routers between themselves, the only routes that matter are /32 routes to endpoints or attached customer networks.

Label binding filtering can be used to distribute only specified sets of labels to reduce resource usage and network load.

There are 2 types of label binding filters:

- which label bindings should be advertised to LDP neighbors, configured in the `/mpls ldp advertise-filter` menu
- which label bindings should be accepted from LDP neighbors, configured in `/mpls ldp accept-filter` menu

Filters are organized in the ordered list, specifying prefixes that must include the prefix that is tested against the filter and neighbor (or wildcard).

In the given example setup all routers can be configured so that they advertise labels only for routes that allow reaching the endpoints of tunnels. For this 2 advertise filters need to be configured on all routers:

```
/mpls ldp advertise-filter add prefix=111.111.111.0/24 advertise=yes
/mpls ldp advertise-filter add prefix=0.0.0.0/0 advertise=no
```

This filter causes routers to advertise only bindings for routes that are included by the 111.111.111.0/24 prefix which covers loopbacks (111.111.111.1/32, 111.111.111.2/32, etc). The second rule is necessary because the default filter results when no rule matches are to allow the action in question.

In the given setup there is no need to set up accept filter because by convention introduced by 2 abovementioned rules no LDP router will distribute unnecessary bindings.

Note that filter changes do not affect existing mappings, so to take the filter into effect, connections between neighbors need to be reset. either by removing neighbors from the LDP neighbor table or by restarting the LDP instance.

So on R2, for example, we get:

```
[admin@R2] /mpls/ldp/remote-mapping> print
Flags: I - INACTIVE; D - DYNAMIC
Columns: VRF, DST-ADDRESS, NEXTHOP, LABEL, PEER
#    VRF   DST-ADDRESS    NEXTHOP     LABEL      PEER
0 ID main  111.111.111.2              17         111.111.111.3:0
1 ID main  111.111.111.1              16         111.111.111.3:0
2  D main  111.111.111.3  111.12.0.2  impl-null  111.111.111.3:0
3  D main  111.111.111.4  111.12.0.2  18         111.111.111.3:0
4 ID main  111.111.111.2              16         111.111.111.1:0
5  D main  111.111.111.1  111.11.0.1  impl-null  111.111.111.1:0
6 ID main  111.111.111.3              17         111.111.111.1:0
7 ID main  111.111.111.4              18         111.111.111.1:0
```

LDP on Ipv6 and Dual-Stack links

RouterOS implements RFC 7552 to support LDP on dual-stack links.

Supported AFIs can be selected by LDP instance, as well as explicitly configured per LDP interface.

```
/mpls ldp
add afi=ip,ipv6 lsr-id=111.111.111.1 preferred-afi=ipv6
/mpls ldp interface
add interface=ether2 afi=ip
add interface=ether3 afi=ipv6
```

The example above enables LDP instance to use IPv4 and IPv6 address families and sets the preference to IPv6 with `preferred-afi` parameter. LDP interface configuration on the other hand explicitly sets that **ether2** supports only IPv4 and **ether3** supports only IPv6.

The main question occurs how AFI is selected when there are a mix of different AFIs and what if one of the supported AFIs flaps.

The logic behind sending hellos is as follows:

- if an interface has only one AFI:
  - dual-stack element is not sent
  - sends hello only if there is an IP address on the interface from the corresponding AFI.
- If an interface has both AFIs:
  - dual-stack element is always sent and contains the value from preferred-afi
  - sends hellos on each AFI if a corresponding address is present on the interface.

From all received hellos peer determines which AFI to use for connection and for which AFIs to bind and send labels. For LDP to be able to use a specific AFI, receiving hello for that specific AFI is mandatory. Hello packet contains the transport address necessary for proper LDP operation. By comparing received AFI addresses, is determined active/passive role.

The logic behind receiving and processing hellos is as follows:

- if the LDP instance has only one AFI (it means that all interfaces can have only that specific AFI operational):
  - drop hellos from not supported AFI
  - ignore/forget the dual-stack element for the hello packet
  - the role is determined only for this one specific AFI
  - labels are sent only for this one specific AFI
- if the LDP instance has both AFIs (interfaces can have different combinations of supported AFIs):
  - drop hellos from AFI that are not configured as supported on the interface.
  - ignore/forget the dual-stack element (preference is not taken into account) for hello packets, if an interface has only one supported AFI.
  - drop hello if received preference in dual-stack element does not match configured `preferred-afi`.

If there are changes in hello packets, the existing session is terminated only in case if address family used by labels is changed, otherwise, the session is preserved.

Dual-stack element in hello packets is set only if an interface is determined to be dual-stack compatible:

- Normally such an interface should be able to receive hellos from both AFIs,
  - Before proceeding LDP should wait for hello from the preferred AFI.
  - if hello is received only from one AFI:
    - if hello from preferred AFI is not received then it is considered an error.
    - otherwise, wait for missing hello for x seconds (x = 3 * hello-interval)
      - if missing hello appears within a time interval consider peer to be dual-stack
      - if missing hello did not appear, then consider peer to be single-stack
      - if missing hello appeared after the time interval then restart the session.
- the dual-stack element indicates that LDP wants to distribute labels for both AFIs.

In summary, the following combinations of AFIs and dual-stack element (ds6) are possible assuming that preferred-afi=ipv6:

1. ipv4 - wait X seconds, if no changes, then use the IPv4 LDP session and distribute IPv4 labels
2. ipv4+ds6 - wait for IPv6 hello, dual-stack element indicates that there should be IPv6
3. ipv6 - wait X seconds, if no changes, then use the IPv6 LDP session and distribute IPv6 labels
4. ipv6+ds6 - use IPv6 LDP session and distribute IPv6 labels
5. ipv4,ipv6 - use IPv6 LDP session and distribute IPv4 and IPv6 labels
6. ipv4,ipv6+ds6 - use IPv6 LDP session and distribute IPv4 and IPv6 labels

Property Reference

LDP Instance

Sub-menu: `/mpls`

Properties

Property	Description
afi (*ip* / *ipv6*; Default: **)**	Determines supported address families by the instance.
comments (*string*; Default:)	Short description of the entry
disabled (*yes* / *no*; Default: **no**)	
distribute-for-default (*yes* / *no*; Default: no)	Defines whether to map label for the default route.
hop-limit (*integer[0..255]*; Default:)	Max hop limit used for loop detection. Works in combination with the **loop-detect** property.
loop-detect (*yes* / *no*; Default:)	Defines whether to run LSP loop detection. Will not work correctly if not enabled on all LSRs. Should be used only on non-TTL networks such as ATMs.
lsr-id (*IP*; Default:)	Unique label switching router's ID.
path-vector-limit (*IP*; Default:)	Max path vector limit used for loop detection. Works in combination with the **loop-detect** property.
preferred-afi (ip | ipv6; Default: **ipv6**)	Determining which address family connection is preferred. Value is also set in dual-stack element (if used).
transport-addresses (*IP*; Default:)	Specifies LDP session connections origin addresses and also advertises these addresses as transport addresses to LDP neighbors.
use-explicit-null (*yes* / *no*; Default: no)	Whether to distribute explicit-null label bindings.
vrf (*name*; Default: main**)**	Name of the VRF table this instance will operate on.

Interface

Sub-menu: `/mpls ldp interface`

Property	Description
afi (*ip* / *ipv6*; Default: **)**	Determines interface address family. Only AFIs that are configured as supported by the instance is taken into account. If the value is not explicitly specified then it is considered to be equal to the instance-supported AFIs.
accept-dynamic-neighbors (*yes* / *no*; Default:)	Defines whether to discover neighbors dynamically or use only statically configured in LDP neighbors menu
comments (*string*; Default:)	Short description of the entry
disabled (*yes* / *no*; Default: **no**)	
hello-interval (*string*; Default:)	The interval between hello packets that the router sends out on specified interface/s. The default value is 5s.
hold-time (*string*; Default:)	Specifies the interval after which a neighbor discovered on the interface is declared as not reachable. The default value is 15s.
interface (*string*; Default:)	Name of the interface or interface list where LDP will be listening.

transport-addresses (List of *IPs*; Default:)	Used transport addresses if differs from LDP Instance settings.

Neighbors

Sub-menu: `/mpls ldp neighbor`

List of discovered and statically configured LDP neighbors.

Properties

Property	Description
comments (*string*; Default:)	Short description of the entry
disabled (*yes* | *no*; Default: **no**)	
send-targeted (*yes* | *no*; Default:)	Specifies whether to try to send targeted hellos, used for targeted (not directly connected) LDP sessions.
transport (*IP*; Default:)	Remote transport address.

Read-only Properties

Property	Description
active-connect (*yes* | *no*)	
addresses (*list of IPs*)	List of discovered addresses on the neighbor
inactive (*yes* | *no*)	Whether binding is active and can be selected as a candidate for forwarding.
dynamic (*yes* | *no*)	Whether entry was dynamically added
local-transport (*IP*)	Selected local transport address.
on-demand (*yes* | *no*)	
operational (*yes* | *no*)	Indicates whether the peer is operational.
passive (*yes* | *no*)	Indicates whether the peer is in a passive role.
passive-wait (*yes* | *no*)	
path-vector-limit (*integer*)	
peer (*IP:integer*)	LSR-ID and label space of the neighbor
sending-targeted-hello(*yes* | *no*)	Whether targeted hellos are being sent to the neighbor.
throttled (*yes* | *no*)	
used-afi (*yes* | *no*)	Used transport AFI
vpls (*yes* | *no*)	Whether neighbor is used by VPLS tunnel

Accept Filter

Sub-menu: `/mpls ldp accept-filter`

List of label bindings that should be accepted from LDP neighbors.

Property	Description

accept (*yes* | *no*; Default: **no**)	Whether to accept label bindings from the neighbors for the specified prefix.
comments (*string*; Default:)	Short description of the entry
disabled (*yes* | *no*; Default: **no**)	
neighbor(*string*; Default:)	Neighbor to which this filter applies.
prefix (*IP/mask*; Default:)	Prefix to match.
vrf (name; Default:)	

Advertise Filter

Sub-menu: `/mpls ldp advertise-filter`

List of label bindings that should be advertised to LDP neighbors.

Property	Description
advertise (*yes* | *no*; Default: **no**)	Whether to advertise label bindings to the neighbors for the specified prefix.
comments (*string*; Default:)	Short description of the entry
disabled (*yes* | *no*; Default: **no**)	
neighbor(*string*; Default:)	Neighbor to which this filter applies.
prefix (*IP/mask*; Default:)	Prefix to match.
vrf (name; Default:)	

Local Mapping

Sub-menu: `/mpls local-mapping`

This sub-menu shows labels bound to the routes locally in the router. In this menu also static mappings can be configured if there is no intention to use LDP dynamically.

Properties

Property	Description
comments (*string*; Default:)	Short description of the entry
disabled (*yes* | *no*; Default: **no**)	
dst-address (*IP/Mask*; Default:)	Destination prefix the label is assigned to.
label (*integer[0..1048576]* | *alert* | *expl-null* | *expl-null6* | *impl-null* | *none*; Default:)	Label number assigned to destination.
vrf (*name*; Default: main**)**	Name of the VRF table this mapping belongs to.

Read-only Properties

Property	Description
adv-path ()	
inactive (*yes* | *no*)	Whether binding is active and can be selected as a candidate for forwarding.
dynamic (*yes* | *no*)	Whether entry was dynamically added

egress (*yes* | *no*)	
gateway (*yes* | *no*)	Whether the destination is reachable through the gateway.
local (*yes* | *no*)	Whether the destination is locally reachable on the router
peers (*IP:label_space*)	IP address and label space of the peer to which this entry was advertised.

Remote Mapping

Sub-menu: `/mpls remote-mapping`

Sub-menu shows label bindings for routes received from other routers. Static mapping can be configured if there is no intention to use LDP dynamically. This table is used to build Forwarding Table

Properties

Property	Description
comments (*string*; Default:)	Short description of the entry
disabled (*yes* | *no*; Default: **no**)	
dst-address (*IP/Mask*; Default:)	Destination prefix the label is assigned to.
label (*integer[0..1048576]* | *alert* | *expl-null* | *expl-null6* | *impl-null* | *none*; Default:)	Label number assigned to destination.
nexthop (*IP*; Default:**)**	
vrf (*name*; Default: main**)**	Name of the VRF table this mapping belongs to.

Read-only Properties

Property	Description
inactive (*yes* | *no*)	Whether binding is active and can be selected as a candidate for forwarding.
dynamic (*yes* | *no*)	Whether entry was dynamically added
path (*string*)	

VPLS

Summary

The virtual Private Lan Service (VPLS) interface can be considered a tunnel interface just like the EoIP interface. To achieve transparent ethernet segment forwarding between customer sites.

Negotiation of VPLS tunnels can be done by LDP protocol or MP-BGP - both endpoints of tunnel exchange labels they are going to use for the tunnel.

Data forwarding in the tunnel happens by imposing 2 labels on packets: tunnel label and transport label - a label that ensures traffic delivery to the other endpoint of the tunnel.

MikroTik RouterOS implements the following VPLS features:

- VPLS LDP signaling (RFC 4762)
- Cisco style static VPLS pseudowires (RFC 4447 FEC type 0x80)
- VPLS pseudowire fragmentation and reassembly (RFC 4623)
- VPLS MP-BGP based autodiscovery and signaling (RFC 4761)
- Cisco VPLS BGP-based auto-discovery (draft-ietf-l2vpn-signaling-08)
- support for multiple import/export route-target extended communities for BGP based VPLS (both, RFC 4761 and draft-ietf-l2vpn-signaling-08)

VPLS Prerequisities

For VPLS to be able to transport MPLS packets, one of the label distribution protocols should be already running on the backbone, it can be LDP, RSVP-TE, or static bindings.

Before moving forward, familiarize yourself with the prerequisites required for LDP and prerequisites for RSVP-TE.

In case, if BGP should be used as a VPLS discovery and signaling protocol, the backbone should be running iBGP preferably with route reflector/s.

Example Setup

Let's consider that we already have a working LDP setup from the LDP configuration example.

Routers R1, R3, and R4 have connected Customer A sites, and routers R1 and R3 have connected Customer B sites. Customers require transparent L2 connectivity between the sites.

Reference

General

Sub-menu: `/interface vpls`

List of all VPLS interfaces. This menu shows also dynamically created BGP-based VPLS interfaces.

Properties

Property	Description

arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	Address Resolution Protocol
arp-timeout (*time interval* | *auto*; Default: auto)	
bridge (*name*; Default:)	
bridge-cost (*integer [1..200000000]*; Default:)	Cost of the bridge port.
bridge-horizon (*none* | *integer*; Default: **none**)	If set to **none** bridge horizon will not be used.
cisco-static-id (*integer [0..4294967295]*; Default: **0**)	Cisco-style VPLS tunnel ID.
comment (*string*; Default:)	Short description of the item
disable-running-check (*yes* | *no*; Default: **no**)	Specifies whether to detect if an interface is running or not. If set to **no** interface will always have the `running` flag.
disabled (*yes* | *no*; Default: **yes**)	Defines whether an item is ignored or used. By default VPLS interface is disabled.
mac-address (*MAC*; Default:)	
mtu (*integer [32..65536]*; Default: **1500**)	
name (*string*; Default:)	Name of the interface
pw-l2mtu (*integer [0..65536]*; Default: **1500**)	L2MTU value advertised to a remote peer.
pw-type (*raw-ethernet* | *tagged-ethernet* | *vpls*; Default: **raw-ethernet**)	Pseudowire type.
peer (*IP*; Default:)	The IP address of the remote peer.
pw-control-word (*disabled* | *enabled* | *default*; Default: **default**)	Enables/disables Control Word usage. Default values for regular and cisco style VPLS tunnels differ. Cisco style by default has control word usage disabled. Read more in the VPLS Control Word article.
vpls-id (*AsNum* | *AsIp*; Default:)	A unique number that identifies the VPLS tunnel. Encoding is 2byte+4byte or 4byte+2byte number.

Read-only properties

Property	Description
cisco-bgp-signaled (*yes* | *no*)	
vpls (*string*)	name of the bgp-vpls instance used to create dynamic vpls interface
bgp-signaled	
bgp-vpls	
bgp-vpls-prfx	
dynamic (*yes* | *no*)	
l2mtu (integer)	
running (*yes* | *no*)	

Monitoring

Command `/interface vpls monitor [id]` will display the current VPLS interface status

For example:

```
[admin@10.0.11.23] /interface vpls> monitor vpls2
remote-label: 800000
local-label: 43
remote-status:
transport: 10.255.11.201/32
transport-nexthop: 10.0.11.201
imposed-labels: 800000
```

Available read-only properties:

Property	Description
imposed-label (*integer*)	VPLS imposed label
local-label (*integer*)	Local VPLS label
remote-group ()	
remote-label (*integer*)	Remote VPLS label
remote-status (*integer*)	
transport-nexthop (*IP prefix*)	Shows used transport address (typically Loopback address).
transport (*string*)	Name of the transport interface. Set if VPLS is running over the Traffic Engineering tunnel.

VPLS Control Word

Summary

VPLS allows remote sites to share an Ethernet broadcast domain by connecting sites through **pseudo-wires(PW)** tunnels over a packet switching network (PSN). Since VPLS encapsulation adds additional overhead, each interface in LSP should be able to transmit a large enough packet.

Each ethernet chipset has hardware limitations on the maximum packet size that it can transmit. Even now there are Ethernets that support only one Vlan tag, meaning that the maximum packet size without Ethernet header and checksum (L2MTU) is 1504 bytes. Obviously, it is not enough to forward VPLS encapsulated Ethernet frame without fragmentation (at least 1524 L2MTU support is required). See MTU in RouterOS for maximum supported L2MTUs on RouterBOADs.

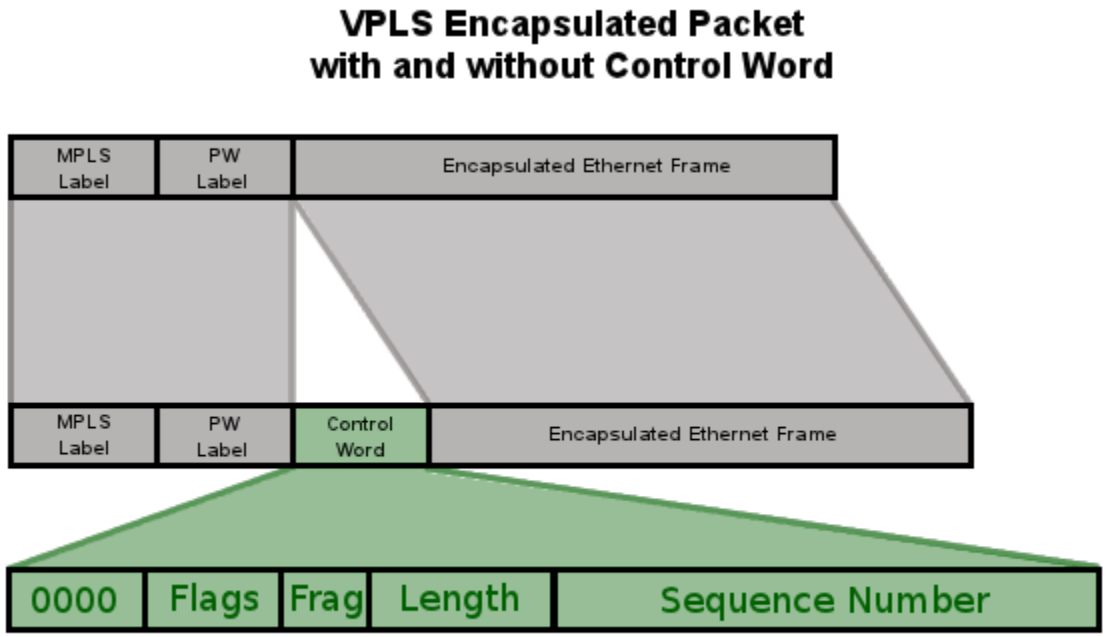
Since not even all RouterBOARDs support enough L2MTU to transmit VPLS encapsulated packet without fragmentation, RouterOS has added Pseudowire Fragmentation and Reassembly (PWE3) support according to RFC 4623 using 4-byte **Control Word (CW)**.

Control Word Usage

In RouterOS, Control Word is used for packet fragmentation and reassembly inside the VPLS tunnel and is done by utilizing optional **Control Word (CW)**. CW is added between PW label (demultiplexor) and packet payload and adds an additional 4-byte overhead.

> ⚠ Reordering OOO packets are not implemented, out of order fragments will be dropped

CW usage is controlled by the of `use-control-word` parameter in VPLS configuration.



As you can see **Control Word** is divided into 5 fields:

- 0000 - 4-bits identifies that the packet is PW (not IP)
- Flags - 4bits
- Frag - 2bits value that indicates payload fragmentation.
- Len - 6bits
- Seq - 16bits sequence number used to detect packet loss / misordering.

According to RFC generation and processing of sequence numbers is optional.

Traffic Eng

Properties

Sub-menu: `/interface traffic-eng`

Property	Description
affinity-exclude (*integer*; Default: **not set**)	Do not use interface if resource-class matches any of specified bits.
affinity-include-all (*integer*; Default: **not set**)	Use interface only if resource-class matches all of specified bits.
affinity-include-any (*integer*; Default: **not set**)	Use interface if resource-class matches any of specified bits.
auto-bandwidth-avg-interval (*time*; Default: **5m**)	Interval in which actual amount of data is measured, from which average bandwidth is calculated.
auto-bandwidth-range (*Disabled | Min [bps][-Max[bps]]*; Default: **0bps**)	Auto bandwidth adjustment range. `Read more >>`
auto-bandwidth-reserve (*integer[%]*; Default: **0%**)	Specifies percentage of additional bandwidth to reserve. `Read more >>`
auto-bandwidth-update-interval (*time*; Default: **1h**)	Interval during which tunnel keeps track of highest average rate.
bandwidth (*integer[bps]*; Default: **0bps**)	How much bandwidth to reserve for TE tunnel. Value is in bits per second. `Read more >>`
bandwidth-limit (*disabled | integer[%]*; Default: **disabled**)	Defines actual bandwidth limitation of TE tunnel. Limit is configured in percent of specified tunnel `bandwidth`. `Read more >>`
comment (*string*; Default:)	Short description of the item
disable-running-check (*yes | no*; Default: **no**)	Specifies whether to detect if interface is running or not. If set to **no** interface will always have `running` flag.
disabled (*yes | no*; Default: **yes**)	Defines whether item is ignored or used.
from-address (*auto | IP*; Default: **auto**)	Ingress address of the tunnel. If set to **auto** least IP address is picked.
holding-priority (*integer [0..7]*; Default: **not set**)	Is used to decide whether this session can be preempted by another session. 0 sets the highest priority.
mtu (*integer*; Default: **1500**)	Layer3 Maximum Transmission Unit
name (*string*; Default:)	Name of the interface
primary-path (*string*; Default:)	Primary label switching paths defined in `/mpls traffic-eng tunnel-path` menu.
primary-retry-interval (*time*; Default: **1m**)	Interval after which tunnel will try to use primary path.
record-route (*yes | no*; Default: **not set**)	If enabled, the sender node will receive information about the actual route that the LSP tunnel traverses. Record Route is analogous to a path vector, and hence can be used for loop detection.
reoptimize-interval (*time*; Default: **not set**)	Interval after which tunnel will re-optimize current path. If current path is not the best path then after optimization best path will be used. `Read more >>`
secondary-paths (*string[,string]*; Default:)	List of label switching paths used by TE tunnel if primary path fails. Paths are defined in `/mpls traffic-eng tunnel-path` menu.

setup-priority (*integer[0..7]*; Default: **not set**)	Parameter is used to decide whether this session can preempt another session. 0 sets the highest priority.
to-address (*IP*; Default: **0.0.0.0**)	Remote end of TE tunnel.

Monitoring

To verify TE tunnel's status **`monitor`** command can be used.

```
/interface traffic-eng monitor 0
tunnel-id: 12
primary-path-state: on-hold
secondary-path-state: established
secondary-path: static
active-path: static
active-lspid: 3
active-label: 66
explicit-route: "S:192.168.55.10/32,L:192.168.55.13/32,L:192.168.55.17/32"
recorded-route: "192.168.55.13[66],192.168.55.17[59],192.168.55.18[3]"
reserved-bandwidth: 5.0Mbps
```

Reoptimization

Path can be re-optimized manually by entering the command `/interface traffic-eng reoptimize [id]` (where [id] is an item number or interface name). It allows network administrators to reoptimize the LSPs that have been established based on changes in bandwidth, traffic, management policy, or other factors.

Let's say TE tunnel chose another path after a link failure on best path. You can verify optimization by looking at **`explicit-route`** or **`recorded-route`** values if record-route parameter is enabled.

```
/interface traffic-eng monitor 0
tunnel-id: 12
primary-path-state: established
primary-path: dyn
secondary-path-state: not-necessary
active-path: dyn active-lspid: 1
active-label: 67
explicit-route: "S:192.168.55.10/32,S:192.168.55.13/32,S:192.168.55.14/32,
S:192.168.55.17/32,S:192.168.55.18/32"
recorded-route: "192.168.55.13[67],192.168.55.17[60],192.168.55.18[3]"
reserved-bandwidth: 5.0Mbps
```

Whenever the link comes back, TE tunnel will use the same path even it is not the best path (unless reoptimize-interval is configured). To fix it we can manually reoptimize the tunnel path.

```
/interface traffic-eng reoptimize 0
```

```
/interface traffic-eng monitor 0
tunnel-id: 12
primary-path-state: established
primary-path: dyn
secondary-path-state: not-necessary
active-path: dyn
active-lspid: 2 active-label: 81
explicit-route: "S:192.168.55.5/32,S:192.168.55.2/32,S:192.168.55.1/32"
recorded-route: "192.168.55.2[81],192.168.55.1[3]"
reserved-bandwidth: 5.0Mbps
```

Notice how explicit-route and recorded-route changed to a shorter path.

MPLS Reference

MPLS Case Studies

Network Management

In This Section:

ARP

Summary

Sub-menu: `/ip arp`

Even though IP packets are addressed using IP addresses, hardware addresses must be used to actually transport data from one host to another. Address Resolution Protocol is used to map OSI level 3 IP addresses to OSI level 2 MAC addresses. A router has a table of currently used ARP entries. Normally the table is built dynamically, but to increase network security, it can be partially or completely built statically by means of adding static entries.

Properties

This section describes the ARP table configuration options.

Property	Description
address (*IP*; Default:)	IP address to be mapped
interface (*string* ; Default:)	Interface name the IP address is assigned to
mac-address (*MAC*; Default: **00:00:00:00:00:00**)	MAC address to be mapped to
published (*yes* | *no*; Default: **no**)	Static proxy-arp entry for individual IP addresses. When an ARP query is received for the specific IP address, the device will respond with its own MAC address. No need to set proxy-arp on the interface itself for all the MAC addresses to be proxied. The interface will respond to an ARP request only when the device has an active route towards the destination

Read-only properties:

Property	Description
complete (*yes* | *no*)	Complete flag is included in ARP entries when the ARP `status` is permanent, reachable, stale, probe, or delay
dhcp (*yes* | *no*)	Whether the ARP entry is added by DHCP server
disabled (*yes* | *no*)	Whether the ARP entry is disabled
dynamic (*yes* | *no*)	Whether the entry is dynamically created
invalid (*yes* | *no*)	Whether the entry is not valid

status (*delay* | *failed* | *incomplete* | *permanent* | *probe* | *reachable* | *stale*)	Shows the ARP entry state: • `delay` - neighbor entry validation is currently delayed • `failed` - ARP resolution has failed, the system was not able to obtain the MAC address for the given IP address • `incomplete` - system does not have the MAC address information for the IP address, it has not yet been resolved • `permanent` - ARP entry is considered permanent and will not be removed from the table, even if it is not actively being used. This is set for manually configured ARP entries • `probe` - neighbor is being probed • `reachable` - ARP resolution is successful, and the MAC address associated with the IP address is know, the entry is valid until the reachability timeout expires • `stale` - entry is still valid, but it is aged. This means that the system has not recently communicated with the device associated with the IP address.

⚠ The default maximum number of ARP entries depends on the installed amount of RAM. It can be adjusted with the command "/ip settings set max-neighbor-entries=x", see more details on IPv4 Settings.

ARP Modes

It is possible to set several ARP modes on the interface configuration.

- `disabled` - the interface will not use ARP
- `enabled` - the interface will use ARP
- `local-proxy-arp` - the router performs proxy ARP on the interface and sends replies to the same interface
- `proxy-arp` - the router performs proxy ARP on the interface and sends replies to other interfaces
- `reply-only` - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the IP/ARP table. No dynamic entries will be automatically stored in the IP/ARP table. Therefore for communications to be successful, a valid static entry must already exist.

Disabled

If the ARP feature is turned off on the interface, i.e., arp=disabled is used, ARP requests from clients are not answered by the router. Therefore, a static ARP entry should be added to the clients as well. For example, the router's IP and MAC addresses should be added to the Windows workstations using the arp command:

```
C:\> arp -s 10.5.8.254  00-aa-00-62-c6-09
```

Enabled

This mode is enabled by default on all interfaces. ARPs will be discovered automatically and new dynamic entries will be added to the ARP table.

Proxy ARP

A router with a properly configured proxy ARP feature acts as a transparent ARP proxy between different networks.

This behavior can be useful, for example, if you want to assign dial-in (ppp, pppoe, pptp) clients' IP addresses from the same address space as used on the connected LAN.

Proxy ARP can be enabled on each interface individually with command **arp=proxy-arp**:

Setup proxy ARP:

```
[admin@MikroTik] /interface ethernet> set 1 arp=proxy-arp

[admin@MikroTik] /interface ethernet> print

Flags: X - disabled, R - running
  #    NAME                MTU   MAC-ADDRESS         ARP
  0  R ether1              1500  00:30:4F:0B:7B:C1 enabled
  1  R ether2              1500  00:30:4F:06:62:12 proxy-arp
```

Reply Only

If ARP property is set to `reply-only` on the interface, then the router only replies to ARP requests. Neighbour MAC addresses will be resolved only using statically configured entries from the "/ip arp" menu, but there will be no need to add the router's MAC address to other hosts' ARP tables like in case if ARP is disabled.

Local Proxy Arp

If the ARP property is set to `local-proxy-arp` on an interface, then the router performs proxy ARP to/from this interface only. i.e. for traffic that comes in and goes out of the same interface. In a normal LAN, the default behavior is for two network hosts to communicate directly with each other, without involving the router.

With `local-proxy-arp` enabled, the router will respond to all client hosts with the router's own interface MAC address instead of the other host's MAC address.

E.g. If Host A (192.168.88.2/24) queries for the MAC address of Host B (192.168.88.3/24), the router would respond with its own MAC address. In other words, if `local-proxy-arp` is enabled, the router would assume responsibility for forwarding traffic between Host A 192.168.88.2 and Host B 192.168.88.3. All the ARP cache entries on Hosts A and B will reference the router's MAC address. In this case, the router is performing local-proxy-arp for the entire subnet 192.168.88.0/24.

An example for RouterOS local-proxy-arp could be a bridge setup with a DHCP server and isolated bridge ports where hosts from the same subnet can reach each other only at Layer3 through bridge IP.

```
/interface bridge
add arp=local-proxy-arp name=bridge1
/interface bridge port
add bridge=bridge1 horizon=1 interface=ether2
add bridge=bridge1 horizon=1 interface=ether3
add bridge=bridge1 horizon=1 interface=ether4
```

Gratuitous ARP

It is possible to create Gratuitous ARP requests in RouterOS. To do so you must use the Traffic-Generator tool, below is an example of how to generate a Gratuitous ARP request to update the ARP table on a remote device:

```
/tool traffic-generator inject interface=ether2 \
data="
ffffffffffff4c5e0c14ef78080600010800060400014c5e0c14ef780a057a01ffffffffffff0a057a0100000000000000000000000000000000
00000000"
```

You must change the MAC address (4c5e0c14ef78) and the IP address (0a057a01) to your router's address. The IP address and the MAC address must be from the device that requests an ARP table update. You also need to specify through which interface (ether2) you want to send the Gratuitous ARP request. Make sure that receiving device supports Gratuitous ARP requests.

Cloud

Summary

MikroTik offers multiple services for your RouterBOARD devices that are connected to the Internet. These services are meant to ease the inconveniences when configuring, setting up, controlling, maintaining, or monitoring your device. A more detailed list of available services that IP/Cloud can provide can be found below.

Services

Be aware that if the router has multiple public IP addresses and/or multiple internet gateways, the exact IP used for communicating with MikroTik's Cloud server may not be as expected!

IP/Cloud requires a working perpetual license on Cloud Hosted Router (CHR).

DDNS

DDNS or Dynamic DNS is a service that updates the IPv4 address for A records and the IPv6 address for AAAA records periodically. Such a service is very useful when your ISP has provided a dynamic IP address that changes periodically, but you always need an address that you can use to connect to your device remotely. Below you can find operation details that are relevant to the IP/Cloud's DDNS service:

- Checks for outgoing IP address change: every 60 seconds
- Waits for the MikroTik's Cloud server's response: 15 seconds
- DDNS record TTL: 60 seconds
- Sends encrypted packets to **cloud2.mikrotik.com** using UDP/15252 port

Since RouterOS v6.43 if your device is able to reach **cloud2.mikrotik.com** using IPv6, then a DNS **AAAA** record is going to be created for your public IPv6 address. If your device is only able to reach cloud2.mikrotik.com using IPv4, then only a DNS **A** record is going to be created for your public IPv4 address. cloud.mikrotik.com is used for older RouterOS versions prior 6.44

To enable the DDNS service:

```
[admin@MikroTik] /ip cloud set ddns-enabled=yes
[admin@MikroTik] /ip cloud print
         ddns-enabled: yes
 ddns-update-interval: none
          update-time: yes
       public-address: 159.148.147.196
  public-address-ipv6: 2a02:610:7501:1000::2
             dns-name: 529c0491d41c.sn.mynetname.net
               status: updated
```

When the service is enabled, a DNS name will be stored on MikroTik's Cloud server permanently and this DNS name will resolve to the last IP that your RouterOS instance has sent to MikroTik's Cloud server.

To disable the DDNS service:

```
/ip cloud set ddns-enabled=no
```

⚠ As soon as you disable the service, your device sends a command to MikroTik's Cloud server to remove the stored DNS name.

To manually trigger a DNS update:

```
[admin@MikroTik] > /ip cloud force-update
```

⚠ To actually connect to the device using the DNS name provided by the cloud server, a user must configure the router's firewall to permit such access from the WAN port. (Default MikroTik configuration does not permit access to services such as WebFig, WinBox, etc. from the WAN port).

Update time

Correct time on a device is important, it causes issues with the system's logs, breaks HTTPS connectivity to the device, tunnel connectivity, and other issues. To have your system's clock updated, you can use NTP or SNTP, though it requires you to specify an IP address for the NTP Server. In most cases, NTP/SNTP is not required in order to simply have a correct time set on the device, for simplicity you can use the IP Cloud's update time service. Below you can find operation details that are relevant to the IP/Cloud's update time service:

- Approximate time (accuracy of several seconds, depends on UDP packet latency)
- Updates time after a reboot and during every DDNS update (when router's WAN IP address changes or after the force-update command is used)
- Sends encrypted packets to **cloud2.mikrotik.com** using UDP/15252 port
- Detects time-zone depending on the router's public IP address and our commercial database

To enable the time update service:

```
[admin@MikroTik] > /ip cloud set update-time=yes
```

To enable automatic time zone detection:

```
[admin@MikroTik] > /system clock set time-zone-autodetect=yes
```

⚠ If `/ip cloud update-time` is set to `auto`, then the device's clock will be updated with MikroTik's Cloud server time (if no NTP or SNTP clien t is enabled).

Backup

It is possible to store your device's backup on MikroTik's Cloud server. The backup service allows you to upload an encrypted backup file, download it and apply the backup file to your device as long as your device is able to reach MikroTik's Cloud server. Below you can find operation details that are relevant to the IP/Cloud's backup service:

- 1 free backup slot for each device
- Allowed backup size: 15MB
- Sends encrypted packets to **cloud2.mikrotik.com** using UDP/15252 and TCP/15252 port

To create a new backup and upload it the MikroTik's Cloud server:

```
[admin@MikroTik] > /system backup cloud upload-file action=create-and-upload password=test123!!!
[admin@MikroTik] > /system backup cloud print
 0 name="cloud-20180921-162649" size=13.2KiB ros-version="6.44beta9" date=sep/21/2018 16:26:49 status="ok"
secret-download-key="AbCdEfGhIjKlM1234567890"
```

⚠ The `create-and-upload` action command will create a new system's backup file, encrypt the backup file with AES using the provided password and upload it. For `upload` action command the password property has no effect since the `upload` action command uploads only already created system's backup files.

To download the uploaded backup file and save it to the device's memory:

```
[admin@MikroTik] > /system backup cloud download-file action=download number=0
### OR
[admin@MikroTik] > /system backup cloud download-file action=download secret-download-
key=AbCdEfGhIjKlM1234567890
```

⚠ **Warning:** The secret-download-key is a unique identifier that can be used to download your encrypted backup to your other devices. Since you can download your encrypted backup from any location and any device by using the secret-download-key, then you should try to keep this identifier a secret. The downloaded backup is still encrypted using AES, nevertheless, make sure you are using a strong password!

To remove the uploaded backup:

```
/system backup cloud remove-file number=0
```

To upload an existing backup file (created previously):

```
[admin@MikroTik] > /system backup save encryption=aes-sha256 name=old_backup password=test123!!!
[admin@MikroTik] > /system backup cloud upload-file action=upload src-file=old_backup.backup
[admin@MikroTik] > /system backup cloud print
 0 name="cloud-20180921-164044" size=13.2KiB ros-version="6.44beta9" date=sep/21/2018 16:40:44 status="ok"
secret-download-key="AbCdEfGhIjKlM1234567890"
```

⚠ Make sure that the backup was encrypted using AES, otherwise, the IP/Cloud will reject the backup upload. Since there is only 1 free backup slot per device, then you need to remove the existing backup before uploading a new one.

⚠ **Warning:** When importing a backup all MAC addresses are set to the MAC addresses that the device was using. This is useful when you are replacing a device that has failed, but this might not be desired when applying the same backup on multiple devices since it will set the same MAC addresses on multiple devices, which can cause connectivity issues. You can always use the `reset-mac-address` command on each interface to set the original MAC address back.

Properties

Sub-menu: `/ip cloud`

Property	Description
ddns-enabled (*yes* / *no*; Default: **no**)	If set to `yes`, then the device will send an encrypted message to MikroTik's Cloud server. The server will then decrypt the message and verify that the sender is an authentic MikroTik device. If all is OK, then MikroTik's Cloud server will create a DDNS record for this device and send a response to the device. Every minute the IP/Cloud service on the router will check if the WAN IP address matches the one sent to MikroTik's Cloud server and will send an encrypted update to the cloud server if the IP address changes.
ddns-update-interval (*time, minimum 60 seconds*; Default: **none**)	If set DDNS will attempt to connect IP Cloud servers at the set interval. If set to **none** it will continue to internally check IP address update and connect to IP Cloud servers as needed. Useful if the IP address used is not on the router itself and thus, cannot be checked as a value internal to the router.
update-time (*yes* / *no*; Default: **yes**)	If set to `yes` then router clock will be set to time, provided by the cloud server **IF** there is no NTP or SNTP client enabled. If set to `no`, then IP/Cloud service will never update the device's clock. If update-time is set to `yes`, Clock will be updated even when ddns-enabled is set to `no`.

public-address (*read-only: address*)	Shows the device's IPv4 address that was sent to the cloud server. This field is visible only after at least one IP Cloud request was successfully completed.
public-address-ipv6 (*read-only: address*)	Shows the device's IPv6 address that was sent to the cloud server. This field is visible only after at least one IP Cloud request was successfully completed.
warning (*read-only: string*)	Shows a warning message if the IP address sent by the device differs from the IP address in the UDP packet header as visible by MikroTik's Cloud server. Typically this happens if the device is behind NAT. Example: "DDNS server received a request from IP 123.123.123.123 but your local IP was 192.168.88.23; DDNS service might not work"
dns-name (*read-only: name*)	Shows the DNS name assigned to the device. Name consists of 12 characters serial number appended by *.sn.mynetname.net*. This field is visible only after at least one ddns-request is successfully completed.
status (*read-only: string*)	Contains text string that describes the current dns-service state. The messages are self explanatory • **updating...** • **updated** • **Error: no Internet connection** • **Error: request timed out** • **Error: REJECTED. Contact MikroTik support** • **Error: internal error** - should not happen. One possible cause is if the router runs out of memory

Advanced

Sub-menu: `/ip cloud advanced`

Property	Description
use-local-address (*yes* / *no*; Default: **no**)	By default, the DNS name will be assigned to the detected public address (from the UDP packet header). If you wish to send your "local" or "internal" IP address, then set this to `yes`

Cloud backup

Sub-menu: `/system backup cloud`

Below you can find commands and properties that are relevant to the specific command, other properties will not have any effect.

- download-file

Property	Description
action (*download*)	Downloads an uploaded backup file from MikroTik's Cloud server.
number (*integer*)	Specifies the backup slot on MikroTik's Cloud server, the free backup slot is always going to be in the `0th` slot.
secret-download-key (*string*)	Unique identifier that can be used to download your uploaded backup file. When downloading the uploaded backup file you do not have to be using the same device, from which the backup was uploaded from. Useful when deploying a backup on a new device.

- remove-file

Property	Description
number (*integer*)	Deletes the backup file in the specified backup slot, the free backup slot is always going to be in the `0th` slot.

- upload-file

Property	Description
action (*create-and-upload*)	Uploads a backup file to MikroTik's Cloud server. • `create-and-upload` - creates a new backup file with the specified password and uploads it • `upload` - uploads a created system's backup file.
name (*string*)	Specifies the backup's name that will show up in the uploaded backups list. This is **NOT** the source backup's name, this name is only used for visual representation.
src-file (*file*)	Backup's file name to upload that was created using `/system backup`. This property only has an effect when the action is set to `upload`.
password (*string*)	Create, encrypt and upload a backup file with the specified password. This property only has an effect when the action is set to `create-and-upload`.

DNS

Introduction

Domain Name System (DNS) usually refers to the Phonebook of the Internet. In other words, DNS is a database that links strings (known as hostnames), such as www.mikrotik.com to a specific IP address, such as 159.148.147.196.

A MikroTik router with a DNS feature enabled can be set as a DNS cache for any DNS-compliant client. Moreover, the MikroTik router can be specified as a primary DNS server under its DHCP server settings. When the remote requests are enabled, the MikroTik router responds to TCP and UDP DNS requests on port 53.

When both static and dynamic servers are set, static server entries are preferred, however, it does not indicate that a static server will always be used (for example, previously query was received from a dynamic server, but static was added later, then a dynamic entry will be preferred).

When DNS server *allow-remote-requests* are used make sure that you limit access to your server over TCP and UDP protocol port 53 only for known hosts.

There are several options on how you can manage DNS functionality on your LAN - use public DNS, use the router as a cache, or do not interfere with DNS configuration. Let us take as an example the following setup: Internet service provider (ISP) → Gateway (GW) → Local area network (LAN). The GW is RouterOS based device with the default configuration:

- You do not configure any DNS servers on the "GW" DHCP server network configuration - the device will forward the DNS server IP address configuration received from `ISP` to `LAN` devices;
- You configure DNS servers on the "GW" DHCP server network configuration - the device will give configured DNS servers to `LAN` devices (also "/ip dns set allow-remote-requests=yes" *must* be enabled);
- "dns-none" configured under DNS servers on "GW" DHCP server network configuration - the device will not forward any of the **dynamic** DNS servers to `LAN` devices;

DNS configuration

DNS facility is used to provide domain name resolution for the router itself as well as for the clients connected to it.

Property	Description
allow-remote-requests (*yes* | *no*; Default: **no**)	Specifies whether to allow router usage as a DNS cache for remote clients. Otherwise, only the router itself will use DNS configuration.
cache-max-ttl (*time*; Default: **1w**)	Maximum time-to-live for cache records. In other words, cache records will expire unconditionally after cache-max-TTL time. Shorter TTLs received from DNS servers are respected.
cache-size (*integer[64..4294967295]*; Default: **2048**)	Specifies the size of the DNS cache in KiB.
max-concurrent-queries (*integer*; Default: **100**)	Specifies how many concurrent queries are allowed.
max-concurrent-tcp-sessions (*integer*; Default: **20**)	Specifies how many concurrent TCP sessions are allowed.

max-udp-packet-size (*integer [50..65507]*; Default: **4096**)	Maximum size of allowed UDP packet.
query-server-timeout (*time*; Default: **2s**)	Specifies how long to wait for a query response from a server.
query-total-timeout (*time*; Default: **10s**)	Specifies how long to wait for query response in total. Note that this setting must be configured taking into account "query-server-timeout" and the number of used DNS servers.
servers (*list of IPv4/IPv6 addresses*; Default:)	List of DNS server IPv4/IPv6 addresses
cache-used (*integer*)	Shows the currently used cache size in KiB
dynamic-server (*IPv4/IPv6 list*)	List of dynamically added DNS servers from different services, for example, DHCP.
doh-max-concurrent-queries (*integer*; Default: **50**)	Specifies how many DoH concurrent queries are allowed.
doh-max-server-connections (*integer*; Default: **5**)	Specifies how many concurrent connections to the DoH server are allowed.
doh-timeout (*time*; Default: **5s**)	Specifies how long to wait for query response from the DoH server.
use-doh-server (*string; Default:)*	Specified which DoH server must be used for DNS queries. DoH functionality overrides "*servers*" usage if specified. The server must be specified with an "https://" prefix.
verify-doh-cert (*yes* | *no*; Default: **no**)	Specifies whether to validate the DoH server, when one is being used. Will use the "/certificate" list in order to verify server validity.

```
[admin@MikroTik] > ip dns print
                    servers:
             dynamic-servers: 10.155.0.1
              use-doh-server:
             verify-doh-cert: no
  doh-max-server-connections: 5
  doh-max-concurrent-queries: 50
                doh-timeout: 5s
       allow-remote-requests: yes
         max-udp-packet-size: 4096
        query-server-timeout: 2s
         query-total-timeout: 10s
      max-concurrent-queries: 100
 max-concurrent-tcp-sessions: 20
                cache-size: 2048KiB
               cache-max-ttl: 1d
                cache-used: 48KiB
```

Dynamic DNS servers are obtained from different facilities available in RouterOS, for example, DHCP client, VPN client, IPv6 Router Advertisements, etc.

Servers are processed in a queue order - static servers as an ordered list, dynamic servers as an ordered list. When DNS cache has to send a request to the server, it tries servers one by one until one of them responds. After that this server is used for all types of DNS requests. Same server is used for any types of DNS requests, for example, A and AAAA types. If you use only dynamic servers, then the DNS returned results can change after reboot, because servers can be loaded into IP/DNS settings in a different order due to a different speeds on how they are received from facilities mentioned above.

If at some point the server which was being used becomes unavailable and can not provide DNS answers, then the DNS cache restarts the DNS server lookup process and goes through the list of specified servers once more.

DNS Cache

This menu provides two lists with DNS records stored on the server:

- *"*/ip dns cache*"*: this menu provides a list with cache DNS entries that RouterOS cache can reply with to client requests ;
- *"*/ip dns cache all*"*: This menu provides a complete list with all cached DNS records stored including also, for example, PTR records.

You can empty the DNS cache with the command: "/ip dns cache flush".

DNS Static

The MikroTik RouterOS DNS cache has an additional embedded DNS server feature that allows you to configure multiple types of DNS entries that can be used by the DNS clients using the router as their DNS server. This feature can also be used to provide false DNS information to your network clients. For example, resolving any DNS request for a certain set of domains (or for the whole Internet) to your own page.

```
[admin@MikroTik] /ip dns static add name=www.mikrotik.com address=10.0.0.1
```

The server is also capable of resolving DNS requests based on POSIX basic regular expressions so that multiple requests can be matched with the same entry. In case an entry does not conform with DNS naming standards, it is considered a regular expression. The list is ordered and checked from top to bottom. Regular expressions are checked first, then the plain records.

Use regex to match DNS requests:

```
[admin@MikroTik] /ip dns static add regexp="[*mikrotik*]" address=10.0.0.2
```

If DNS static entries list matches the requested domain name, then the router will assume that this router is responsible for any type of DNS request for the particular name. For example, if there is only an "A" record in the list, but the router receives an "AAAA" request, then it will reply with an "A" record from the static list and will query the upstream server for the "AAAA" record. If a record exists, then the reply will be forwarded, if not, then the router will reply with an "ok" DNS reply without any records in it. If you want to override domain name records from the upstream server with unusable records, then you can, for example, add a static entry for the particular domain name and specify a dummy IPv6 address for it "::ffff".

List all of the configured DNS entries as an ordered list:

```
[admin@MikroTik] /ip/dns/static/print
Columns: NAME, REGEXP, ADDRESS, TTL
# NAME             REGEXP       ADDRESS   TTL
0 www.mikrotik.com               10.0.0.1  1d
1                  [*mikrotik*]  10.0.0.2  1d
```

Property	Description
address (*IPv4/IPv6*)	The address that will be used for "A" or "AAAA" type records.
cname (*string*)	Alias name for a domain name.
forward-to	The IP address of a domain name server to which a particular DNS request must be forwarded.
mx-exchange (*string*)	The domain name of the MX server.
name (*string*)	Domain name.
srv-port (*integer*; Default: 0)	The TCP or UDP port on which the service is to be found.
srv-target	The canonical hostname of the machine providing the service ends in a dot.
text (*string*)	Textual information about the domain name.
type (*A* | *AAAA* | *CNAME* | *FWD* | *MX* | *NS* | *NXDOMAIN* | *SRV* | *TXT* ; Default: *A*)	Type of the DNS record.
address-list (*string*)	Name of the Firewall address list to which address must be dynamically added when some request matches the entry. Entry will be removed from the address list when TTL expires.
comment (*string*)	Comment about the domain name record.
disabled (*yes* | *no*; Default: yes)	Whether the DNS record is active.
match-subdomain (*yes* | *no*; Default: no)	Whether the record will match requests for subdomains.

mx-preference (*integer*; Default: 0)	Preference of the particular MX record.
ns (*string*)	Name of the authoritative domain name server for the particular record.
regexp (POSIX regex)	Regular expression against which domain names should be verified.
srv-priority (*integer*; Default: 0)	Priority of the particular SRV record.
srv-weight (*integer*; Default: 0)	Weight of the particular SRV record.
ttl (*time*; Default: *24h*)	Maximum time-to-live for cached records.

Regexp is case-sensitive, but DNS requests are not case sensitive, RouterOS converts DNS names to lowercase before matching any static entries. You should write regex only with lowercase letters. Regular expression matching is significantly slower than plain text entries, so it is advised to minimize the number of regular expression rules and optimize the expressions themselves.

Be careful when you configure regex through mixed user interfaces - CLI and GUI. Adding the entry itself might require escape characters when added from CLI. It is recommended to add an entry and the execute print command in order to verify that regex was not changed during addition.

DNS over HTTPS (DoH)

Starting from RouterOS version v6.47 it is possible to use DNS over HTTPS (DoH). DoH uses HTTPS protocol to send and receive DNS requests for better data integrity. The main goal is to provide privacy by eliminating "man-in-the-middle" attacks (MITM).

Currently, DoH is not compatible with FWD-type static entries, in order to utilize FWD entries, DoH must not be configured.

Watch our video about this feature.

It is strongly recommended to import the root CA certificate of the DoH server you have chosen to use for increased security. We strongly suggest not using third-party download links for certificate fetching. Use the Certificate Authority's own website.

There are various ways to find out what root CA certificate is necessary. The easiest way is by using your WEB browser, navigating to the DoH site, and checking the security of the website. You can download the certificate straight from the browser or fetch the certificate from a trusted source.

Download the certificate, upload it to your router and import it:

```
/certificate import file-name=CertificateFileName
```

Configure the DoH server:

```
/ip dns set use-doh-server=DoH_Server_Query_URL verify-doh-cert=yes
```

Note that you need at least one regular DNS server configured for the router to resolve the DoH hostname itself. If you do not have any dynamical or static DNS server configured, add a static DNS entry for the DoH server domain name like this:

```
/ip dns set servers=1.1.1.1
```

If you do not have any dynamical or static DNS server configured, add a static DNS entry for the DoH server domain name like this:

```
/ip dns static add address=IP_Address name=Domain_Name
```

RouterOS prioritizes DoH over the DNS server if both are configured on the device.

If */certificate/settings/set crl-use* is set to *yes,* RouterOS will check CRL for each certificate in a certificate chain, therefore, an entire certificate chain should be installed into a device - starting from Root CA, intermediate CA (if there are such), and certificate that is used for specific service.

For example, Google DoH, Cloudflare, and OpenDNS full chain contain three certificates, NextDNS has four certificates.

Known compatible/incompatible DoH services

Compatible DoH services:

- Cloudflare
- Google
- NextDNS
- OpenDNS

Incompatible DoH services:

- Mullvad
- Yandex

Adlist

Adlist is an integral component of network-level ad blocking, comprising a curated collection of domain names known for serving advertisements. This feature operates by utilizing Domain Name System (DNS) resolution to intercept requests to these domains. When a client device queries a DNS server for a domain listed on the adlist, the DNS resolution process is altered. Instead of returning the actual IP address of the ad-serving domain, the DNS server responds with the IP address 0.0.0.0. This effectively null-routes the request, as 0.0.0.0 is a non-routable meta-address used to denote an invalid, unknown, or non-applicable target. By redirecting ad-related requests in this manner, the adlist feature ensures that advertisement content is not loaded, enhancing network performance and improving the user experience by reducing unwanted ad traffic.

Before configuring, increase the DNS cache as it's used to store adlist entries. If limit is reached and error in DNS,error topic is printed "*adlist read: max cache size reached*"

Property	Description
url	Used to specify the URL of an adlist.
ssl-verify	Specifies whether to validate the server's SSL certificate when connecting to an online resource. Will use the "/certificate" list in order to verify server validity.
file	Used to specify a local file path from which to read adlist data
pause	Temporarily pause the use of all adlist.

Configuration examples:

URL based adlist:

```
/ip/dns/adlist add url=https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts ssl-verify=no
```

To see how many domain names are present and matched, you can run:

```
/ip/dns/adlist/print
Flags: X - disabled
0 url="https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts" ssl-verify=no match-count=122 name-
count=164769
```

Locally hosted adlist:

To create your adlist, you can create a Txt file with the domains. Example:

```
0.0.0.0 example1.com
0.0.0.0 eu1.example.com
0.0.0.0 ex.com
0.0.0.0 com.example.com
```

> You can create the txt file on your PC, but it is also possible to create it in RouterOS, with following commands
>
> "/file/add name=host.txt", and then you can run "file/edit host.txt contents" after adding entries, press "ctrl o" to save the entries.

To add file to adlist :

```
/ip/dns/adlist/add file=host.txt match-count=0 name-count=4
```

> You can verify that file is formatted correctly with "/ip/dns/adlist/print" ,the results will show how many hostnames you have added, the hostname format must match the format given in previous example.

DHCP

DHCP Client

Summary

```
/ip dhcp-client
```

The DHCP (Dynamic Host Configuration Protocol) is used for the easy distribution of IP addresses in a network. The MikroTik RouterOS implementation includes both server and client parts and is compliant with RFC 2131.

The MikroTik RouterOS DHCP client may be enabled on any Ethernet-like interface at a time. The client will accept an address, netmask, default gateway, and two DNS server addresses. The received IP address will be added to the interface with the respective netmask. The default gateway will be added to the routing table as a dynamic entry. Should the DHCP client be disabled or not renew an address, the dynamic default route will be removed. If there is already a default route installed prior to the DHCP client obtaining one, the route obtained by the DHCP client would be shown as invalid.

RouterOS DHCP client asks for the following options:

- option 1 - SUBNET_MASK,
- option 3 - GATEWAY_LIST,
- option 6 - TAG_DNS_LIST,
- option 33 - STATIC_ROUTE,
- option 42 - NTP_LIST,
- option 121 - CLASSLESS_ROUTE,

DHCP Options

DHCP client has the possibility to set up options that are sent to the DHCP server. For example, hostname and MAC address. The syntax is the same as for DHCP server options.

Currently, there are three variables that can be used in options:

- HOSTNAME;
- CLIENT_MAC - client interface MAC address;
- CLIENT_DUID - client DIUD of the router, same as used for the DHCPv6 client. In conformance with RFC4361

DHCP client default options include these default Options:

Name	code	value
clientid_duid	61	0xff$(CLIENT_DUID)
clientid	61	0x01$(CLIENT_MAC)
hostname	12	$(HOSTNAME)

Properties

Property	Description
add-default-route (*yes* | *no* | *special-classless*; Default: **yes**)	Whether to install default route in routing table received from DHCP server. By default, the RouterOS client complies with RFC and ignores option 3 if classless option 121 is received. To force the client not to ignore option 3 set *special-classless*. This parameter is available in v6rc12+ - **yes** - adds classless route if received, if not then add default route (old behavior) - **special-classless** - adds both classless routes if received and a default route (MS style)
client-id (*string*; Default:)	Corresponds to the settings suggested by the network administrator or ISP. If not specified, the client's MAC address will be sent
comment (*string*; Default:)	Short description of the client
default-route-distance (*integer:0..255*; Default:)	Distance of default route. Applicable if `add-default-route` is set to `yes`.
disabled (*yes* | *no*; Default: **yes**)	
host-name (*string*; Default:)	The hostname of the client is sent to a DHCP server. If not specified, the client's system identity will be used.

interface (*string*; Default:)	The interface on which the DHCP client will be running.
script (*script*; Default:)	Execute script when DHCP client obtains a new lease or loses an existing one. This parameter is available in v6.39rc33+ These are available variables that are accessible for the event script: • bound - 1 - lease is added/changed; 0 - lease is removed • server-address - server address • lease-address - lease address provided by a server • interface - name of the interface on which the client is configured • gateway-address - gateway address provided by a server • vendor-specific - stores value of option 43 received from DHCP server • lease-options - an array of received options Example >>
use-peer-dns (*yes* | *no*; Default: **yes**)	Whether to accept the DNS settings advertised by DHCP Server. (Will override the settings put in the `/ip dns` submenu.
use-peer-ntp (*yes* | *no*; Default: **yes**)	Whether to accept the NTP settings advertised by DHCP Server. (Will override the settings put in the `/system ntp client` submenu)

Read-only properties

Property	Description
address (*IP/Netmask*)	IP address and netmask, which is assigned to DHCP Client from the Server
dhcp-server (*IP*)	The IP address of the DHCP server.
expires-after (*time*)	A time when the lease expires (specified by the DHCP server).
gateway (*IP*)	The IP address of the gateway which is assigned by the DHCP server
invalid (*yes* | *no*)	Shows whether a configuration is invalid.
netmask (*IP*)	
primary-dns (*IP*)	The IP address of the first DNS resolver, which was assigned by the DHCP server
primary-ntp (*IP*)	The IP address of the primary NTP server, assigned by the DHCP server
secondary-dns (*IP*)	The IP address of the second DNS resolver, assigned by the DHCP server
secondary-ntp (*IP*)	The IP address of the secondary NTP server, assigned by the DHCP server
status (*bound* | *error* | *rebinding...* | *requesting...* | *searching...* | *stopped*)	Shows the status of the DHCP Client

Menu specific commands

Property	Description
release (*numbers*)	Release current binding and restart the DHCP client
renew (*numbers*)	Renew current leases. If the renewal operation was not successful, the client tries to reinitialize the lease (i.e. it starts the lease request procedure (rebind) as if it had not received an IP address yet)

Configuration Examples

Simple DHCP client

Add a DHCP client on the ether1 interface:

```
/ip dhcp-client add interface=ether1 disabled=no
```

After the interface is added, you can use the "print" or "print detail" command to see what parameters the DHCP client acquired:

```
[admin@MikroTik] ip dhcp-client> print detail
Flags: X - disabled, I - invalid
 0   interface=ether1 add-default-route=yes use-peer-dns=yes use-peer-ntp=yes
     status=bound address=192.168.0.65/24 gateway=192.168.0.1
     dhcp-server=192.168.0.1 primary-dns=192.168.0.1 primary-ntp=192.168.0.1
     expires-after=9m44s
[admin@MikroTik] ip dhcp-client>
```


If the interface used by the DHCP client is part of the VRF configuration, then the default route and other received routes from the DHCP server will be added to the VRF routing table.

DHCP client status can be checked with:

```
/ip dhcp-client print detail
```

Lease script example

It is possible to execute a script when a DHCP client obtains a new lease or loses an existing one. This is an example script that automatically adds a default route with routing-table=WAN1 and removes it when the lease expires or is removed.

```
/ip dhcp-client
add add-default-route=no dhcp-options=hostname,clientid disabled=no interface=ether2 script="{\r\
    \n    :local rmark \"WAN1\"\r\
    \n    :local count [/ip route print count-only where comment=\"WAN1\"]\r\
    \n    :if (\$bound=1) do={\r\
    \n        :if (\$count = 0) do={\r\
    \n            /ip route add gateway=\$\"gateway-address\" comment=\"WAN1\" routing-table=\$rmark\r\
    \n        } else={\r\
    \n            :if (\$count = 1) do={\r\
    \n                :local test [/ip route find where comment=\"WAN1\"]\r\
    \n                :if ([/ip route get \$test gateway] != \$\"gateway-address\") do={\r\
    \n                    /ip route set \$test gateway=\$\"gateway-address\"\r\
    \n                }\r\
    \n            } else={\r\
    \n                :error \"Multiple routes found\"\r\
    \n            }\r\
    \n        }\r\
    \n    } else={\r\
    \n        /ip route remove [find comment=\"WAN1\"]\r\
    \n    }\r\
    \n}\r\
    \n"
```

Resolve default gateway when 'router' (option3) is from a different subnet

In some cases, administrators tend to set the 'router' option which cannot be resolved with offered IP's subnet. For example, the DHCP server offers 192.168.88.100/24 to the client, and option 3 is set to 172.16.1.1. This will result in an unresolved default route:

```
#      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
0  DS  0.0.0.0/0                          172.16.1.1              1
1 ADC  192.168.88.0/24    192.168.88.100  ether1
```

To fix this we need to add /32 route to resolve the gateway over ether1, which can be done by the running script below each time the DHCP client gets an address

```
/system script add name="dhcpL" source={ /ip address add address=($"lease-address" . "/32") network=$"gateway-
address" interface=$interface }
```

Now we can further extend the script, to check if the address already exists, and remove the old one if changes are needed

```
/system script add name="dhcpL" source={
  /ip address {
    :local ipId [find where comment="dhcpL address"]
    :if ($ipId != "") do={
      :if (!([get $ipId address] = ($"lease-address" . "/32") && [get $ipId network]=$"gateway-address" )) do={
        remove $ipId;
        add address=($"lease-address" . "/32") network=$"gateway-address" \
          interface=$interface comment="dhcpL address"
      }
    } else={
      add address=($"lease-address" . "/32") network=$"gateway-address" \
        interface=$interface comment="dhcpL address"
    }
  }
}
```

DHCPv6 Client

Summary

Sub-menu: `/ipv6 dhcp-client`

DHCP-client in RouterOS is capable of being a DHCPv6-client and DHCP-PD client. So it is able to get a prefix from the DHCP-PD server as well as the DHCPv6 stateful address from the DHCPv6 server.

Properties

Property	Description
add-default-route (*yes* | *no*; Default: **no**)	Whether to add default IPv6 route after a client connects.
comment (*string*; Default:)	Short description of the client
disabled (*yes* | *no*; Default: **no**)	
interface (*string*; Default:)	The interface on which the DHCPv6 client will be running.
pool-name (*string*; Default:)	Name of the IPv6 pool in which received IPv6 prefix will be added
pool-prefix-length (*string*; Default:)	Prefix length parameter that will be set for IPv6 pool in which received IPv6 prefix is added. Prefix length must be greater or equal as the length of received prefix, otherwise, prefix-length will be set to received prefix length + 8 bits.
prefix-hint (*string*; Default:)	Include a preferred prefix length.
request (*prefix, address*; Default:)	to choose if the DHCPv6 request will ask for the address or the IPv6 prefix, or both.

script (*string*; Default:)	Run this script on the DHCP-client status change. Available variables: • pd-valid - if the prefix is acquired by the client; • pd-prefix - the prefix acquired by the client if any; • na-valid - if the address is acquired by the client; • na-address - the address acquired by the client if any. • options - array of received options (only ROSv7)
use-peer-dns (*yes* | *no*; Default: **yes**)	Whether to accept the DNS settings advertised by the IPv6 DHCP Server.

Read-only properties

Property	Description
duid (*string*)	Auto-generated DUID that is sent to the server. DUID is generated using one of the MAC addresses available on the router.
request (*list*)	specifies what was requested - prefix, address, or both.
dynamic (*yes* | *no*)	
expires-after (*time*)	A time when the IPv6 prefix expires (specified by the DHCPv6 server).
invalid (*yes* | *no*)	Shows whether a configuration is invalid.
prefix (*IPv6 prefix*)	Shows received IPv6 prefix from DHCPv6-PD server
status (*stopped* | *searching* | *requesting...* | *bound* | *renewing* | *rebinding* | *error* | *stopping*)	Shows the status of DHCPv6 Client: • **stopped** - dhcpv6 client is stopped • **searching** - sending "solicit" and trying to get "advertise" • **requesting** - sent "request" waiting for "reply" • **bound** - received "reply". Prefix assigned. • **renewing** - sent "renew", waiting for "reply" • **rebinding** - sent "rebind", waiting for "reply" • **error** - reply was not received in time or some other error occurred. • **stopping** - sent "release"

Menu specific commands

Property	Description
release (*numbers*)	Release current binding and restart DHCPv6 client
renew (*numbers*)	Renew current leases. If the renewal operation was not successful, the client tries to reinitialize the lease (i.e. it starts the lease request procedure (rebind) as if it had not received an IP address yet)

Script

It is possible to add a script that will be executed when a prefix or an address is acquired and applied or expires and is removed using the DHCP client. There are separated sets of variables that will have the value set by the client depending on prefix or address status change as the client can acquire both and each of them can have a different effect on the router configuration.

Available variables for dhcp-client

- pd-valid - value - 1 or 0 - if prefix is acquired and it is applied or not
- pd-prefix - value ipv6/num (ipv6 prefix with mask) - the prefix inself
- na-valid - value - 1 or 0 - if address is acquired and it is applied or not
- na-address - value - ipv6 address - the address

IAID

To determine what IAID will be used, convert the internal ID of an interface on which the DHCP client is running from hex to decimal.

For example, the DHCP client is running on interface PPPoE-out1. To get internal ID use the following command:

```
[admin@t36] /interface> :put [find name="pppoe-out1"]
*15
```

Now convert hex value 15 to decimal and you get IAID=21

Configuration Examples

Simple DHCPv6 client

This simple example demonstrates how to enable dhcp client to receive IPv6 prefix and add it to the pool.

```
/ipv6 dhcp-client add request=prefix pool-name=test-ipv6 pool-prefix-length=64 interface=ether13
```

Detailed print should show status of the client and we can verify if prefix is received

```
[admin@x86-test] /ipv6 dhcp-client> print detail
Flags: D - dynamic, X - disabled, I - invalid
 0 interface=bypass pool-name="test-ipv6" pool-prefix-length=64 status=bound
prefix=2001:db8:7501:ff04::/62 expires-after=2d23h11m53s request=prefix
```

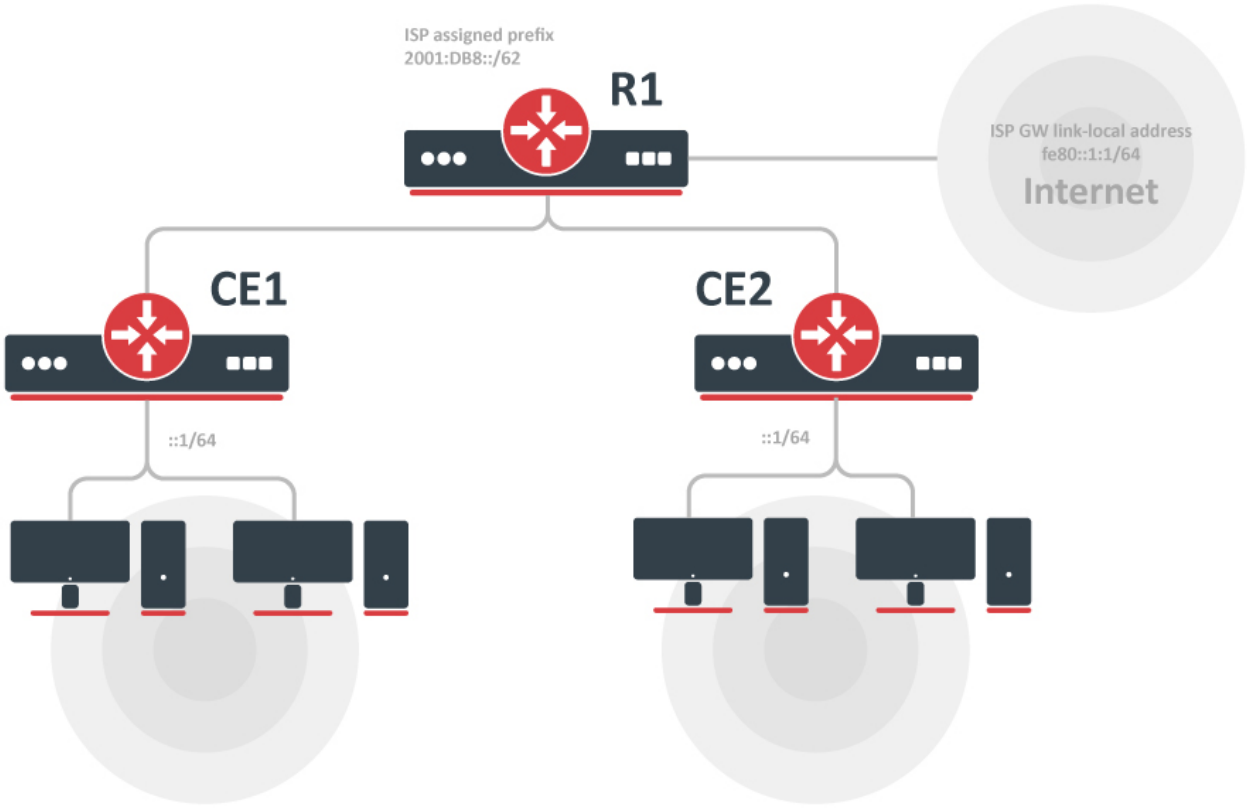
Notice that server gave us prefix 2a02:610:7501:ff04::/62 . And it should be also added to ipv6 pools

```
[admin@MikroTik] /ipv6 pool> print
Flags: D - dynamic
# NAME PREFIX REQUEST PREFIX-LENGTH
0 D test-ipv6 2001:db8:7501:ff04::/62 prefix 64
```

It works! Now you can use this pool, for example, for pppoe clients.

Use received prefix for local RA

Consider following setup:



- ISP is routing prefix 2001:DB8::/62 to the router R1
- Router R1 runs DHCPv6 server to delegate /64 prefixes to the customer routers CE1 CE2
- DHCP client on routers CE1 and CE2 receives delegated /64 prefix from the DHCP server (R1).
- Client routers uses received prefix to set up RA on the local interface

Configuration

R1

```
/ipv6 route
add gateway=fe80::1:1%to-ISP

/ipv6 pool
add name=myPool prefix=2001:db8::/62 prefix-length=64

/ipv6 dhcp-server
 add address-pool=myPool disabled=no interface=to-CE-routers lease-time=3m name=server1
```

CE1

```
/ipv6 dhcp-client
add interface=to-R1 request=prefix pool-name=my-ipv6

/ipv6 address
add address=::1/64 from-pool=my-ipv6 interface=to-clients advertise=yes
```

CE2

```
/ipv6 dhcp-client
 add interface=to-R1 request=prefix pool-name=my-ipv6
/ipv6 address add address=::1/64 from-pool=my-ipv6 interface=to-clients advertise=yes
```

Check the status

After configuration is complete we can verify that each CE router received its own prefix

On server:

```
[admin@R1] /ipv6 dhcp-server binding> print
 Flags: X - disabled, D - dynamic
# ADDRESS DUID IAID SERVER STATUS
 1 D 2001:db8:1::/64 0019d1393536 566 server1 bound
2 D 2001:db8:2::/64 0019d1393535 565 server1 bound
```

On client:

```
[admin@CE1] /ipv6 dhcp-client> print
Flags: D - dynamic, X - disabled, I - invalid
# INTERFACE STATUS REQUEST PREFIX
0 to-R1 bound prefix 2001:db8:1::/64

[admin@CE1] /ipv6 dhcp-client> /ipv6 pool print
Flags: D - dynamic
# NAME PREFIX PREFIX-LENGTH
0 D my-ipv6 2001:db8:1::/64 64
```

We can also see that IPv6 address was automatically added from the prefix pool:

```
[admin@CE1] /ipv6 address> print
Flags: X - disabled, I - invalid, D - dynamic, G - global, L - link-local
# ADDRESS FROM-POOL INTERFACE ADVERTISE 0 G 2001:db8:1::1/64 to-clients yes
..
```

And pool usage shows that 'Address' is allocating the pool

```
[admin@CE1] /ipv6 pool used> print
 POOL PREFIX OWNER INFO
my-ipv6 2001:db8:1::/64 Address to-clients
```

DHCP Server

Summary

The DHCP (Dynamic Host Configuration Protocol) is used for the easy distribution of IP addresses in a network. The MikroTik RouterOS implementation includes both server and client parts and is compliant with RFC 2131.

The router supports an individual server for each Ethernet-like interface. The MikroTik RouterOS DHCP server supports the basic functions of giving each requesting client an IP address/netmask lease, default gateway, domain name, DNS-server(s) and WINS-server(s) (for Windows clients) information (set up in the DHCP networks submenu)

In order for the DHCP server to work, IP pools must also be configured (do not include the DHCP server's own IP address into the pool range) and the DHCP networks.

It is also possible to hand out leases for DHCP clients using the RADIUS server; the supported parameters for a RADIUS server are as follows:

Access-Request:

- NAS-Identifier - router identity
- NAS-IP-Address - IP address of the router itself
- NAS-Port - unique session ID
- NAS-Port-Type - Ethernet
- Calling-Station-Id - client identifier (active-client-id)
- Framed-IP-Address - IP address of the client (active-address)
- Called-Station-Id - the name of DHCP server
- User-Name - MAC address of the client (active-mac-address)
- Password - " "

Access-Accept:

- Framed-IP-Address - IP address that will be assigned to a client
- Framed-Pool - IP pool from which to assign an IP address to a client
- Rate-Limit - Datarate limitation for DHCP clients. Format is: rx-rate[/tx-rate] [rx-burst-rate[/tx-burst-rate] [rx-burst-threshold[/tx-burst-threshold] [rx-burst-time[/tx-burst-time][priority] [rx-rate-min[/tx-rate-min]]]]. All rates should be numbers with optional 'k' (1,000s) or 'M' (1,000,000s). If tx-rate is not specified, rx-rate is as tx-rate too. Same goes for tx-burst-rate and tx-burst-threshold and tx-burst-time. If both rx-burst-threshold and tx-burst-threshold are not specified (but burst-rate is specified), rx-rate and tx-rate are used as burst thresholds. If both rx-burst-time and tx-burst-time are not specified, 1s is used as default. Priority takes values 1..8, where 1 implies the highest priority, but 8 - the lowest. If rx-rate-min and tx-rate-min are not specified rx-rate and tx-rate values are used. The rx-rate-min and tx-rate-min values can not exceed rx-rate and tx-rate values.
- Ascend-Data-Rate - TX/RX data rate limitation if multiple attributes are provided, first limits tx data rate, second - RX data rate. If used together with Ascend-Xmit-Rate, specifies RX rate. 0 if unlimited
- Ascend-Xmit-Rate - tx data rate limitation. It may be used to specify the TX limit only instead of sending two sequential Ascend-Data-Rate attributes (in that case Ascend-Data-Rate will specify the receive rate). 0 if unlimited
- Session-Timeout - max lease time (lease-time)

DHCP server requires a real interface to receive raw ethernet packets. If the interface is a Bridge interface, then the Bridge must have a real interface attached as a port to that bridge which will receive the raw ethernet packets. It cannot function correctly on a dummy (empty bridge) interface.

DHCP Server Properties

Property	Description
add-arp (*yes* | *no*; Default: **no**)	Whether to add dynamic ARP entry. If set to **no** either ARP mode should be enabled on that interface or static ARP entri es should be administratively defined in */ip arp* submenu.
address-pool (*string* | *static-only*; Default: **static-only**)	IP pool, from which to take IP addresses for the clients. If set to **static-only**, then only the clients that have a static lease (added in the lease submenu) will be allowed.
allow-dual-stack-queue (*yes* | *no*; Default: **yes**)	Creates a single simple queue entry for both IPv4 and IPv6 addresses, and uses the MAC address and DUID for identification. Requires IPv6 DHCP Server to have this option enabled as well to work properly.
always-broadcast (*yes* | *no*; Default: **no**)	Changes whether to force broadcast DHCP replies: • no - replies are sent based on the client's broadcast flag. If the server sends three consecutive offers, the third and forth offer will be sent as a broadcast; • yes - replies are always broadcasted even when the client has not specified the broadcast flag.

authoritative (*after-10sec-delay* / *after-2sec-delay* / *yes* / *no*; Default: **yes**)	Option changes the way how a server responds to DHCP requests: • yes - replies to clients' requests for an address that is not available from this server, DHCP server will send a negative acknowledgment (DHCPNAK); • no - DHCP server ignores clients' requests for addresses that are not available from this server; • after-10sec-delay - requests with "secs < 10" will be processed as in "no" setting case and requests with "secs >= 10" will be processed as in "yes" case; • after-2sec-delay - requests with "secs < 2" will be processed as in "no" setting case and requests with "secs >= 2" will be processed as in "yes" case; If all requests with "secs < x" should be ignored, then **delay-threshold=x** setting should be used.
bootp-lease-time (*forever* / *lease-time* / *time*; Default: **for ever**)	Accepts two predefined options or time value: • forever - lease never expires • lease-time - use time from lease-time parameter
bootp-support (*none* / *static* / *dynamic*; Default: **static**)	Support for BOOTP clients: • none - do not respond to BOOTP requests • static - offer only static leases to BOOTP clients • dynamic - offer static and dynamic leases for BOOTP clients
client-mac-limit (*integer* / *unlimited*; Default: **unlimited**)	Specifies whether to limit a specific number of clients per single MAC address or leave unlimited. Note that this setting should not be used in relay setups.
conflict-detection (*yes* / *no*; Default: **yes**)	Allows disabling/enabling conflict detection. If the option is enabled, then whenever the server tries to assign a lease it will send ICMP and ARP messages to detect whether such an address in the network already exists. If any of the above get a reply address is considered already used.
delay-threshold (*time* / *none*; Default: **none**)	If the sec's field in the DHCP packet is smaller than the delay threshold, then this packet is ignored. If set to **none** - there is no threshold (all DHCP packets are processed)
dhcp-option-set (*name* / *none* ; Default: **none**)	Use a custom set of DHCP options defined in the option sets menu.
insert-queue-before (*bottom* / *first* / *name*; Default: **first**)	Specify where to place dynamic simple queue entries for static DCHP leases with a rate-limit parameter set.
interface (*string*; Default:)	The interface on which the DHCP server will be running.
lease-script (*string*; Default: **""**)	A script that will be executed after a lease is assigned or de-assigned. Internal "global" variables that can be used in the script: • leaseBound - set to "1" if bound, otherwise set to "0" • leaseServerName - DHCP server name • leaseActMAC - active mac address • leaseActIP - active IP address • lease-hostname - client hostname • lease-options - an array of received options
lease-time (*time*; Default: **30m**)	The time that a client may use the assigned address. The client will try to renew this address after half of this time and will request a new address after the time limit expires.
name (*string*; Default:)	Reference name
parent-queue (*string* / *none*; Default: **none**)	A dynamically created queue for this lease will be configured as a child queue of the specified parent queue.

relay (*IP*; Default: **0.0.0.0**)	The IP address of the relay this DHCP server should process requests from: • 0.0.0.0 - the DHCP server will be used only for direct requests from clients (no DHCP relay allowed) • 255.255.255.255 - the DHCP server should be used for any incoming request from a DHCP relay except for those, which are processed by another DHCP server that exists in the `/ip dhcp-server` submenu.
server-address (*IP*; Default: **0.0.0.0**)	The IP address of the server to use in the next step of the client's bootstrap process (For example, to assign a specific server address in case several addresses are assigned to the interface)
use-framed-as-classless (*yes* | *no*; Default: **yes**)	Forward RADIUS Framed-Route as a DHCP Classless-Static-Route to DHCP-client. Whenever both Framed-Route and Classless-Static-Route are received Classless-Static-Route is preferred.
use-radius (*yes* | *no* | *accounting*; Default: **no**)	Whether to use RADIUS server: • no - do not use RADIUS; • yes - use RADIUS for accounting and lease; • accounting - use RADIUS for accounting only.

Leases

Sub-menu: `/ip dhcp-server lease`

DHCP server lease submenu is used to monitor and manage server leases. The issued leases are shown here as dynamic entries. You can also add static leases to issue a specific IP address to a particular client (identified by MAC address).

Generally, the DHCP lease is allocated as follows:

- an unused lease is in the "waiting" state
- if a client asks for an IP address, the server chooses one
- if the client receives a statically assigned address, the lease becomes offered, and then bound with the respective lease time
- if the client receives a dynamic address (taken from an IP address pool), the router sends a ping packet and waits for an answer for 0.5 seconds. During this time, the lease is marked testing
- in the case where the address does not respond, the lease becomes offered and then bound with the respective lease time
- in other cases, the lease becomes busy for the lease time (there is a command to retest all busy addresses), and the client's request remains unanswered (the client will try again shortly)

A client may free the leased address. The dynamic lease is removed, and the allocated address is returned to the address pool. But the static lease becomes busy until the client reacquires the address.

⚠ IP addresses assigned statically are not probed!

Property	Description
address (*IP*; Default: **0.0.0.0**)	Specify IP address (or ip pool) for static lease. If set to **0.0.0.0** - a pool from the DHCP server will be used
address-list (*string*; Default: **none**)	Address list to which address will be added if the lease is bound.
allow-dual-stack-queue (*yes* | *no*; Default: **yes**)	Creates a single simple queue entry for both IPv4 and IPv6 addresses, and uses the MAC address and DUID for identification. Requires IPv6 DHCP Server to have this option enabled as well to work properly.
always-broadcast (*yes* | *no*; Default: **no**)	Changes whether to force broadcast DHCP replies: • no - replies are sent based on the client's broadcast flag. If the server sends three consecutive offers, the third and forth offer will be sent as a broadcast; • yes - replies are always broadcasted even when the client has not specified the broadcast flag.
block-access (*yes* | *no*; Default: **no**)	Block access for this client
client-id (*string*; Default: **none**)	If specified, must match the DHCP 'client identifier' option of the request

dhcp-option (*string*; Default: **none**)	Add additional DHCP options from option list.
dhcp-option-set (*string*; Default: **none**)	Add an additional set of DHCP options.
insert-queue-before (*bottom | first | name*; Default: **first**)	Specify where to place dynamic simple queue entries for static DCHP leases with rate-limit parameter set.
lease-time (*time*; Default: **0s**)	Time that the client may use the address. If set to **0s** lease will never expire.
mac-address (*MAC*; Default: **00:00:00:00:00:00**)	If specified, must match the MAC address of the client
parent-queue (*string | none*; Default: **none**)	A dynamically created queue for this lease will be configured as a child queue of the specified parent queue.
queue-type (*default, ethernet-default, multi-queue-ethernet-default, pcq-download-default, synchronous-default, default-small, hotspot-default, only-hardware-queue, pcq-upload-default, wireless-default*)	Queue type that can be assigned to the specific lease
rate-limit (*integer[/integer] [integer[/integer] [integer[/integer] [integer[/integer]]]];*; Default:)	Adds a dynamic simple queue to limit IP's bandwidth to a specified rate. Requires the lease to be static. Format is: rx-rate[/tx-rate] [rx-burst-rate[/tx-burst-rate] [rx-burst-threshold[/tx-burst-threshold] [rx-burst-time[/tx-burst-time]]]]. All rates should be numbers with optional 'k' (1,000s) or 'M' (1,000,000s). If tx-rate is not specified, rx-rate is as tx-rate too. Same goes for tx-burst-rate and tx-burst-threshold and tx-burst-time. If both rx-burst-threshold and tx-burst-threshold are not specified (but burst-rate is specified), rx-rate and tx-rate is used as burst thresholds. If both rx-burst-time and tx-burst-time are not specified, 1s is used as default.
routes ([dst-address/mask] [gateway] [distance]; Default: ***none***)	Routes that appear on the server when the client is connected. It is possible to specify multiple routes separated by commas. This setting will be ignored for OpenVPN.
server (*string*)	Server name which serves this client
use-src-mac (*yes* / *no*; Default: no)	When this option is set server uses the source MAC address instead of the received CHADDR to assign the address.

Menu specific commands

check-status (*id*)	Check the status of a given busy (status is conflict or declined) dynamic lease, and free it in case of no response
make-static (*id*)	Convert a dynamic lease to a static one

Store Configuration

Sub-menu: `/ip dhcp-server config`

Store Leases On Disk: The configuration of how often the DHCP leases will be stored on disk. If they would be saved on a disk on every lease change, a lot of disk writes would happen which is very bad for Compact Flash (especially, if lease times are very short). To minimize writes on disk, all changes are saved on disk every store-leases-disk seconds. Additionally, leases are always stored on disk on graceful shutdown and reboot.

Manual changes to leases - addition/removal of a static lease, removal of a dynamic lease will cause changes to be pushed for this lease to storage.

Accounting: The accounting parameter in the DHCP server configuration enables or disables accounting for DHCP leases. When accounting is enabled, the DHCP server logs information about IP address assignments and lease renewals. This information can be useful for tracking and monitoring network usage, analyzing traffic patterns, or generating reports on IP address allocations.

Interim-update: The interim-update parameter determines whether the DHCP server sends periodic updates to the accounting server during a lease. These updates provide information about the lease duration, usage, and other relevant details. Enabling interim updates allows for more accurate tracking of lease activity.

Radius-password: The radius-password parameter is used to set the password for the RADIUS (Remote Authentication Dial-In User Service) server. RADIUS is a networking protocol commonly used for providing centralized authentication, authorization, and accounting for network access. When configuring the DHCP server to communicate with a RADIUS server for authentication or accounting purposes, you need to specify the correct password to establish a secure connection. This parameter ensures that the DHCP server can authenticate with the RADIUS server using the specified password.

Rate limiting

It is possible to set the bandwidth to a specific IPv4 address by using DHCPv4 leases. This can be done by setting a rate limit on the DHCPv4 lease itself, by doing this a dynamic simple queue rule will be added for the IPv4 address that corresponds to the DHCPv4 lease. By using the *rate-limit* parameter you can conveniently limit a user's bandwidth.

> ⚠ For any queues to work properly, the traffic must not be FastTracked, make sure your Firewall does not FastTrack traffic that you want to limit.

First, make the DHCPv4 lease static, otherwise, it will not be possible to set a rate limit to a DHCPv4 lease:

```
[admin@MikroTik] > /ip dhcp-server lease print
Flags: X - disabled, R - radius, D - dynamic, B - blocked
 #   ADDRESS               MAC-ADDRESS       HOST-NAME              SERVER              RATE-
LIMIT              STATUS
 0 D 192.168.88.254        6C:3B:6B:7C:41:3E MikroTik
DHCPv4_Server                                bound

[admin@MikroTik] > /ip dhcp-server lease make-static 0

[admin@MikroTik] > /ip dhcp-server lease print
Flags: X - disabled, R - radius, D - dynamic, B - blocked
 #   ADDRESS               MAC-ADDRESS       HOST-NAME              SERVER              RATE-
LIMIT              STATUS
 0   192.168.88.254        6C:3B:6B:7C:41:3E MikroTik
DHCPv4_Server                                bound
```

Then you can set a rate to a DHCPv4 lease that will create a new dynamic simple queue entry:

```
[admin@MikroTik] > /ip dhcp-server lease set 0 rate-limit=10M/10M

[admin@MikroTik] > /queue simple print
Flags: X - disabled, I - invalid, D - dynamic
 0  D name="dhcp-ds<6C:3B:6B:7C:41:3E>" target=192.168.88.254/32 parent=none packet-marks="" priority=8/8
queue=default-small/default-small limit-at=10M/10M max-limit=10M/10M burst-limit=0/0 burst-threshold=0/0 burst-
time=0s/0s
      bucket-size=0.1/0.1
```

> ⚠ By default allow-dual-stack-queue is enabled, this will add a single dynamic simple queue entry for both DCHPv6 binding and DHCPv4 lease, without this option enabled separate dynamic simple queue entries will be added for IPv6 and IPv4.

If *allow-dual-stack-queue* is enabled, then a single dynamic simple queue entry will be created containing both IPv4 and IPv6 addresses:

```
[admin@MikroTik] > /queue simple print
Flags: X - disabled, I - invalid, D - dynamic
 0  D name="dhcp-ds<6C:3B:6B:7C:41:3E>" target=192.168.88.254/32,fdb4:4de7:a3f8:418c::/66 parent=none packet-
marks="" priority=8/8 queue=default-small/default-small limit-at=10M/10M max-limit=10M/10M burst-limit=0/0
burst-threshold=0/0
      burst-time=0s/0s bucket-size=0.1/0.1
```

Network

Sub-menu: `/ip dhcp-server network`

Properties

Property	Description
address (*IP /netmask*; Default:)	the network DHCP server(s) will lease addresses from
boot-file-name (*string*; Default:)	Boot filename
caps-manager (*string*; Default:)	A comma-separated list of IP addresses for one or more CAPsMAN system managers. DHCP Option 138 (capwap) will be used.
dhcp-option (*string*; Default:)	Add additional DHCP options from the option list.
dhcp-option-set (*string*; Default:)	Add an additional set of DHCP options.
dns-none (*yes | no*; Default: **no**)	If set, then DHCP Server will not pass dynamic DNS servers configured on the router to the DHCP clients if no DNS Server in DNS-server is set. By default, if there are no DNS servers configured, then the dynamic DNS Servers will be passed to DHCP clients.
dns-server (*string*; Default:)	the DHCP client will use these as the default DNS servers. Two comma-separated DNS servers can be specified to be used by the DHCP client as primary and secondary DNS servers
domain (*string*; Default:)	The DHCP client will use this as the 'DNS domain' setting for the network adapter.
gateway (*IP*; Default: **0.0.0.0**)	The default gateway to be used by DHCP Client.
netmask (*integer: 0..32*; Default: **0**)	The actual network mask is to be used by the DHCP client. If set to '0' - netmask from network address will be used.
next-server (*IP*; Default:)	The IP address of the next server to use in bootstrap.
ntp-server (*IP*; Default:)	the DHCP client will use these as the default NTP servers. Two comma-separated NTP servers can be specified to be used by the DHCP client as primary and secondary NTP servers
wins-server (*IP*; Default:)	The Windows DHCP client will use these as the default WINS servers. Two comma-separated WINS servers can be specified to be used by the DHCP client as primary and secondary WINS servers

RADIUS Support

Since RouterOS v6.43 it is possible to use RADIUS to assign a rate limit per lease, to do so you need to pass the Mikrotik-Rate-Limit attribute from your RADIUS Server for your lease. To achieve this you first need to set your DHCPv4 Server to use RADIUS for assigning leases. Below is an example of how to set it up:

```
/radius
add address=10.0.0.1 secret=VERYsecret123 service=dhcp
/ip dhcp-server
set dhcp1 use-radius=yes
```

After that, you need to tell your RADIUS Server to pass the Mikrotik-Rate-Limit attribute. In case you are using FreeRADIUS with MySQL, then you need to add appropriate entries into **radcheck** and **radreply** tables for a MAC address, that is being used for your DHCPv4 Client. Below is an example for table entries:

Error rendering macro 'code': Invalid value specified for parameter '[Ljava.lang.Object;@7059db6f'

```
INSERT INTO `radcheck` (`username`, `attribute`, `op`, `value`) VALUES
('00:0C:42:00:D4:64', 'Auth-Type', ':=', 'Accept'),

INSERT INTO `radreply` (`username`, `attribute`, `op`, `value`) VALUES
('00:0C:42:00:D4:64', 'Framed-IP-Address', '=', '192.168.88.254'),
('00:0C:42:00:D4:64', 'Mikrotik-Rate-Limit', '=', '10M'),
```

Alerts

To find any rogue DHCP servers as soon as they appear in your network, the DHCP Alert tool can be used. It will monitor the interface for all DHCP replies and check if this reply comes from a valid DHCP server. If a reply from an unknown DHCP server is detected, an alert gets triggered:

```
[admin@MikroTik] ip dhcp-server alert>/log print
00:34:23 dhcp,critical,error,warning,info,debug dhcp alert on Public:
    discovered unknown dhcp server, mac 00:02:29:60:36:E7, ip 10.5.8.236
[admin@MikroTik] ip dhcp-server alert>
```

When the system alerts about a rogue DHCP server, it can execute a custom script.

As DHCP replies can be unicast, the rogue DHCP detector may not receive any offer to other DHCP clients at all. To deal with this, the rogue DHCP detector acts as a DHCP client as well - it sends out DHCP discover requests once a minute.

> ⚠ The DHCP alert is not recommended on devices that are configured as DHCP clients. Since the alert itself generates DHCP discovery packets, it can affect the operation of the DHCP client itself. Use this feature only on devices that are DHCP servers or using a static IP address.

Sub-menu: `/ip dhcp-server alert`

Properties

Property	Description
alert-timeout (none | time; Default: 1h)	Time after which the alert will be forgotten. If after that time the same server is detected, a new alert will be generated. If set to **none** timeout will never expire.
interface (*string*; Default:)	Interface, on which to run rogue DHCP server finder.
on-alert (*string*; Default:)	Script to run, when an unknown DHCP server is detected.
valid-server (*string*; Default:)	List of MAC addresses of valid DHCP servers.

Read-only properties

Property	Description
unknown-server (*string*)	List of MAC addresses of detected unknown DHCP servers. The server is removed from this list after alert-timeout

Menu specific commands

Property	Description
reset-alert (*id*)	Clear all alerts on an interface

DHCP Options

Sub-menu: `/ip dhcp-server option`

With the help of the DHCP Option list, it is possible to define additional custom options for DHCP Server to advertise. Option precedence is as follows:

- radius,
- lease,
- server,
- network.

This is the order in which the client option request will be filled in.

According to the DHCP protocol, a parameter is returned to the DHCP client only if it requests this parameter, specifying the respective code in the DHCP request Parameter-List (code 55) attribute. If the code is not included in the Parameter-List attribute, the DHCP server will not send it to the DHCP client, but **since RouterOS v7.1rc5 it is possible to force the DHCP option** from the server-side even if the DHCP-client does not request such parameter:

```
ip/dhcp-server/option/set force=yes
```

Properties

Property	Description
code (*integer:1..254*; Default:)	dhcp option code. All codes are available at http://www.iana.org/assignments/bootp-dhcp-parameters
name (*string*; Default:)	Descriptive name of the option
value (*string*; Default:)	Parameter's value. Available data types for options are: ○ 'test' -> ASCII to Hex 0x74657374 ○ '10.10.10.10' -> Unicode IP to Hex 0x0a0a0a0a ○ s'10.10.10.10' -> ASCII to hex 0x31302e31302e31302e3130 ○ s'160' -> ASCII to hex 0x313630 ○ '10' -> Decimal to Hex 0x0a ○ 0x0a0a -> No conversion ○ $(VARIABLE) -> hardcoded values RouterOS has predefined variables that can be used: • HOSTNAME - client hostname • RADIUS_MT_STR1 - from radius MT attr nr. 24 • RADIUS_MT_STR2 - from radius MT attr nr. 25 • REMOTE_ID - agent remote-id • NETWORK_GATEWAY - the first gateway from '*/ip dhcp-server network*', note that this option won't work if used from lease Now it is also possible to combine data types into one, for example: "0x01'vards'$(HOSTNAME)" For example if HOSTNAME is 'kvm', then raw value will be 0x0176617264736b766d.
raw-value (*HEX string*)	Read-only field which shows raw DHCP option value (the format actually sent out)

DHCP Option Sets

Sub-menu: `/ip dhcp-server option sets`

This menu allows combining multiple options in option sets, which later can be used to override the default DHCP server option set.

Example

Classless Route

A classless route adds a specified route in the clients routing table. In our example, it will add

- dst-address=160.0.0.0/24 gateway=10.1.101.1
- dst-address=0.0.0.0/0 gateway=10.1.101.1

According to RFC 3442: The first part is the netmask ("18" = netmask /24). Second part is significant part of destination network ("A00000" = 160.0.0). Third part is IP address of gateway ("0A016501" = 10.1.101.1). Then There are parts of the default route, destination netmask (0x00 = 0.0.0.0/0) followed by default route (0x0A016501 = 10.1.101.1)

```
/ip dhcp-server option
add code=121 name=classless value=0x18A000000A016501000A016501
/ip dhcp-server network
set 0 dhcp-option=classless
```

Result:

```
[admin@MikroTik] /ip route> print
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf,
m - mme, B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADS  0.0.0.0/0                          10.1.101.1         0
 1 ADS  160.0.0.0/24                       10.1.101.1         0
```

A much more robust way would be to use built-in variables, the previous example can be rewritten as:

```
/ip dhcp-server option
add name=classless code=121 value="0x18A00000\$(NETWORK_GATEWAY)0x00\$(NETWORK_GATEWAY)"
```

Auto proxy config

```
/ip dhcp-server option
  add code=252 name=auto-proxy-config value="'https://autoconfig.something.lv/wpad.dat'"
```

Vendor Classes

Since the 6.45beta6 version RouterOS support vendor class, ID matcher. The vendor class is used by DHCP clients to optionally identify the vendor and configuration.

> ⚠ Vendor-class-id matcher changes to generic matcher since RouterOS v7.4beta4.

Example

In the following configuration example, we will give an IP address from a particular pool for an Android-based mobile phone. We will use the RouterBOARD with a default configuration

```
/ip pool
add name=default-dhcp ranges=192.168.88.10-192.168.88.254
add name=pool-for-VID ranges=172.16.16.10-172.16.16.120
```

Configure `vendor-class-id` matcher. DHCP servers configuration remains the default

```
/ip dhcp-server
add address-pool=default-dhcp disabled=no interface=bridge name=defconf
/ip dhcp-server network
add address=192.168.88.0/24 comment=defconf gateway=192.168.88.1
/ip dhcp-server vendor-class-id
add address-pool=pool-for-VID name=samsung server=defconf vid=android-dhcp-9
```

Connect your mobile phone to the device to receive an IP address from the 172.16.16.0 network

```
[admin@mikrotik] > /ip dhcp-server lease print detail
Flags: X - disabled, R - radius, D - dynamic, B - blocked
 0 D address=172.16.16.120 mac-address=30:07:4D:F5:07:49 client-id="1:30:7:4d:f5:7:49" address-lists=""
server=defconf dhcp-option=""
     status=bound expires-after=8m55s last-seen=1m5s active-address=172.16.16.120 active-mac-address=30:07:4D:
F5:07:49
     active-client-id="1:30:7:4d:f5:7:49" active-server=defconf host-name="Galaxy-S8"
```

If you do not know your devices Vendor Class ID, you can turn on DHCP debug logs with `/system logging add topics=dhcp`. Then in the logging entries, you will see **Class-ID**

```
10:30:31 dhcp,debug,packet defconf received request with id 4238230732 from 0.0.0.0
10:30:31 dhcp,debug,packet     secs = 3
10:30:31 dhcp,debug,packet     ciaddr = 0.0.0.0
10:30:31 dhcp,debug,packet     chaddr = 30:07:4D:F5:07:49
10:30:31 dhcp,debug,packet     Msg-Type = request
10:30:31 dhcp,debug,packet     Client-Id = 01-30-07-4D-F5-07-49
10:30:31 dhcp,debug,packet     Address-Request = 172.16.16.120
10:30:31 dhcp,debug,packet     Server-Id = 192.168.88.1
10:30:31 dhcp,debug,packet     Max-DHCP-Message-Size = 1500
10:30:31 dhcp,debug,packet     Class-Id = "android-dhcp-9"
10:30:31 dhcp,debug,packet     Host-Name = "Galaxy-S8"
10:30:31 dhcp,debug,packet     Parameter-List = Subnet-Mask,Router,Domain-Server,Domain-Name,Interface-MTU,
Broadcast-Address,Address-Time,Ren
ewal-Time,Rebinding-Time,Vendor-Specific
10:30:31 dhcp,info defconf assigned 172.16.16.120 to 30:07:4D:F5:07:49
10:30:31 dhcp,debug,packet defconf sending ack with id 4238230732 to 172.16.16.120
10:30:31 dhcp,debug,packet     ciaddr = 0.0.0.0
10:30:31 dhcp,debug,packet     yiaddr = 172.16.16.120
10:30:31 dhcp,debug,packet     siaddr = 192.168.88.1
10:30:31 dhcp,debug,packet     chaddr = 30:07:4D:F5:07:49
10:30:31 dhcp,debug,packet     Msg-Type = ack
10:30:31 dhcp,debug,packet     Server-Id = 192.168.88.1
10:30:31 dhcp,debug,packet     Address-Time = 600
10:30:31 dhcp,debug,packet     Domain-Server = 192.168.88.1,10.155.0.1,10.155.0.126
```

Generic matcher

Since RouterOS 7.4beta4 (2022-Jun-15 14:04) the vendor-id matcher is converted to a generic matcher. The genric matcher allows matching any of the DHCP options.

And an example to match DHCP option 60 similar to vendor-id-class matcher:

```
/ip dhcp-server matcher
add address-pool=pool1 code=60 name=test value=android-dhcp-11
```

Match the client-id with option 61 configured as hex value:

```
/ip dhcp-server matcher
add address-pool=pool1 code=61 name=test value=0x016c3b6bed8364
```

Match the code 12 using the string:

```
/ip dhcp-server matcher
add address-pool=testpool code=12 name=test server=dhcp1 value="MikroTik"
```

Configuration Examples

Setup

To simply configure DHCP server you can use a `setup` command.

First, you configure an IP address on the interface:

```
[admin@MikroTik] > /ip address add address=192.168.88.1/24 interface=ether3 disabled=no
```

Then you use `setup` a command which will automatically ask necessary parameters:

```
[admin@MikroTik] > /ip dhcp-server setup
Select interface to run DHCP server on

dhcp server interface: ether3
Select network for DHCP addresses

dhcp address space: 192.168.88.0/24
Select gateway for given network

gateway for dhcp network: 192.168.88.1
Select pool of ip addresses given out by DHCP server

addresses to give out: 192.168.88.2-192.168.88.254
Select DNS servers

dns servers: 10.155.126.1,10.155.0.1,
Select lease time

lease time: 10m
```

That is all. You have configured an active DHCP server.

Manual configuration

To configure the DHCP server manually to respond to local requests you have to configure the following:

- An **IP pool** for addresses to be given out, make sure that your gateway/DHCP server address is not part of the pool.

```
/ip pool add name=dhcp_pool0 ranges=192.168.88.2-192.168.88.254
```

- A **network** indicating subnets that DHCP-server will lease addresses from, among other information, like a gateway, DNS-server, NTP-server, DHCP options, etc.

```
/ip dhcp-server network add address=192.168.88.0/24 dns-server=192.168.88.1 gateway=192.168.88.1
```

- In our case, the device itself is serving as the gateway, so we'll add the **address** to the bridge interface:

```
/ip address add address=192.168.88.1/24 interface=bridge1 network=192.168.88.0
```

- And finally, add **DHCP Server**, here we will add the previously created address **pool**, and specify on which **interface** the DHCP server should work on

```
/ip dhcp-server add address-pool=dhcp_pool0 disabled=no interface=bridge1 name=dhcp1
```

DHCPv6 Server

Summary

Standards: `RFC 3315, RFC 3633`

Single DUID is used for client and server identification, only IAID will vary between clients corresponding to their assigned interface.

Client binding creates a dynamic pool with a timeout set to binding's expiration time (note that now dynamic pools can have a timeout), which will be updated every time binding gets renewed.

When a client is bound to a prefix, the DHCP server adds routing information to know how to reach the assigned prefix.

Client bindings in the server do not show MAC address anymore (as it was in v5.8), DUID (hex) and IAID are used instead. After upgrade, MAC addresses will be converted to DUIDs automatically, but due to unknown DUID type and unknown IAID, they should be further updated by the user;

RouterOS DHCPv6 server can only delegate IPv6 prefixes, not addresses.

General

Sub-menu: `/ipv6 dhcp-server`

This sub-menu lists and allows to configure DHCP-PD servers.

DHCPv6 Server Properties

Property	Description
address-pool (*enum* / *static-only*; Default: **static-only**)	IPv6 pool, from which to take IPv6 prefix for the clients.
allow-dual-stack-queue (*yes* / *no*; Default: **yes**)	Creates a single simple queue entry for both IPv4 and IPv6 addresses, and uses the MAC address and DUID for identification. Requires IPv6 DHCP Server to have this option enabled as well to work properly.
binding-script (*string*; Default:)	A script that will be executed after binding is assigned or de-assigned. Internal "global" variables that can be used in the script: • bindingBound - set to "1" if bound, otherwise set to "0" • bindingServerName - dhcp server name • bindingDUID - DUID • bindingAddress - active address • bindingPrefix - active prefix
dhcp-option (*string*; Default: **none**)	Add additional DHCP options from option list.
disabled (*yes* / *no*; Default: **no**)	Whether DHCP-PD server participates in the prefix assignment process.
interface (*string*; Default:)	The interface on which server will be running.
lease-time (*time*; Default: **3d**)	The time that a client may use the assigned address. The client will try to renew this address after half of this time and will request a new address after the time limit expires.
name (*string*; Default:)	Reference name

Read-only Properties

Property	Description

dynamic (*yes* | *no*)	
invalid (*yes* | *no*)	

Bindings

Sub-menu: `/ipv6 dhcp-server binding`

DUID is used only for dynamic bindings, so if it changes then the client will receive a different prefix than previously.

Property	Description
address (*IPv6 prefix*; Default:)	IPv6 prefix that will be assigned to the client
allow-dual-stack-queue (*yes* | *no*; Default: **yes**)	Creates a single simple queue entry for both IPv4 and IPv6 addresses, uses the MAC address and DUID for identification. Requires IPv4 DHCP Server to have this option enabled as well to work properly.
comment (*string*; Default:)	Short description of an item.
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled
dhcp-option (*string*; Default:)	Add additional DHCP options from the option list.
dhcp-option-set (*string*; Default:)	Add an additional set of DHCP options.
life-time (*time*; Default: **3d**)	The time period after which binding expires.
duid (*hex string*; Default:)	DUID value. Should be specified only in hexadecimal format.
iaid (*integer [0..4294967295]*; Default:)	Identity Association Identifier, part of the Client ID.
prefix-pool (*string*; Default:)	Prefix pool that is being advertised to the DHCPv6 Client.
rate-limit (*integer [/integer] [integer [/integer] [integer [/integer] [integer [/integer]]]]*; Default:)	Adds a dynamic simple queue to limit IP's bandwidth to a specified rate. Requires the lease to be static. Format is: rx-rate[/tx-rate] [rx-burst-rate[/tx-burst-rate] [rx-burst-threshold[/tx-burst-threshold] [rx-burst-time[/tx-burst-time]]]]. All rates should be numbers with optional 'k' (1,000s) or 'M' (1,000,000s). If tx-rate is not specified, rx-rate is as tx-rate too. Same goes for tx-burst-rate and tx-burst-threshold and tx-burst-time. If both rx-burst-threshold and tx-burst-threshold are not specified (but burst-rate is specified), rx-rate and tx-rate is used as burst thresholds. If both rx-burst-time and tx-burst-time are not specified, 1s is used as default.
server (*string* | *all*; Default: **all**)	Name of the server. If set to **all**, then binding applies to all created DHCP-PD servers.

Read-only properties

Property	Description
dynamic (*yes* | *no*)	Whether an item is dynamically created.
expires-after (*time*)	The time period after which binding expires.
last-seen (*time*)	Time period since the client was last seen.

status (*waiting | offered | bound*)	Three status values are possible: • **waiting** - Shown for static bindings if it is not used. For dynamic bindings this status is shown if it was used previously, the server will wait 10 minutes to allow an old client to get this binding, otherwise binding will be cleared and prefix will be offered to other clients. • **offered** - if **solicit** message was received, and the server responded with **advertise** a message, but the **request** was not received. During this state client have 2 minutes to get this binding, otherwise, it is freed or changed status to **waiting** for static bindings. • **bound** - currently bound.

For example, dynamically assigned /62 prefix

```
[admin@RB493G] /ipv6 dhcp-server binding> print detail
 Flags: X - disabled, D - dynamic
0 D address=2a02:610:7501:ff00::/62 duid="1605fcb400241d1781f7" iaid=0
 server=local-dhcp life-time=3d status=bound expires-after=2d23h40m10s
last-seen=19m50s
1 D address=2a02:610:7501:ff04::/62 duid="0019d1393535" iaid=2
server=local-dhcp life-time=3d status=bound expires-after=2d23h43m47s
last-seen=16m13s
```

Menu specific commands

Property	Description
make-static ()	Set dynamic binding as static.

Rate limiting

It is possible to set the bandwidth to a specific IPv6 address by using DHCPv6 bindings. This can be done by setting a rate limit on the DHCPv6 binding itself, by doing this a dynamic simple queue rule will be added for the IPv6 address that corresponds to the DHCPv6 binding. By using the `rate-limit` the parameter you can conveniently limit a user's bandwidth.

For any queues to work properly, the traffic must not be FastTracked, make sure your Firewall does not FastTrack traffic that you want to limit.

First, make the DHCPv6 binding static, otherwise, it will not be possible to set a rate limit to a DHCPv6 binding:

```
[admin@MikroTik] > /ipv6 dhcp-server binding print
Flags: X - disabled, D - dynamic
# ADDRESS DUID SERVER STATUS
0 D fdb4:4de7:a3f8:418c::/66 0x6c3b6b7c413e DHCPv6_Server bound

[admin@MikroTik] > /ipv6 dhcp-server binding make-static 0

[admin@MikroTik] > /ipv6 dhcp-server binding print
Flags: X - disabled, D - dynamic
# ADDRESS DUID SERVER STATUS
0 fdb4:4de7:a3f8:418c::/66 0x6c3b6b7c413e DHCPv6_Server bound
```

Then you need can set a rate to a DHCPv6 binding that will create a new dynamic simple queue entry:

```
[admin@MikroTik] > /ipv6 dhcp-server binding set 0 rate-limit=10M/10
[admin@MikroTik] > /queue simple print
Flags: X - disabled, I - invalid, D - dynamic
0 D name="dhcp<6c3b6b7c413e fdb4:4de7:a3f8:418c::/66>" target=fdb4:4de7:a3f8:418c::/66 parent=none packet-
marks="" priority=8/8 queue=default
-small/default-small limit-at=10M/10M max-limit=10M/10M burst-limit=0/0
burst-threshold=0/0 burst-time=0s/0s bucket-size=0.1/0.1
```

⚠ By default `allow-dual-stack-queue` is enabled, this will add a single dynamic simple queue entry for both DCHPv6 binding and DHCPv4 lease, without this option enabled separate dynamic simple queue entries will be added for IPv6 and IPv4.

If `allow-dual-stack-queue` is enabled, then a single dynamic simple queue entry will be created containing both IPv4 and IPv6 addresses:

```
[admin@MikroTik] > /queue simple print
Flags: X - disabled, I - invalid, D - dynamic
 0 D name="dhcp-ds<6C:3B:6B:7C:41:3E>" target=192.168.1.200/32,fdb4:4de7:a3f8:418c::/66 parent=none packet-
marks="" priority=8/8 queue=default
-small/default-small limit-at=10M/10M max-limit=10M/10M
burst-limit=0/0 burst-threshold=0/0 burst-time=0s/0s bucket-size=0.1/0.1
```

RADIUS Support

Since RouterOS v6.43 it is possible to use RADIUS to assign a rate-limit per DHCPv6 binding, to do so you need to pass the Mikrotik-Rate-Limit attribute from your RADIUS Server for your DHCPv6 binding. To achieve this you first need to set your DHCPv6 Server to use RADIUS for assigning bindings. Below is an example of how to set it up:

```
/radius
add address=10.0.0.1 secret=VERYsecret123 service=dhcp
/ipv6 dhcp-server
set dhcp1 use-radius=yes
```

After that, you need to tell your RADIUS Server to pass the Mikrotik-Rate-Limit attribute. In case you are using FreeRADIUS with MySQL, then you need to add appropriate entries into **radcheck** and **radreply** tables for a MAC address, that is being used for your DHCPv6 Client. Below is an example for table entries:

```
INSERT INTO `radcheck` (`username`, `attribute`, `op`, `value`) VALUES
('000c4200d464', 'Auth-Type', ':=', 'Accept'),
 INSERT INTO `radreply` (`username`, `attribute`, `op`, `value`) VALUES
('000c4200d464', 'Delegated-IPv6-Prefix', '=', 'fdb4:4de7:a3f8:418c::/66'),
('000c4200d464', 'Mikrotik-Rate-Limit', '=', '10M');
```

⚠ By default allow-dual-stack-queue is enabled and will add a single dynamic queue entry if the MAC address from the IPv4 lease (or DUID, if the DHCPv4 Client supports `Node-specific Client Identifiers` from RFC4361), but DUID from DHCPv6 Client is not always based on the MAC address from the interface on which the DHCPv6 client is running on, DUID is generated on a per-device basis. For this reason, a single dynamic queue entry might not be created, separate dynamic queue entries might be created instead.

Configuration Example

Enabling IPv6 Prefix delegation

Let's consider that we already have a running DHCP server.

To enable IPv6 prefix delegation, first, we need to create an address pool:

```
/ipv6 pool add name=myPool prefix=2001:db8:7501::/60 prefix-length=62
```

Notice that prefix-length is 62 bits, which means that clients will receive /62 prefixes from the /60 pool.

The next step is to enable DHCP-PD:

```
/ipv6 dhcp-server add name=myServer address-pool=myPool interface=local
```

To test our server we will set up wide-dhcpv6 on an ubuntu machine:

- install wide-dhcpv6-client
- edit "/etc/wide-dhcpv6/dhcp6c.conf" as above

You can use also RouterOS as a DHCP-PD client.

```
interface eth2{
send ia-pd 0;
};

id-assoc pd {
prefix-interface eth3{
sla-id 1;
sla-len 2;
};
};
```

- Run DHCP-PD client:

```
sudo dhcp6c -d -D -f eth2
```

- Verify that prefix was added to the:

```
mrz@bumba:/media/aaa$ ip -6 addr
 ..
2: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
 inet6 2001:db8:7501:1:200:ff:fe00:0/64 scope global
 valid_lft forever preferred_lft forever
 inet6 fe80::224:1dff:fe17:81f7/64 scope link
 valid_lft forever preferred_lft forever
```

- You can make binding to specific client static so that it always receives the same prefix:

```
[admin@RB493G] /ipv6 dhcp-server binding> print
Flags: X - disabled, D - dynamic
# ADDRESS DU IAID SER.. STATUS 0 D 2001:db8:7501:1::/62 16 0 loc.. bound
[admin@RB493G] /ipv6 dhcp-server binding> make-static 0
```

- DHCP-PD also installs a route to assigned prefix into IPv6 routing table:

```
[admin@RB493G] /ipv6 route> print
 Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, o - ospf, b - bgp, U -
unreachable
 # DST-ADDRESS GATEWAY DISTANCE
...
2 ADS 2001:db8:7501:1::/62 fe80::224:1dff:fe17:8... 1
```

DHCP Relay

Summary

Sub-menu: `/ip dhcp-relay`

The purpose of the DHCP relay is to act as a proxy between DHCP clients and the DHCP server. It is useful in networks where the DHCP server is not on the same broadcast domain as the DHCP client.

DHCP relay does not choose the particular DHCP server in the DHCP-server list, it just sends the incoming request to all the listed servers.

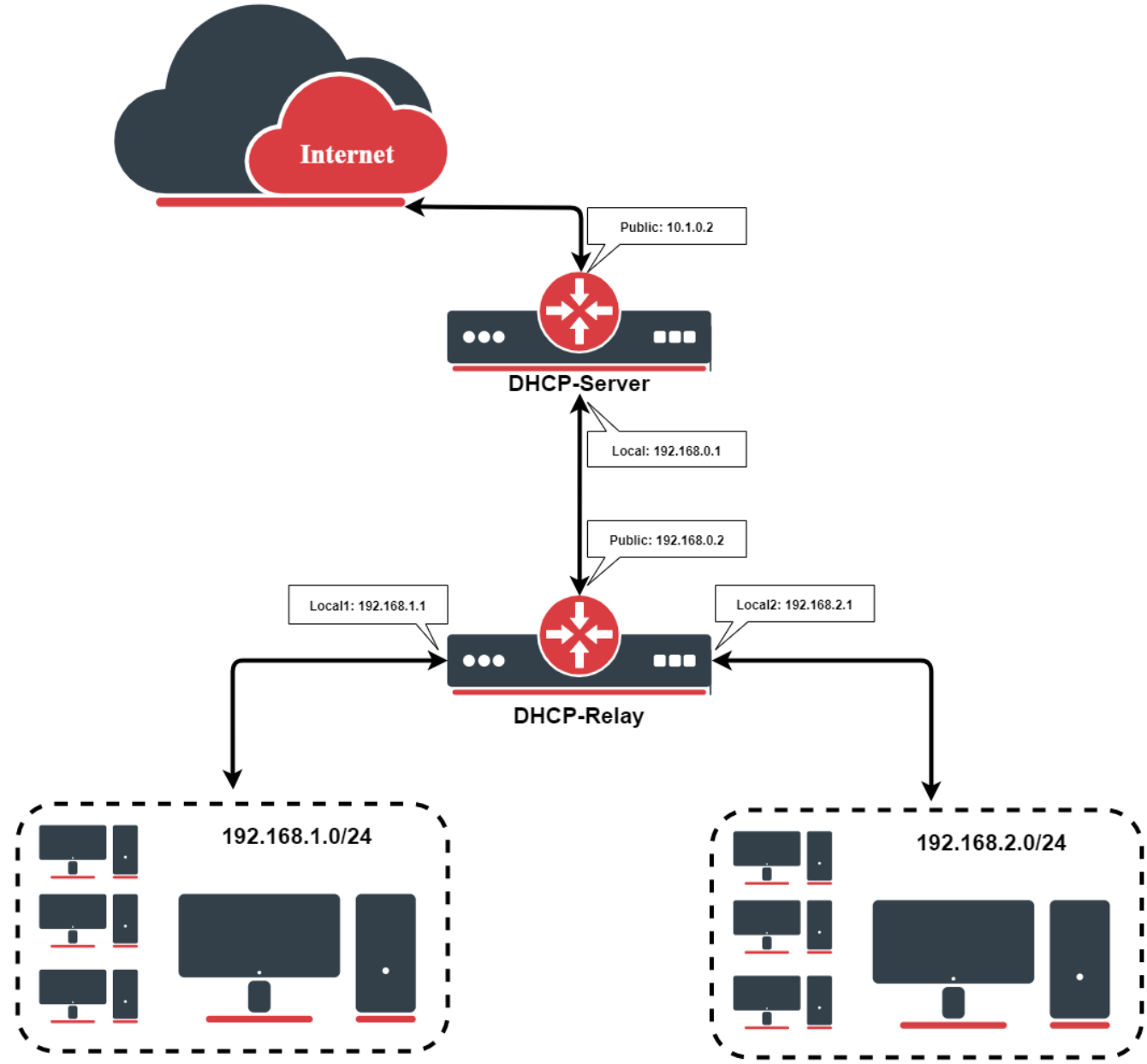
Properties

Property	Description
add-relay-info (*yes* / *no*; Default: **no**)	Adds DHCP relay agent information if enabled according to RFC 3046. Agent Circuit ID Sub-option contains mac address of an interface, Agent Remote ID Sub-option contains MAC address of the client from which request was received.
delay-threshold (*time* / *none*; Default: **none**)	If secs field in DHCP packet is smaller than delay-threshold, then this packet is ignored
dhcp-server (*string*; Default:)	List of DHCP servers' IP addresses which should the DHCP requests be forwarded to
interface (*string*; Default:)	Interface name the DHCP relay will be working on.
local-address (*IP*; Default: **0.0.0.0**)	The unique IP address of this DHCP relay needed for DHCP server to distinguish relays. If set to **0.0.0.0** - the IP address will be chosen automatically
relay-info-remote-id (*string*; Default:)	specified string will be used to construct Option 82 instead of client's MAC address. Option 82 consist of: interface from which packets was received + client mac address or **relay-info-remote-id**
name (*string*; Default:)	Descriptive name for the relay

Configuration Example

Let us consider that you have several IP networks 'behind' other routers, but you want to keep all DHCP servers on a single router. To do this, you need a DHCP relay on your network which will relay DHCP requests from clients to the DHCP server.

This example will show you how to configure a DHCP server and a DHCP relay that serves 2 IP networks - 192.168.1.0/24 and 192.168.2.0/24 that are behind a router DHCP-Relay.



IP Address Configuration

IP addresses of DHCP-Server:

```
[admin@DHCP-Server] ip address> print
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         BROADCAST       INTERFACE
 0   192.168.0.1/24     192.168.0.0     192.168.0.255   To-DHCP-Relay
 1   10.1.0.2/24        10.1.0.0        10.1.0.255        Public
[admin@DHCP-Server] ip address>
```

IP addresses of DHCP-Relay:

```
[admin@DHCP-Relay] ip address> print
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         BROADCAST       INTERFACE
 0   192.168.0.2/24     192.168.0.0     192.168.0.255   To-DHCP-Server
 1   192.168.1.1/24     192.168.1.0     192.168.1.255   Local1
 2   192.168.2.1/24     192.168.2.0     192.168.2.255   Local2
[admin@DHCP-Relay] ip address>
```

DHCP Server Setup

To setup 2 DHCP Servers on the DHCP-Server router add 2 pools. For networks 192.168.1.0/24 and 192.168.2.0:

```
/ip pool add name=Local1-Pool ranges=192.168.1.11-192.168.1.100
/ip pool add name=Local2-Pool ranges=192.168.2.11-192.168.2.100
[admin@DHCP-Server] ip pool> print
 # NAME                                    RANGES
 0 Local1-Pool                             192.168.1.11-192.168.1.100
 1 Local2-Pool                             192.168.2.11-192.168.2.100
[admin@DHCP-Server] ip pool>
```

Create DHCP Servers:

```
/ip dhcp-server add interface=To-DHCP-Relay relay=192.168.1.1 \
   address-pool=Local1-Pool name=DHCP-1 disabled=no
/ip dhcp-server add interface=To-DHCP-Relay relay=192.168.2.1 \
   address-pool=Local2-Pool name=DHCP-2 disabled=no
[admin@DHCP-Server] ip dhcp-server> print
Flags: X - disabled, I - invalid
 #   NAME         INTERFACE     RELAY           ADDRESS-POOL LEASE-TIME ADD-ARP
 0   DHCP-1       To-DHCP-Relay 192.168.1.1     Local1-Pool  3d00:00:00
 1   DHCP-2       To-DHCP-Relay 192.168.2.1     Local2-Pool  3d00:00:00
[admin@DHCP-Server] ip dhcp-server>
```

Configure respective networks:

```
/ip dhcp-server network add address=192.168.1.0/24 gateway=192.168.1.1 \
   dns-server=159.148.60.20
/ip dhcp-server network add address=192.168.2.0/24 gateway=192.168.2.1 \
   dns-server 159.148.60.20
[admin@DHCP-Server] ip dhcp-server network> print
 # ADDRESS            GATEWAY         DNS-SERVER      WINS-SERVER     DOMAIN
 0 192.168.1.0/24     192.168.1.1     159.148.60.20
 1 192.168.2.0/24     192.168.2.1     159.148.60.20
[admin@DHCP-Server] ip dhcp-server network>
```

DHCP Relay Config

Configuration of DHCP-Server is done. Now let's configure DHCP-Relay:

```
/ip dhcp-relay add name=Local1-Relay interface=Local1 \
   dhcp-server=192.168.0.1 local-address=192.168.1.1 disabled=no
/ip dhcp-relay add name=Local2-Relay interface=Local2 \
   dhcp-server=192.168.0.1 local-address=192.168.2.1 disabled=no
[admin@DHCP-Relay] ip dhcp-relay> print
Flags: X - disabled, I - invalid
 #   NAME                        INTERFACE      DHCP-SERVER     LOCAL-ADDRESS
 0   Local1-Relay                Local1         192.168.0.1     192.168.1.1
 1   Local2-Relay                Local2         192.168.0.1     192.168.2.1
[admin@DHCP-Relay] ip dhcp-relay>
```

DHCP Relay with VRF (introduced in 7.15)

Let's take the previous setup but we'll consider that the interface to the DHCP server and interfaces to DHCP clients are added in VRF:

```
/ip vrf
add interfaces=To-DHCP-Server name=vrf_server
add interfaces=Local2 name=vrf2
add interfaces=Local1 name=vrf1
```

In the DHCP-relay configuration dhcp-server-vrf should be added:

```
/ip dhcp-relay/set dhcp-server-vrf=vrf_server numbers=0,1
```

Due to VRF configuration there are several routing-tables - we should add additional routes:

```
/ip route
add disabled=no distance=1 dst-address=192.168.0.0/24 gateway=To-DHCP-Server@vrf_server pref-src="" routing-
table=vrf1 scope=10 suppress-hw-offload=no \
    target-scope=10
add disabled=no distance=1 dst-address=192.168.0.0/24 gateway=To-DHCP-Server@vrf_server pref-src="" routing-
table=vrf2 scope=10 suppress-hw-offload=no \
    target-scope=10
add disabled=no dst-address=192.168.1.0/24 gateway=Local1@vrf1 routing-table=vrf_server suppress-hw-offload=no
add disabled=no distance=1 dst-address=192.168.2.0/24 gateway=Local2@vrf2 pref-src="" routing-table=vrf_server
scope=30 suppress-hw-offload=no \
    target-scope=10
```

To achieve successful DHCP-server - DHCP-relay communication we should add NAT rules:

```
/ip firewall nat
add action=dst-nat chain=dstnat dst-address=192.168.2.1 dst-port=67 in-interface=To-DHCP-Server protocol=udp
src-address=192.168.0.1 to-addresses=\
    192.168.0.2
add action=dst-nat chain=dstnat dst-address=192.168.1.1 dst-port=67 in-interface=To-DHCP-Server protocol=udp
src-address=192.168.0.1 to-addresses=\
    192.168.0.2
```

SOCKS

Introduction

SOCKS (Socket Secure) is a proxy server that allows TCP-based application data to relay across the firewall, even if the firewall would block the packets. The SOCKS protocol is independent of application protocols, so it can be used for many services, e.g, WWW, FTP, TELNET, and others.

At first, an application client connects to the SOCKS proxy server, then the proxy server looks in its access list to see whether the client is permitted to access the remote application resource or not, if it is permitted, the proxy server relies on the packet to the application server and creates a connection between the application server and client.

Remember to configure your application client to use SOCKS!.

You should secure the SOCKS proxy using its access list and/or firewall to disallow access from outside. Failing to secure the proxy server may introduce security issues to your network, and may provide a way for spammers to send junk mail through the router.

Property Description

Property	Description
connection-idle-timeout (time; default: 2m)	time after which idle connections are terminated
enabled (yes | no; default: no)	whether to enable or no the SOCKS proxy
max-connections (integer: 1..500; default: 200)	maxumum number of simultaneous connections
port (integer: 1..65535; default: 1080)	TCP port on which the SOCKS server listens for connections

Access List

```
/ip socks access
```

In the SOCKS access list, you can add rules which will control access to the SOCKS server. This list is similar to firewall lists.

Property	Description
action (allow | deny; default: allow)	allow - allow packets, matching this rule, to be forwarded for further processing deny - deny access for packets, matching this rule
dst-address (IP address/netmask)	destination (server's) address
dst-port (port)	destination TCP port
src-address (IP address/netmask)	source (client's) address for a packet
src-port (port)	source TCP port

Active Connections

The Active Connection list shows all established TCP connections, which are maintained through the SOCKS proxy server.

```
/ip socks connections
```

Property	Description
dst-address (read-only: IP address)	destination (application server) IP address
rx (read-only: integer)	bytes received
src-address (read-only: IP address)	source (application client) IP address
tx (read-only: integer)	bytes sent
type (read-only: in | out | unknown) - connection type	in - incoming connection out - outgoing connection unknown - connection has just been initiated

Example

To see current TCP connections:

```
[admin@MikroTik] ip socks connections> print
 # SRC-ADDRESS               DST-ADDRESS               TX         RX
 0 192.168.0.2:3242          159.148.147.196:80        4847       2880
 1 192.168.0.2:3243          159.148.147.196:80        3408       2127
 2 192.168.0.2:3246          159.148.95.16:80          10172      25207
 3 192.168.0.2:3248          194.8.18.26:80            474        1629
 4 192.168.0.2:3249          159.148.95.16:80          6477       18695
 5 192.168.0.2:3250          159.148.95.16:80          4137       27568
 6 192.168.0.2:3251          159.148.95.16:80          1712       14296
 7 192.168.0.2:3258          80.91.34.241:80           314        208
 8 192.168.0.2:3259          80.91.34.241:80           934        524
 9 192.168.0.2:3260          80.91.34.241:80           930        524
10 192.168.0.2:3261          80.91.34.241:80           312        158
11 192.168.0.2:3262          80.91.34.241:80           312        158
[admin@MikroTik] ip socks connections>
```

Application Examples

FTP service through SOCKS server

Let us consider that we have a network 192.168.0.0/24 which is masqueraded, using a router with a public IP 10.1.0.104/24 and a private IP 192.168.0.1 /24. Somewhere in the network is an FTP server with IP address 10.5.8.8. We want to allow access to this FTP server for a client in our local network with IP address 192.168.0.2/24.

We have already masqueraded our local network:

```
[admin@MikroTik] ip firewall nat> print
Flags: X - disabled, I - invalid, D - dynamic
 0   chain=srcnat action=masquerade src-address=192.168.0.0/24
[admin@MikroTik] ip firewall nat>
```

And access to public FTP servers is denied in the firewall:

```
[admin@MikroTik] ip firewall filter> print
Flags: X - disabled, I - invalid, D - dynamic
 0   chain=forward action=drop src-address=192.168.0.0/24 dst-port=21 protocol=tcp
[admin@MikroTik] ip firewall filter>
```

We have to enable the SOCKS server:

```
[admin@MikroTik] ip socks> set enabled=yes
[admin@MikroTik] ip socks> print
                   enabled: yes
                      port: 1080
    connection-idle-timeout: 2m
           max-connections: 200
[admin@MikroTik] ip socks>
```

Add access to a client with an IP address 192.168.0.2/32 to SOCKS access list, allow data transfer from FTP server to client (allow destination ports from 1024 to 65535 for any IP address), and drop everything else:

```
[admin@MikroTik] ip socks access> add src-address=192.168.0.2 dst-port=21 \
\... action=allow
[admin@MikroTik] ip socks access> add dst-port=1024-65535 action=allow
[admin@MikroTik] ip socks access> add action=deny
[admin@MikroTik] ip socks access> print
Flags: X - disabled
 0   src-address=192.168.0.2 dst-port=21 action=allow
 1   dst-port=1024-65535 action=allow
 2   action=deny
[admin@MikroTik] ip socks access>
```

That's all - the SOCKS server is configured. To see active connections and data transmitted and received:

```
[admin@MikroTik] ip socks connections> print
 # SRC-ADDRESS               DST-ADDRESS               TX         RX
 0 192.168.0.2:1238          10.5.8.8:21               1163       4625
 1 192.168.0.2:1258          10.5.8.8:3423             0          3231744
[admin@MikroTik] ip socks connections>
```


In order to use the SOCKS proxy server, you have to specify its IP address and port in your FTP client. In this case, IP address would be 192.168.0.1 (local IP address of the router/SOCKS server) and TCP port 1080.

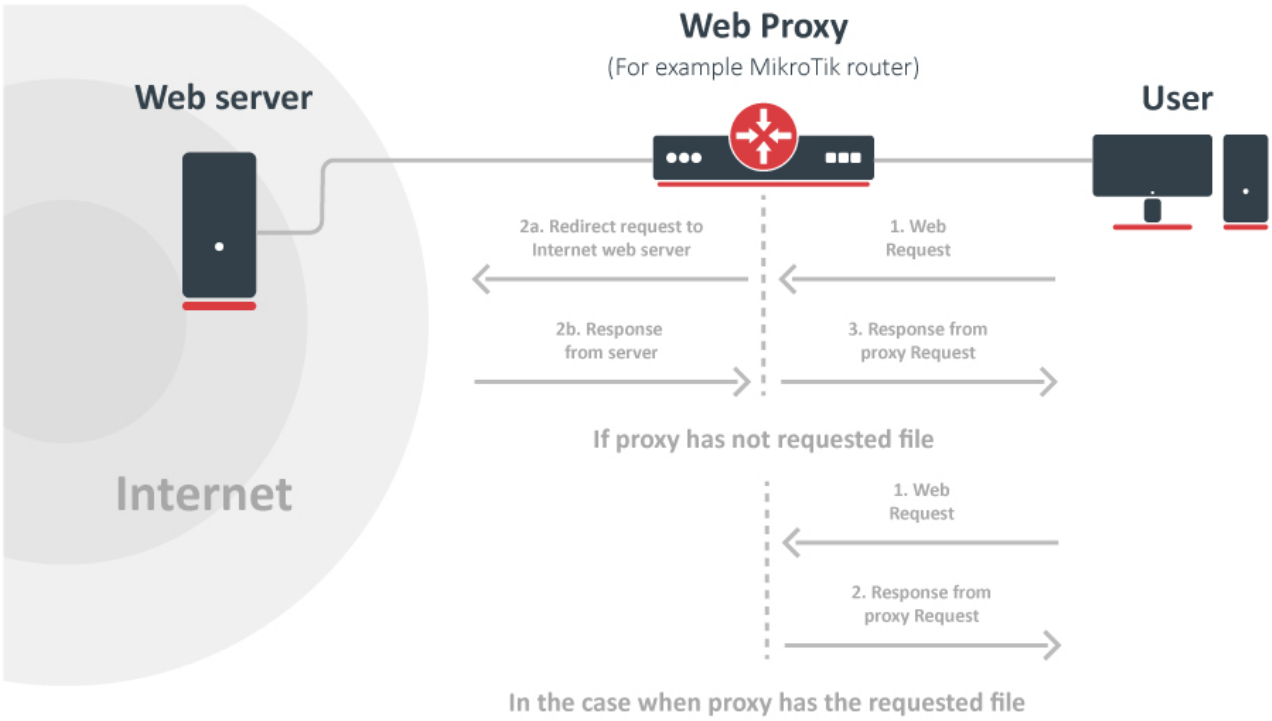
Proxy

Summary

MikroTik RouterOS performs proxying of HTTP and HTTP-proxy (for FTP and HTTP protocols) requests. The proxy server performs the Internet object cache function by storing requested Internet objects, i.e., data available via HTTP and FTP protocols on a system positioned closer to the recipient in the form of speeding up customer browsing by delivering them requested file copies from the proxy cache at local network speed. MikroTik RouterOS implements the following proxy server features:

- Regular HTTP proxy – customer (itself) specifies what is a proxy server for him;
- Transparent proxy – the customer does not know about the proxy being enabled and there isn't a necessity for any additional configuration for the web browser of the client;
- Access list by source, destination, URL, and requested method (HTTP firewall);
- Cache access list to specify which objects to cache, and which not;
- Direct Access List – to specify which resources should be accessed directly, and which - through another proxy server;
- Logging facility – allows to get and store information about the proxy operation;
- Parent proxy support – allows to specify another proxy server, *(if they don't have the requested object ask their parents, or to the original server);*

A proxy server usually is placed at various points between users and the destination server (*also known as the origin server*) on the Internet.



A *Web proxy (cache)* watches requests coming from clients, saving copies of the responses for itself. Then, if there is another request for the same URL, it can use the response that it has, instead of asking the origin server for it again. If the proxy has not requested a file, it downloads that from the original server.

There can be many potential purposes of proxy servers:

- To increase access speed to resources (it takes less time for the client to get the object);
- Works as HTTP firewall (deny access to undesirable web pages);

Allows filtering web content (by specific parameters, like source address, a destination address, port, URL, HTTP request method) scan outbound content, e.g., for data leak protection.

> ⚠ It may be useful to have a Web proxy running even with no cache when you want to use it only as something like an HTTP and FTP firewall (for example, denying access to undesired web pages or denying a specific type of files e.g. .mp3 files) or to redirect requests to external proxy (possibly, to a proxy with caching functions) transparently.

Configuration examples

```
/ip/proxy
```

In MikroTik RouterOS, a proxy configuration is performed in the */ip/proxy* menu. See below how to enable the proxy on port 8080 and set up 192.168.88.254 as the proxy source address:

```
[admin@MikroTik] > ip/proxy/set enabled=yes port=8080 src-address=192.168.88.254
[admin@MikroTik] > ip/proxy/print
                  enabled: yes
              src-address: 192.168.88.254
                     port: 8080
                anonymous: no
             parent-proxy: ::
        parent-proxy-port: 0
      cache-administrator: webmaster
          max-cache-size: unlimited
    max-cache-object-size: 2048KiB
           cache-on-disk: no
    max-client-connections: 600
    max-server-connections: 600
          max-fresh-time: 3d
     serialize-connections: no
         always-from-cache: no
            cache-hit-dscp: 4
                cache-path: web-proxy
```

When setting up a regular proxy service, make sure it serves only your clients and prevents unauthorized access to it by creating a firewall that allows only your clients to use a proxy, otherwise, it may be used as an open proxy.

Transparent proxy configuration example

RouterOS can also act as a Transparent Caching server, with no configuration required in the customer's web browser. A transparent proxy does not modify the requested URL or response. RouterOS will take all HTTP requests and redirect them to the local proxy service. This process will be entirely transparent to the user (users may not know anything about a proxy server that is located between them and the original server), and the only difference to them will be the increased browsing speed.

To enable the transparent mode, the firewall rule in destination NAT has to be added, specifying which connections (to which ports) should be transparently redirected to the proxy. Check proxy settings above and redirect us users (192.168.1.0/24) to a proxy server:

```
[admin@MikroTik] ip firewall nat> add chain=dstnat protocol=tcp src-address=192.168.1.0/24 dst-port=80
action=redirect to-ports=8080
[admin@MikroTik] ip firewall nat> print
Flags: X - disabled, I - invalid, D - dynamic
 0   chain=dstnat protocol=tcp dst-port=80 action=redirect to-ports=8080
```

The web proxy can be used as a transparent and normal web proxy at the same time. In transparent mode, it is possible to use it as a standard web proxy, too. However, in this case, proxy users may have trouble reaching web pages that are accessed transparently.

Proxy-based firewall – Access List

An access list is implemented in the same way as MikroTik firewall rules processed from the top to the bottom. The first matching rule specifies the decision of what to do with this connection. Connections can be matched by their source address, destination address, destination port, sub-string of the requested URL (Uniform Resource Locator), or request method. If none of these parameters is specified, every connection will match this rule.

If a connection is matched by a rule, the action property of this rule specifies whether a connection will be allowed or not (deny). If a connection does not match any rule, it will be allowed.

In this example assume that we have configured a transparent proxy server, it will block the website http://www.facebook.com, we can always block the same for different networks by giving src-address:

```
/ip proxy access add src-address=192.168.1.0/24 dst-host=www.facebook.com action=deny
```

Users from network 192.168.1.0/24 will not be able to access the website www.facebook.com.

You can block also websites that contain specific words in the URL:

```
/ip proxy access add dst-host=:mail action=deny
```

This statement will block all websites which contain the word “mail” in the URL. Like www.mail.com, www.hotmail.com, mail.yahoo.com, etc.

We can also stop downloading specific types of files like .flv, .avi, .mp4, .mp3, .exe, .dat, …etc.

```
/ip proxy access
add path=*.flv action=deny
add path=*.avi action=deny
add path=*.mp4 action=deny
add path=*.mp3 action=deny
add path=*.zip action=deny
add path=*.rar action=deny
```

Here are available also different wildcard characters, to create specific conditions and to match them by proxy access list. Wildcard properties (dst-host and dst-path) match a complete string (i.e., they will not match "example.com" if they are set to "example"). Available wildcards are '*' (match any number of any characters) and '?' (match any one character).

Regular expressions are also accepted here, but if the property should be treated as a regular expression, it should start with a colon (':').

To show that no symbols are allowed before the given pattern, we use the ^ symbol at the beginning of the pattern.

To specify that no symbols are allowed after the given pattern, we use the $ symbol at the end of the pattern.

Enabling RAM or Store-based caching.

In this example, it will presume that you already have the proxy configured and working and you just want to enable caching. If a command/parameter detailed description is required check the reference section which is located right below the example section.

- RAM-based caching:
  - Good if you have a device with a considerable amount of RAM for caching. Enabling this on a device with RAM 256MB or less will not give your network any benefit.
  - Way faster cache writes/read than one that is stored on USB or SATA connected mediums.
- Store-based caching:
  - Larger proxy caches are available simply due to medium capacity differences.

RAM proxy cache:

Important commands:

- max-cache-size=
- max-cache-object-size=
- cache-on-disk=

```
[admin@MikroTik] /ip proxy> set max-cache-size=unlimited max-cache-object-size=50000KiB cache-on-disk=no
...
[admin@MikroTik] /ip proxy> print
                 enabled: yes
             src-address: ::
                    port: 8080
               anonymous: no
            parent-proxy: 0.0.0.0
       parent-proxy-port: 0
     cache-administrator: webmaster
          max-cache-size: unlimited  <-------
   max-cache-object-size: 500000KiB  <-------
           cache-on-disk: no  <-------
  max-client-connections: 600
  max-server-connections: 600
          max-fresh-time: 3d
   serialize-connections: no
       always-from-cache: no
          cache-hit-dscp: 4
              cache-path: proxy-cache
```

Store proxy cache:

Important commands:

- max-cache-size=
- max-cache-object-size=
- cache-on-disk=
- cache-path=

```
[admin@MikroTik] > ip proxy set cache-on-disk=yes cache-path=/usb1/proxy/cache

[admin@MikroTik] > ip proxy print
                 enabled: yes
             src-address: ::
                    port: 8080
               anonymous: no
            parent-proxy: 0.0.0.0
       parent-proxy-port: 0
     cache-administrator: webmaster
          max-cache-size: unlimited  <-------
   max-cache-object-size: 50000KiB  <-------
           cache-on-disk: yes  <-------
  max-client-connections: 600
  max-server-connections: 600
          max-fresh-time: 3d
   serialize-connections: no
       always-from-cache: no
          cache-hit-dscp: 4
              cache-path: usb1/proxy/cache  <-------

[admin@MikroTik] > file print
 # NAME                                                              TYPE
 0 skins                                                             directory
 5 usb1/proxy                                                        directory
 6 usb1/proxy/cache                                                  web-proxy store   <-------
 7 usb1/lost+found                                                   directory
```

Check if a cache is working:

```
[admin@MikroTik] > ip proxy monitor
                 status: running
                 uptime: 2w20h28m25s
    client-connections: 15
    server-connections: 7
              requests: 79772
                  hits: 30513
            cache-used: 481KiB
         total-ram-used: 1207KiB
  received-from-servers: 4042536KiB
       sent-to-clients: 4399757KiB
   hits-sent-to-clients: 176934KiB
```

Reference

List of all available parameters and commands per menu.

General

```
/ip/proxy
```

Property	Description
always-from-cache (*yes* | *no*; Default: **no**)	ignore client refresh requests if the content is considered fresh
anonymous (*yes* | *no*; Default: **no**)	If not set, the IP address of the client would be passed X-Forwarded-For header (could be accessed using HTTP_X_FORWARDED_FOR environment variable in remote servers)
cache-administrator (*string*; Default: **webmaster**)	Administrator's e-mail displayed on proxy error page
cache-hit-dscp (*integer: 0..63*; Default: **4**)	Automatically mark cache hit with the provided DSCP value
cache-on-disk (*yes* | *no*; Default: **no**)	Whether to store cache on disk
cache-path (*string*; Default: **web-proxy**)	A path where the cache will be stored, when cache-on-disk is enabled.
max-cache-object-size (*integer: 0..4294967295 [KiB]*; Default: **2048KiB**)	Specifies the maximal cache object size, measured in kilobytes
max-cache-size (*none* | *unlimited* | *integer: 0.. 4294967295[KiB]*; Default: **unlimited**)	Specifies the maximal cache size, measured in kilobytes
max-client-connections (*integer: Dynamic* ; Default: **600**)	Maximal number of connections accepted from clients (any further connections will be rejected)
max-fresh-time (*time*; Default: **3d**)	Maximal time to store a cached object. The validity period of an object is usually defined by the object itself, but in case it is set too high, you can override the maximal value
max-server-connections (*integer: Dynamic* ; Default: **600**)	Maximal number of connections made to servers (any further connections from clients will be put on hold until some server connections will terminate)

parent-proxy (*Ip4 | ip6*; Default: **0.0.0.0**)	IP address and port of another HTTP proxy to redirect all requests to. If set to **0.0.0.0** parent proxy is not used.
parent-proxy-port (*integer: 0..65535*; Default: **0**)	Port that parent proxy is listening on.
port (*integer: 0..65535*; Default: **8080**)	TCP port the proxy server will be listening on. This port has to be specified on all clients that want to use the server as an HTTP proxy. A transparent (with zero configuration for clients) proxy setup can be made by redirecting HTTP requests to this port in the IP firewall using the destination NAT feature
serialize-connections (*yes | no*; Default: **no**)	Do not make multiple connections to the server for multiple client connections, if possible (i.e. server supports persistent HTTP connections). Clients will be served on the FIFO principle; the next client is processed when the response transfer to the previous one is completed. If a client is idle for too long (max 5 seconds by default), it will give up waiting and open another connection to the server
src-address (*Ip4 | Ip6*; Default: **0.0.0.0**)	A proxy will use a specified address when connecting to the parent proxy or website. If set to **0.0.0.0** then the appropriate IP address will be taken from the routing table.

Access List

```
/ip/proxy/access
```

An access list is configured like regular firewall rules. Rules are processed from the top to the bottom. The first matching rule specifies the decision of what to do with this connection. There is a total of 6 classifiers that specify matching constraints. If none of these classifiers is specified, the particular rule will match every connection.

If a connection is matched by a rule, the action property of this rule specifies whether a connection will be allowed or not. If the particular connection does not match any rule, it will be allowed.

Property	Description
action (*allow | deny*; Default: **allow**)	Specifies whether to pass or deny matched packets
dst-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The destination address of the target server.
dst-host (*string*; Default:)	IP address or DNS name used to make a connection to the target server (this is the string user wrote in a browser before specifying the port and path to a particular web page
dst-port (*integer[-integer[,integer[,...]]]: 0..65535*; Default:)	List or range of ports the packet is destined to
local-port (*integer: 0..65535*; Default:)	Specifies the port of the web proxy via which the packet was received. This value should match one of the ports the web proxy is listening on.
method (*any | connect | delete | get | head | options | post | put | trace*; Default:)	The HTTP method used in the request (see HTTP Methods section at the end of this document)
path (*string*; Default:)	Name of the requested page within the target server (i.e. the name of a particular web page or document without the name of the server it resides on)
redirect-to (*string*; Default:)	In case of access is denied by this rule, the user shall be redirected to the URL specified here
src-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The source address of the connection originator.

Read-only properties:

Property	Description
hits (*integer*)	Count of requests that were matched by this rule

Wildcard properties (dst-host and dst-path) match a complete string (i.e., they will not match "example.com" if they are set to "example"). Available wildcards are '*' (match any number of any characters) and '?' (match any one character). Regular expressions are also accepted here, but if the property should be treated as a regular expression, it should start with a colon (':').

Small hints in using regular expressions:

- \\ symbol sequence is used to enter \ character in the console;
- \. pattern means. only (in regular expressions single dot in a pattern means any symbol);
- to show that no symbols are allowed before the given pattern, we use the ^ symbol at the beginning of the pattern;
- to specify that no symbols are allowed after the given pattern, we use the $ symbol at the end of the pattern;
- to enter [or] symbols, you should escape them with backslash "\.";

It is strongly recommended to deny all IP addresses except those behind the router as the proxy still may be used to access your internal-use-only (intranet) web servers. Also, consult examples in Firewall Manual on how to protect your router.

Direct Access

```
/ip/proxy/direct
```

If a **parent-proxy** property is specified, it is possible to tell the proxy server whether to try to pass the request to the parent proxy or to resolve it by connecting to the requested server directly. The direct Access List is managed just like the Proxy Access List described in the previous chapter except for the action argument. Unlike the access list, the direct proxy access list has a default action equal to deny. It takes place when no rules are specified or a particular request did not match any rule.

Property	Description
action (*allow* | *deny*; Default: **allow**)	Specifies the action to perform on matched packets: • allow - always resolve matched requests directly bypassing the parent router • deny - resolve matched requests through the parent proxy. If no one is specified this has the same effect as **allow**.
dst-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The destination address of the target server.
dst-host (*string*; Default:)	IP address or DNS name used to make a connection to the target server (this is the string user wrote in a browser before specifying port and path to a particular web page
dst-port (*integer[-integer[,integer[,...]]]: 0..65535*; Default:)	List or range of ports used by connection to the target server.
local-port (*integer: 0..65535*; Default:)	Specifies the port of the web proxy via which the packet was received. This value should match one of the ports the web proxy is listening on.
method (*any | connect | delete | get | head | options | post | put | trace*; Default:)	The HTTP method used in the request (see HTTP Methods section at the end of this document)
path (*string*; Default:)	Name of the requested page within the target server (i.e. the name of a particular web page or document without the name of the server it resides on)
src-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The source address of the connection originator.

Read-only properties:

Property	Description
hits (*integer*)	Count of requests that were matched by this rule

Cache Management

```
/ip/proxy/cache
```

The cache access list specifies, which requests (domains, servers, pages) have to be cached locally by web proxy, and which do not. This list is implemented exactly the same way as the web proxy access list. The default action is to cache an object (if no matching rule is found).

Property	Description
action (*allow* | *deny*; Default: **allow**)	Specifies the action to perform on matched packets: • allow - cache objects from matched request • deny - do not cache objects from matched request
dst-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The destination address of the target server
dst-host (*string*; Default:)	IP address or DNS name used to make a connection to the target server (this is the string user wrote in a browser before specifying port and path to a particular web page
dst-port (*integer[-integer[,integer[,...]]]: 0.. 65535*; Default:)	List or range of ports the packet is destined to.
local-port (*integer: 0..65535*; Default:)	Specifies the port of the web proxy via which the packet was received. This value should match one of the ports the web proxy is listening on.
method (*any | connect | delete | get | head | options | post | put | trace*; Default:)	The HTTP method used in the request (see HTTP Methods section at the end of this document)
path (*string*; Default:)	Name of the requested page within the target server (i.e. the name of a particular web page or document without the name of the server it resides on)
src-address (*Ip4[-Ip4 | /0..32] | Ip6/0..128*; Default:)	The source address of the connection originator

Read-only properties:

Property	Description
hits (*integer*)	Count of requests that were matched by this rule

Connections

```
/ip/proxy/connections
```

This menu contains the list of current connections the proxy is serving.

Read-only properties:

Property	Description
client ()	
dst-address (*Ip4 | Ip6*)	IPv4/Ipv6 destination address of the connection
protocol (*string*)	Protocol name

rx-bytes (*integer*)	The number of bytes received by the client
server ()	
src-address (*Ip4 | Ip6*)	Ipv4/ipv6 address of the connection originator
state (*closing | connecting | converting | hotspot | idle | resolving | rx-header | tx-body | tx-eof | tx-header | waiting*)	Connection state: • closing - the data transfer is finished, and the connection is being finalized • connecting - establishing toe connection • converting - replacing header and footer fields in response or request packet • hotspot - check if hotspot authentication allows continuing (for hotspot proxy) • idle - staying idle • resolving - resolving the server's DNS name • rx-header - receiving HTTP header • tx-body - transmitting HTTP body to the client • tx-eof - writing chunk-end (when converting to chunked response) • tx-header - transmitting HTTP header to the client • waiting - waiting for transmission from a peer
tx-bytes (*integer*)	The number of bytes sent by the client

Cache Inserts

```
/ip/proxy/inserts
```

This menu shows statistics on objects stored in a cache (cache inserts).

Read-only properties:

Property	Description
denied (*integer*)	A number of inserts were denied by the caching list.
errors (*integer*)	Number of disk or other system-related errors
no-memory (*integer*)	Number of objects not stored because there was not enough memory
successes (*integer*)	A number of successful cache inserts.
too-large (*integer*)	Number of objects too large to store

Cache Lookups

```
/ip/proxy/lookup
```

This menu shows statistics on objects read from cache (cache lookups).

Read-only properties:

Property	Description
denied (*integer*)	Number of requests denied by the access list.
expired (*integer*)	Number of requests found in cache, but expired, and, thus, requested from an external server

no-expiration-info (*integer*)	Conditional request received for a page that does not have the information to compare the request with
non-cacheable (*integer*)	Number of requests requested from the external servers unconditionally (as their caching is denied by the cache access list)
not-found (*integer*)	Number of requests not found in the cache, and, thus, requested from an external server (or parent proxy if configured accordingly)
successes (*integer*)	Number of requests found in the cache.

Cache Contents

```
/ip/proxy/cache-contents
```

This menu shows cached contents.

Read-only properties:

Property	Description
file-size (*integer*)	Cached object size
last-accessed (*time*)	
last-accessed-time (*time*)	
last-modified (*time*)	
last-modified-time (*time*)	
uri (*string*)	

HTTP Methods

Options

This method is a request for information about the communication options available on the chain between the client and the server identified by the **Request-URI**. The method allows the client to determine the options and (or) the requirements associated with a resource without initiating any resource retrieval

GET

This method retrieves whatever information identified by the Request-URI. If the Request-URI refers to a data processing process then the response to the GET method should contain data produced by the process, not the source code of the process procedure(-s), unless the source is the result of the process.

The GET method can become a conditional GET if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. The conditional GET method is used to reduce the network traffic specifying that the transfer of the entity should occur only under circumstances described by conditional header field(-s).

The GET method can become a partial GET if the request message includes a Range header field. The partial GET method intends to reduce unnecessary network usage by requesting only parts of entities without transferring data already held by the client.

The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching.

HEAD

This method shares all features of GET method except that the server must not return a message-body in the response. This retrieves the metainformation of the entity implied by the request which leads to its wide usage of it for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request may be cacheable in the way that the information contained in the response may be used to update the previously cached entity identified by that Request-URI.

POST

This method requests that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI.

The actual action performed by the POST method is determined by the origin server and usually is Request-URI dependent.

Responses to POST method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields.

PUT

This method requests that the enclosed entity be stored under the supplied Request-URI. If another entity exists under specified Request-URI, the enclosed entity should be considered as an updated (newer) version of that residing on the origin server. If the Request-URI is not pointing to an existing resource, the origin server should create a resource with that URI.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this method are not cacheable.

TRACE

This method invokes a remote, application-layer loop-back of the request message. The final recipient of the request should reflect the message received back to the client as the entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of 0 in the request. A TRACE request must not include an entity.

Responses to this method MUST NOT be cached.

Routing

In This Section:

Routing Protocol Overview

Feature Status

N/A - Feature not yet available

OK - Initial tests successful

NOK - initial tests not successful

Highlight Colors:

- Yellow - partially working
- Green - Working
- Red - Not working at the moment

Feature	v7.1	v7.2	v7.3	v7.6	v7.7	v7.10	v7.12	v7.14
Winbox								
BGP support								
OSPF support								
RIP support								
Router ID support								
Routing filter support								
Generic								
/31 address support	N/A		Routed traffic does not work to odd address.					
Convert route rules after upgrade from v6.x								
Static IPv6 upgrade from ROS v6								
IPv4 Route Rules								
IPv6 Route Rules								
ECMP flags								
dst@table								
gateway@table								
gateway%interface								
recursive route over ipv6 LL address								
3 level recursive gateway with ECMP								
IPV6 ECMP								
IPv6 connected ECMP								
Addresses from same subnet to multiple interfaces	N/A							
Show time when route was last updated	N/A							
Check Gateway	BFD not ready							
Scope and target scope								
IPv4 Mangle routing-mark								

IPv6 Mangle routing-mark								
Packet SRC address	Does not work correctly with /32 addresses							
Routing-table parameter for ping and telnet								
Show if route is hardware accelerated	Shows if route is candidate for HW acceleration							
Custom route selection policy								
IPv4 with IPv6 nexthops for RFC5549								
Routing id								
VRF								
Management services support for VRFs	telnet, ssh, api, www services can be set to listen on specific VRF							
Some kind of mechanism to import/export routes from one vrf to another within same router	N/A							
BFD	N/A					Initial support		
OSPF								
Convert OSPF config from v6 to v7 after upgrade	Known conversion problems: • NBMA neighbors place in backbone • ospf-v2 networks + interface may have issues • dynamic interfaces may have issues • MPLS PE CE features are not converted							
OSPF neighbors in NSSA Area								
OSPF in broadcast network								
OSPF with routing filters								
OSPF Virtual Link								
OPSF input filtering								
HMAC-SHA auth RFC5709	N/A			Initial support				
OSPF SNMP monitoring	N/A							
BGP SNMP monitoring						For ipv4 sessions		
IS-IS								
IPv4							Initial support	
IPv6								
Traffic Engineering								
BGP								
Convert BGP config from v6 to v7 after upgrade								
BGP Templates and dynamic peers								
BGP connect listen on a network								
BGP guess remote.as								
Show from which peer route received	OK (/routing/route/print detail --> belongs-to)							
BGP Address Families								
BGP input.accept-*								

eBGP nexthop self								
Input Filter								
Output Filter								
BGP Local address auto selection								
BGP route reflect								
BGP route server								
BGP Roles https://datatracker.ietf.org/doc/draft-ietf-idr-bgp-open-policy/?include_text=1	rfc roles not fully implemented							
BGP session uptime in "established" state								
BGP session last established time								
BGP Flow Spec	Flow spec attributes are forwarded							
BGP Selection								
BGP Selection (Multipath)	N/A							
BGP Confederation								
BGP Aggregation	N/A							
BGP ORF	N/A							
Discard prefix RTBH RFC 6666	N/A							
AS-wide Unique BGP Identifier RFC 6286	N/A							
Exported PDU PCAP saver								
Exported PDU PCAP loader								
BGP Advertisement monitoring		Initial implementation by dumping to pcap		Advertisements rework				
BGP Prefix limit			Initial support					
BGP advertise IPv4 prefix with IPv6 nexthop (RFC5549)								
BGP VPNv6 support						Prerequisites are made, need to add actual BGP Afi		
MPLS								
Static label mapping								
Static mapping upgrade from v6								
LDP IPv4 mapping								
LDP IPv6 mapping								
LDP signaled VPLS								
LDP config upgrade from v6								
LDP Dual Stack								
TE								
TE Config upgrade from v6								
VPLS Encap to TE								
BGP signaled VPLS								
VPLS config upgrade from v6								
Fast reroute								
MPLS ECMP								
One label per VRF								
Ability to use MPLS EXP-bit in Queues	N/A							
MPLS Fast-Path	N/A							

RPKI session								
RPKI possibility to view received info of specific prefix								
RPKI show connection status								
Filters								
Convert routing filters after upgrade from v6.x								
Syntax completion								
Routing filter chain drop by default without rules								
Routing filter prefix match								
Routing filter protocol match								
Routing filter append communities								
Routing filter append large community								
Routing filter set weight								
Routing filter set local pref								
Routing filter set MED								
Routing filter set origin								
Routing filter set igp metric from OSPF cost								
Routing filter match prefix with address list								
Routing filter match community/large community lists								
Routing filter add a prefix to address list	N/A							
Routing filter validate prefix with RPKI								
Multicast								
IGMP-Proxy								
PIM-SM	Initial support							

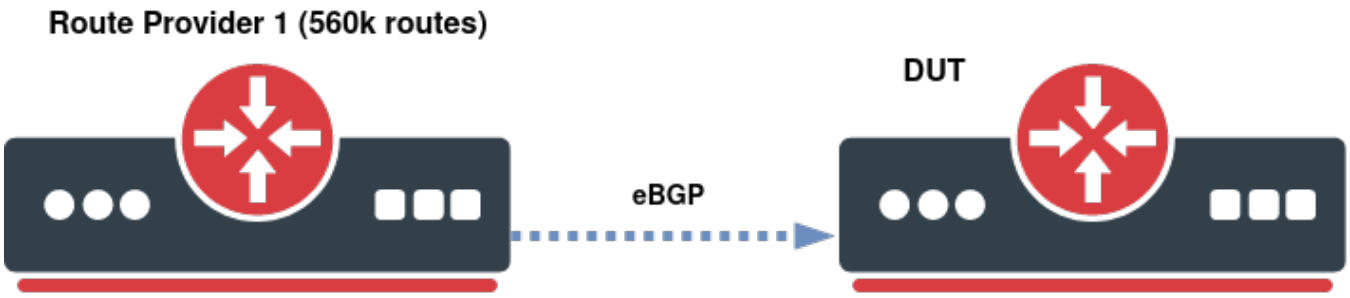
Performance Status

Used hardware:

- CCR1036, 16GB RAM (tile)
- CCR2004(arm64)
- CCR1100AHx4(arm)
- Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 32GB RAM (as a host for CHRs)

The simulated upstream peer is a CHR router running ROSv6 with a copy of the global IPv4 routing table (585K routes loaded from MRT dump).

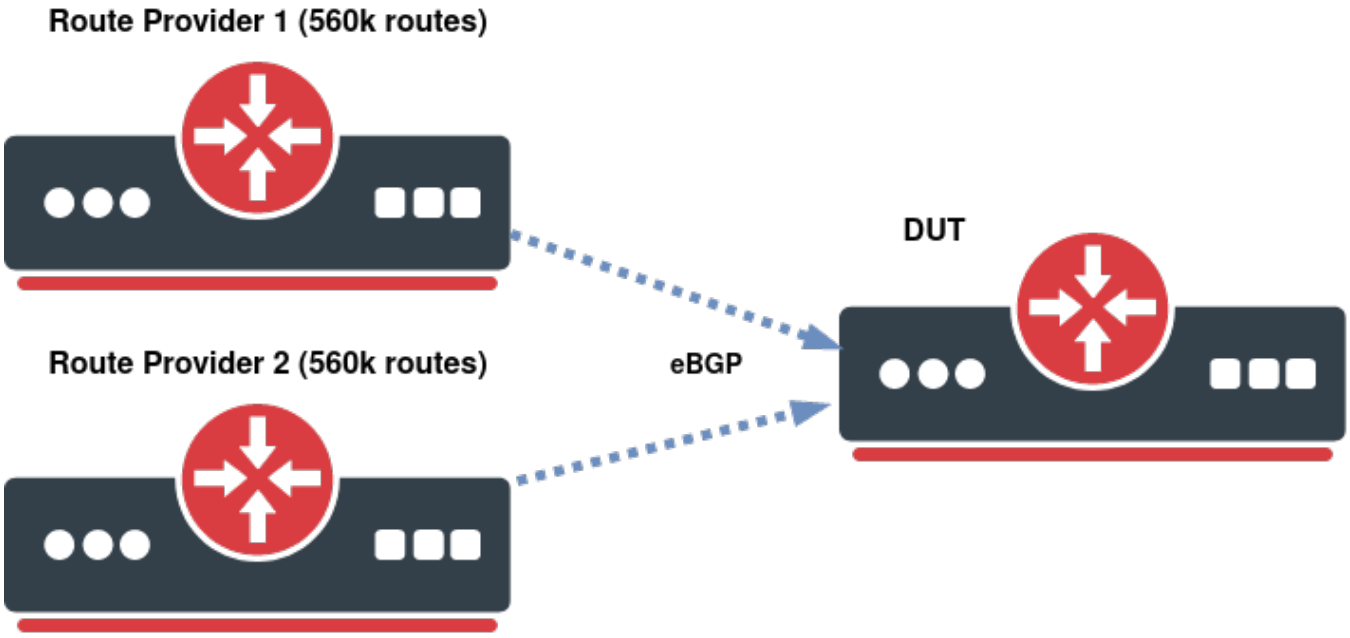
One Peer Receive Only



DUT establishes a connection to simulated upstream peers, receives routes, and installs them in FIB.

	v6.44	v7.1beta3	v7.1rc7
CCR	0:40 - 2:12	0:46	
RB1100x4 1.4GHz	0:32-0:38	0:23	
CCR2004	0:32	0:18	
x86 (CHR)	0:20		
RB450G (in/out affinity=alone)	after trying for 9min - ran out of memory at 558K routes	2:02 (121MB free)	
RB450G (in/out affinity=main)	-	1:54	
RB450G (affinity in=alone out=input)	-	2:12	

Two Peers Receive Only

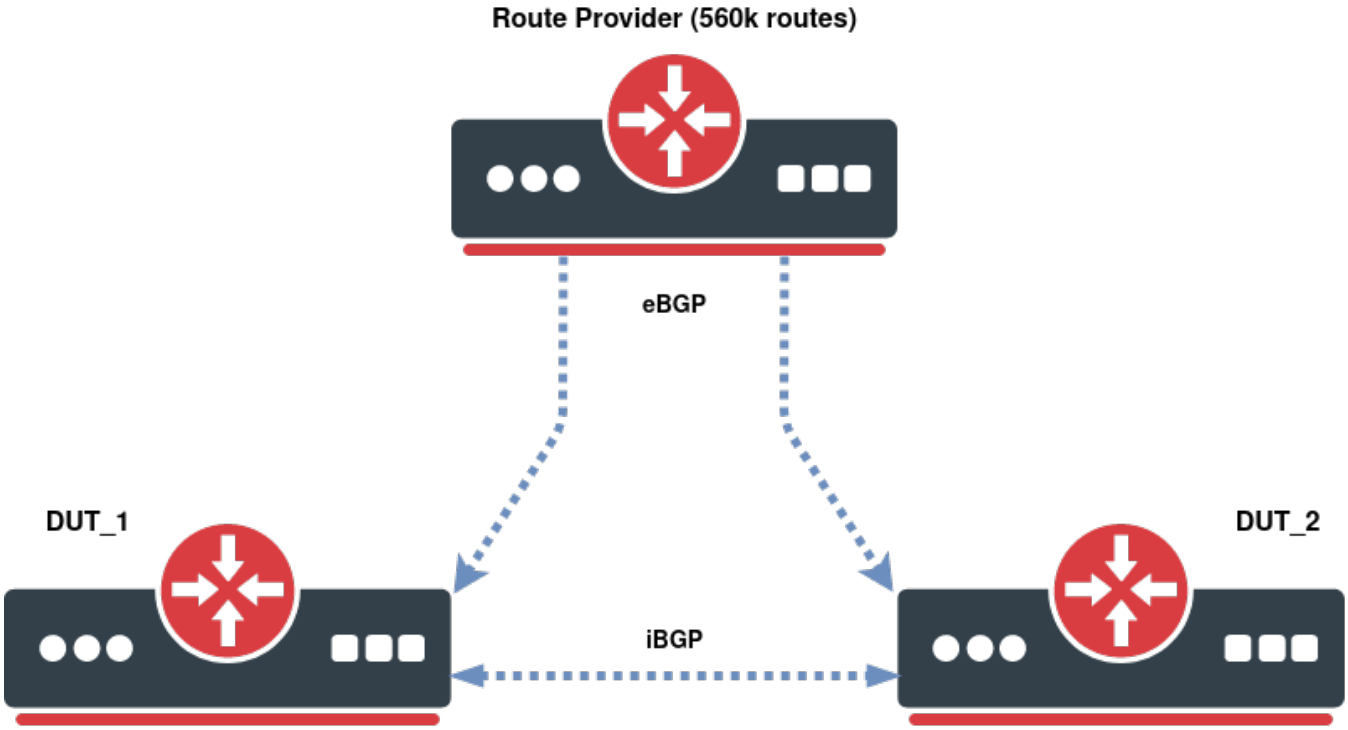


DUT establishes a connection to two simulated upstream peers, receives routes, picks the best route, and installs in FIB. On ROSv7 affinity settings are set to "alone".

	v6.44	FRR	v7.1beta3	v7.1rc7 (846k routes per peer)
CCR	1:01 - 2:45		1:11	
RB1100x4 1.4GHz	0:51		0:30	
CCR2004	0:51		0:29	0:33
router x				0:40
x86 (CHR)	0:25			
x86 (virtual)		0:26(4cores)		
		0:46(2cores)		
		0:30(2cores no LDP)		

Multi-homing Sim

Two DUT devices establish eBGP sessions to simulated x86 upstream routers. Both DUTs are interconnected with the iBGP session. Each DUT receives routes from upstream and readvertises routes over iBGP. On ROSv7 affinity, settings are set to "alone" and early-cut disabled.



- Route Provider: CHR (ROSv6)
- DUT_1: CCR1036
- DUT_2: CCR1036

v7.1beta3	1:11
v7.1beta2	1:29
v6.xx	1:02 - 8:30

- Route Provider: CHR (ROSv6)
- DUT_1: CCR2004
- DUT_2: RB1100AHx2

v7.1beta3	0:36
v6.xx	0:59

Memory Usage:

```
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME,
KERNEL-TIME, CUR-BUSY, MAX-BU>
   #  TASKS                          PRIVATE-M  SHARED-M  P  R  V  RE  ID       PID  R  PROCESS-  KERNEL-  CUR
MAX-BUS  CUR  MAX-CALC
   0  routing tables                 12.0MiB    30.2MiB   0  0  0  12  main     111  0  8s980ms   2s60ms   0ms
1s320ms  0ms  10s700ms

rib

      connected
networks

   1  fib                            2816.0KiB  0         0  0  0      fib      130  1  3s        4s660ms
7s220ms       7s220ms
   2  ospf                           512.0KiB   256.0KiB  0  0  0      ospf     137  1  1s220ms   130ms
980ms         1s40ms
      connected
networks

   3  fantasy                        256.0KiB   0         0  0  0      fantasy  138  1  60ms      80ms
40ms          40ms
   4  configuration and reporting    3840.0KiB  512.0KiB  0  0  0      static   139  1  1s270ms   110ms
260ms         260ms
   5  rip                            512.0KiB   0         0  0  0      rip      136  1  120ms     70ms
60ms          120ms
      connected
networks

   6  routing policy configuration   768.0KiB   768.0KiB  0  0  0      policy   133  1  2s290ms   3s170ms
80ms          80ms
   7  BGP service                    768.0KiB   0         0  0  0      bgp      134  1  2s760ms   5s480ms
20ms          60ms
      connected
networks

   8  BFD service                    512.0KiB   0         0  0  0      12       135  1  100ms     90ms
40ms          120ms
      connected
networks

   9  BGP Input 10.155.101.186       3072.0KiB  6.2MiB    0  0  0      20       183  1  1s350ms   1s190ms
20ms          20ms
  10  BGP Output 10.155.101.186      5.5MiB     0         0  0  0      21       184  1  5s400ms   500ms
3s880ms       3s880ms
  11  BGP Input 10.155.101.232       3072.0KiB  6.2MiB    0  0  0      22       187  1  970ms     740ms
20ms          20ms
  12  BGP Output 10.155.101.232      8.2MiB     0         0  0  0      23       188  1  10s830ms  960ms
7s            7s
  13  Global memory                             256.0KiB              global     0  0
```

Moving from ROSv6 to v7 with examples

Routing Tables

By default, all routes are added to the "main" routing table as it was before. From a configuration point of view, the biggest differences are routing table limit increase, routing table monitoring differences, and how routes are added to specific routing tables (see next example)
v7 introduces a new menu /routing route, which shows all address family routes as well as all filtered routes with all possible route attributes. `/ip route` a nd `/ipv6 route` menus are used to add static routes and for simplicity show only basic route attributes.

For more in-depth information on routing see this article (IP Routing).

Another new change is that most common route print requests are processed by the routing process which significantly improves the speed compared to v6.

Use of Routing Tables and Policy Routing

The main difference from v6 is that the routing table must be added to the `/routing table` menu before actually referencing it anywhere in the configuration. And **fib** parameter should be specified if the routing table is intended to push routes to the FIB.
The routing rule configuration is the same except for the menu location (instead of `/ip route rule`, now it is `/routing rule`).

Let's consider a basic example where we want to resolve 8.8.8.8 only in the routing table named myTable to the gateway 172.16.1.1:

```
/routing table add name=myTable fib
/routing rule add dst-address=8.8.8.8 action=lookup-only-in-table table=myTable
/ip route add dst-address=8.8.8.8 gateway=172.16.1.1@main routing-table=myTable
```

Instead of routing rules, you could use mangle to mark packets with routing-mark, the same way as it was in ROSv6.

OSPF Configuration

OSPFv3 and OSPFv2 are now merged into one single menu `/routing ospf`. At the time of writing this article, there are no default instances and areas. To start both OSPFv2 and OSPF v3 instances, first, you need to create an instance for each and then add an area to the instance.

```
/routing ospf instance
add name=v2inst version=2 router-id=1.2.3.4
add name=v3inst version=3 router-id=1.2.3.4
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=backbone_v3 area-id=0.0.0.0 instance=v3inst
```

At this point, you are ready to start OSPF on the network interface. In the case of IPv6, you add either interface on which you want to run OSPF (the same as ROSv6) or the IPv6 network. In the second case, OSPF will automatically detect the interface. Here are some interface configuration examples:

```
/routing ospf interface-template
add network=192.168.0.0/24 area=backbone_v2
add network=2001:db8::/64 area=backbone_v3
add network=ether1 area=backbone_v3
```

ROSv7 uses templates to match the interface against the template and apply configuration from the matched template. OSPF menus `interface` and `neighbor` contains read-only entries purely for status monitoring.

~~All route distribution control is now done purely with routing filter select, no more redistribution knobs in the instance~~ (Since the v7.1beta7 redistribution knob is back, you still need to use routing filters to set route costs and type if necessary). This gives greater flexibility on what routes from which protocols you want to redistribute.
For example, let's say you want to redistribute only static IPv4 routes from the 192.168.0.0/16 network range.

```
/routing ospf instance
set backbone_v2 out-filter-chain=ospf_out redistribute=static
```

```
/routing filter rule add chain=ospf_out rule="if (dst in 192.168.0.0/16) {accept}"
```

⚠ The default action of the routing filter chain is "drop"

BGP Configuration

There is a complete redesign of the BGP configuration compared to ROSv6. The first biggest difference is that there is no more **`instance`** and **`peer`** configuration menus. Instead, we have **`connection`**, **`template`** and **`session`** menus.
The reason for such a structure is to strictly split parameters that are responsible for connection and parameters that are BGP protocol specific.

Let's start with the Template. It contains all BGP protocol-related configuration options. It can be used as a template for dynamic peers and apply a similar config to a group of peers. Note that this is not the same as peer groups on Cisco devices, where the group is more than just a common configuration.

By default, there is a default template that requires you to set your own AS.

```
/routing/bgp/template set default as=65533
```

⚠ Starting from v7.1beta4 template parameters are exposed in the "connection" configuration. This means that the template is not mandatory anymore, allowing for an easier basic BGP connection setup, similar to what it was in ROSv6.

Most of the parameters are similar to ROSv6 except that some are grouped in the output and input section making the config more readable and easier to understand whether the option is applied on input or output. If you are familiar with CapsMan then the syntax is the same, for example, to specify the output selection chain you set `output.filter-chain=myBgpChain`.

You can even inherit template parameters from another template, for example:

```
/routing/bgp/template
add name=myAsTemplate as=65500 output.filter-chain=myAsFilter
set default template=myAsTemplate
```

Another important aspect of the new routing configuration is the global Router ID, which sets router-id and group peers in one instance. RouterOS adds a default ID which picks instance-id from any interface's highest IP. The default BGP template by default is set to use the "default" ID.
If for any reason you need to tweak or add new instances it can be done in `/routing id` menu.

Very interesting parameters are **`input.affinity`** and **`output.affinity`**, they allow control in which process input and output of active session will be processed:

- **alone** - input and output of each session are processed in its own process, most likely the best option when there are a lot of cores and a lot of peers
- **afi, instance, vrf, remote-as** - try to run input/output of new session in process with similar parameters
- **main** - run input/output in the main process (could potentially increase performance on single-core even possibly on multicore devices with small amount of cores)
- **input** - run output in the same process as input (can be set only for output affinity)

Now that we have parameters set for the template we can add BGP connections. A minimal set of parameters are `remote.address`, `template, connect`, `listen` and `local.role`

Connect and listen to parameters specify whether peers will try to connect and listen to a remote address or just connect or just listen. It is possible that in setups where peer uses the multi-hop connection `local.address` must be configured too (similar as it was with `update-source` in ROSv6).

> ⚠ It is not mandatory to specify a remote AS number. ROS v7 can determine remote ASN from an open message. You should specify the remote AS only when you want to accept a connection from that specific AS.

Peer role is now a mandatory parameter, for basic setups, you can just use ibgp, ebgp (more information on available roles can be found in the corresponding RFC draft https://datatracker.ietf.org/doc/draft-ietf-idr-bgp-open-policy/?include_text=1), keep in mind that at the moment capabilities, communities, and filtering described in the draft is not implemented.

Very basic iBGP set up to listen on the whole local network for connections:

```
/routing/bgp/connection
add remote.address=10.155.101.0/24 listen=yes template=default local.role=ibgp
```

Now you can monitor the status of all connected and disconnected peers from `/routing bgp session` menu.

Other great debugging information on all routing processes can be monitored from `/routing stats` menu

```
[admin@v7_ccr_bgp] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME,
KERNEL-TIME, CUR-B>
# TASKS PRIVATE-M SHARED-ME PSS RSS VMS RET ID PID R PROCESS-TI KERN>
0 routing tables 12.2MiB 20.0MiB 18.7MiB 42.2MiB 83.4MiB 8 main 319 0 19s750ms 8s50>
rib >
connected networks >
1 fib 512.0KiB 0 7.4MiB 30.9MiB 83.4MiB fib 384 1 5s160ms 22s5>
2 ospf 1024.0KiB 1024.0KiB 5.9MiB 25.9MiB 83.4MiB 382 ospf 388 1 1m42s170ms 1m31>
connected networks >
3 fantasy 512.0KiB 0 2061.0KiB 5.9MiB 83.4MiB fantasy 389 1 1s410ms 870m>
4 configuration and reporting 40.0MiB 512.0KiB 45.0MiB 64.8MiB 83.4MiB static 390 1 12s550ms 1s17>
5 rip 768.0KiB 0 5.3MiB 24.7MiB 83.4MiB rip 387 1 1s380ms 1s20>
connected networks >
6 routing policy configuration 512.0KiB 256.0KiB 2189.0KiB 6.0MiB 83.4MiB policy 385 1 1s540ms 1s20>
7 BGP service 768.0KiB 0 2445.0KiB 6.2MiB 83.4MiB bgp 386 1 6s170ms 9s38>
8 BGP Input 10.155.101.217 8.8MiB 6.0MiB 15.6MiB 38.5MiB 83.4MiB 20 21338 1 25s170ms 3s23>
BGP Output 10.155.101.217 >
9 Global memory 256.0KiB global 0 0 >
-- [Q quit|D dump|C-z pause|right]
```

Route filtering differs a bit from ROSv6. In the BGP template, you can now specify output.filter-chain, output.filter-select, input.filter as well as several input.accept-* options.

Now input.accept-* allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered",

A very basic example of a BGP input filter to accept prefixes from 192.168.0.0/16 subnet without modifying any attributes. For other prefixes subtract 1 from the received local pref value and set IGP metric to value from OSPF ext. Additionally, we will accept only specific prefixes from the address list to reduce memory usage

```
/ip/firewall/address-list
add list=bgp_list dst-address=192.168.1.0/24
add list=bgp_list dst-address=192.168.0.0/24
add list=bgp_list dst-address=172.16.0.0/24
```

```
/routing/bgp/template
set default input.filter=bgp_in .accept-nlri=bgp_list
```

```
/routing/filter/rule
add chain=bgp_in rule="if (dst in 192.168.0.0/16) {accept}"
add chain=bgp_in rule="set bgp-local-pref -1; set bgp-igp-metric ospf-ext-metric; accept"
```

If the routing filter chain is not specified BGP will try to advertise every active route it can find in the routing table

The default action of the routing filter chain is "drop"

Monitoring Advertisements

RouterOS v7 by default disables monitoring of the BGP output. This allows to significantly reduce resource usage on setups with large routing tables.

To be able to see output advertisements several steps should be taken:

- enable "output.keep-sent-attributes" in BGP connection configuration
- run "dump-saved-advertisements" from BGP session menu
- view saved output from "/routing/stats/pcap" menu

```
[admin@arm-bgp] /routing/bgp/connection>  set 0 output.keep-sent-attributes=yes
[admin@arm-bgp] /routing/bgp/session> print
Flags: E - established
 0 E remote.address=10.155.101.183 .as=444 .id=192.168.44.2 .refused-cap-opt=no .capabilities=mp,rr,gr,as4
     .afi=ip,ipv6 .messages=4 .bytes=219 .eor=""
     local.address=10.155.101.186 .as=456 .id=10.155.255.186 .capabilities=mp,rr,gr,as4 .afi=ip,ipv6
     .messages=1 .bytes=19 .eor=""
     output.procid=66 .filter-chain=bgp_out .network=bgp-nets .keep-sent-attributes=yes
     input.procid=66 ebgp
     hold-time=3m keepalive-time=1m uptime=4s30ms

[admin@arm-bgp] /routing/bgp/session> dump-saved-advertisements 0 save-to=test_out.pcap
```

Networks

Lastly, you might notice that the `network` menu is missing and probably wondering how to advertise your own networks. Now networks are added to the firewall address-list and referenced in the BGP configuration.
Following ROSv6 network configuration:

```
/routing bgp network add network=192.168.0.0/24 synchronize=yes
/ip route add dst-address=192.168.0.0/24 type=blackhole
```

would translate to v7 as:

```
/ip/firewall/address-list/
add list=bgp-networks address=192.168.0.0/24

/ip/route
add dst-address=192.168.0.0/24 blackhole

/routing/bgp/connection
set peer_name output.network=bgp-networks
```

There is more configuration to be done when adding just one network but offers simplicity when you have to deal with a large number of networks. v7 even allows specifying for each BGP connection its own set of networks.

> ⚠ In v7 it is not possible to turn off synchronization with IGP routes (the network will be advertised only if the corresponding IGP route is present in the routing table).

Routing Filters

Starting from ROSv7.1beta4, the routing filter configuration is changed to a script-like configuration. The rule now can have "if .. then" syntax to set parameters or apply actions based on conditions from the "if" statement.

Multiple rules without action are stacked in a single rule and executed in order like a firewall, the reason is that the "set" parameter order is important and writing one "set"s per line, allows for an easier understanding from top to bottom on what actions were applied.

For example, match static default route and apply action accept can be written in one config rule:

```
/routing/filter/rule
add chain=ospf_in rule="if (dst==0.0.0.0/0 && protocol static) { accept }"
```

For example, ROSv6 rule "/routing filter add chain=ospf_in prefix=172.16.0.0/16 prefix-length=24 protocol=static action=accept" converted to ROSv7 would be:

```
/routing/filter/rule
add chain=ospf_in rule="if (dst in 172.16.0.0/16 && dst-len==24 && protocol static) { accept }"
```

Another example, to match prefixes from the 172.16.0.0/16 range with prefix length equal to 24 and set BGP med and prepend values

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len==24 && dst in 172.16.0.0/16) { \n
    set bgp-med 20; set bgp-path-prepend 2; accept }"
```

It is also possible to match prefix length range like this

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len>13 && dst-len<31 && dst in 172.16.0.0/16) { accept }"
```

Filter rules now can be used to match or set communities, large communities, and extended communities from the community list:

```
/routing/filter/rule
add chain=bgp_in rule="set bgp-large-communities 200001:200001:10 "
```

If there are a lot of community sets, that need to be applied in multiple rules, then it is possible to define community sets and use them to match or set:

```
/routing/filter/large-community-set
add set=myLargeComSet communities=200001:200001:10


/routing/filter/rule
add chain=bgp_in rule="append bgp-large-communities myLargeComSet "
```

Since route-target is encoded in extended community attribute to change or match RT you need to operate on extended community attribute, for example:

```
/routing/filter/rule
add chain=bgp_in rule="set bgp-ext-communities rt:327824:20 "
```

RPKI

RouterOS implements an RTR client. You connect to the server which will send route validity information. This information then can be used to validate routes in route filters against a group with "rpki-validate" and further in filters "match-rpki" can be used to match the exact state.

For more info refer to the RPKI documentation.

RIP Configuration

To start RIP, the instance should be configured. There you should select which routes will be redistributed by RIP and if it will redistribute the default route.

```
/routing/rip/instance
add name=instance1 originate-default=never redistribute=connected,static
```

Then interface-template should be configured. There is no need to define networks in ROS version 7 as it was in version 6.

```
/routing/rip/interface-template
add interfaces=ether1 instance=instance1
```

Now the basic configuration is completed on one router. RIP neighbor router should be configured in a similar way.

In ROS v7 the neighbors will appear only when there are routes to be sent or/and to be received.

Prefix lists from ROSv6 are deprecated, now all the filtering must be done by the routing filters.

Routing Protocol Multi-core Support

Overview

RouterOS v7 is capable of splitting tasks between multiple processes.
There is one "**main**" task, which can start/stop sub-tasks and process data between those sub-tasks. Each sub-task can allocate "private" (only accessible by this particular task) and "shared" memory (accessible by all route tasks).

List of tasks that can be split:

- Handling of "print" command;
- Entire OSPF protocol handling;
- Entire RIP protocol handling;
- Static configuration handling;
- Routing Policy configuration;
- BGP connections and configuration handling;
- BGP receive (one task per peer or grouped by specific parameters);
- BGP send (one task per peer or grouped by specific parameters);
- FIB update.

BGP Sub-Tasks

BGP receive and send can be split into sub-tasks by specific parameters, for example, it is possible to run input per each peer or group all peer inputs and run them in the main process. This split by sub-tasks is controlled with **`input.affinity`** and **`output.affinity`** parameter configuration in **`/routing/bgp/template`**.It is possible to boost performance by playing with affinity values on devices with fewer cores since sharing data between tasks is a bit slower than processing the same data within one task. For example, on single-core or two-core devices running input and output in the main or instance process will boost performance.

> ⚠ BGP can have up to 100 unique processes.

All currently used tasks and their allocated private/shared memory can be monitored using the command:

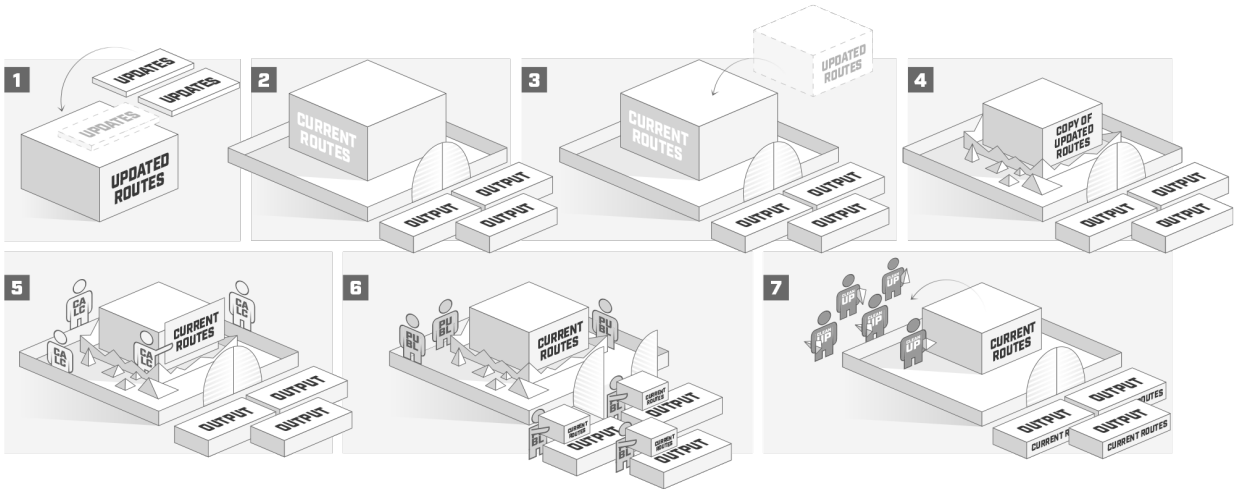
```
/routing/stats/process/print
```

Sample Output:

```
[admin@BGP_MUM] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME,
KERNEL-TIME, CUR-BUSY, MAX-BUSY, CUR-CALC, MAX-CALC
   #  TASKS                          PRIVATE-M  SHARED-M  PSS        RSS      VMS      R  ID       PID  R
PROCESS-  KERNEL-TI  CUR-  MAX-BUSY  CUR-  MAX-CALC
   0  routing tables                 11.8MiB    20.0MiB   19.8MiB    42.2MiB  51.4MiB  7  main     195  0
15s470ms  2s50ms     20ms  1s460ms   20ms  35s120ms

rib

      connected
networks

   1  fib                            2816.0KiB  0         8.1MiB     27.4MiB  51.4MiB     fib      255  1
5s730ms   7m4s790ms        23s350ms        23s350ms
   2  ospf                           512.0KiB   0         3151.0KiB  14.6MiB  51.4MiB     ospf     260  1
20ms      100ms            20ms            20ms
      connected
networks

   3  fantasy                        256.0KiB   0         1898.0KiB  5.8MiB   51.4MiB     fantasy  261  1
40ms      60ms             20ms            20ms
   4  configuration and reporting    4096.0KiB  512.0KiB  9.2MiB     28.4MiB  51.4MiB     static   262  1
3s210ms   40ms             220ms           220ms
   5  rip                            512.0KiB   0         3151.0KiB  14.6MiB  51.4MiB     rip      259  1
50ms      90ms             20ms            20ms
      connected
networks

   6  routing policy configuration   768.0KiB   768.0KiB  2250.0KiB  6.2MiB   51.4MiB     policy   256  1
70ms      50ms             20ms            20ms
   7  BGP service                    768.0KiB   0         3359.0KiB  14.9MiB  51.4MiB     bgp      257  1
4s260ms   8s50ms           30ms            30ms
      connected
networks

   8  BFD service                    512.0KiB   0         3151.0KiB  14.6MiB  51.4MiB     12       258  1
80ms      40ms             20ms            20ms
      connected
networks

   9  BGP Input 10.155.101.232       8.2MiB     6.8MiB    17.0MiB    39.1MiB  51.4MiB     20       270  1
24s880ms  3s60ms           18s550ms        18s550ms
      BGP Output
10.155.101.232

  10  Global memory                      256.0KiB                                        global     0  0
```

Routing Table Update Mechanism

Illustration below tries to explain in more user friendly form on how routing table update mechanism is working.



Routing protocols continuously loop through following procedures:

- "**main**" process waits for updates from other sub tasks (1);
- "**main**" starts to calculate new routes (2..4) if:
    - update from sub task is received;
    - protocol has not published all routes;
    - configuration has changed or link state has changed.
- during new route calculation (5) following event occur:
    - all received updates are applied to the route;
    - gateway reachability is being determined;
    - recursive route is being resolved;
- "**publish**" event is called where "**current**" routes are being published. During this phase, "**current**" routes will not change, but protocols can still receive and send updates (6).
- Do cleanup and free unused memory (7). In this step everything that is no longer used in new "**current**" table is removed (routes, attributes, etc.).

Consider "**updated**" and "**current**" as two copies of routing table, where "**current**" table (2) is the one used at the moment and "**updated**" (1) is table of candidate routes to be published in the next publish event (3 and 4). This method prevents protocols to fill memory with buffered updates while "**main**" process is doing "**publish**", instead protocols sends the newest update directly to "main" process which then copies new update in "**updated**" table. A bit more complicated is OSPF, it internally has similar process to select current OSPF routes which then are sent to the "**main**" for further processing.

Policy Routing

Overview

Policy routing is the method to steer traffic matching certain criteria to a certain gateway. This can be used to force some customers or specific protocols from the servers (for example HTTP traffic) to always be routed to a certain gateway. It can even be used to steer local and overseas traffic to different gateways.

RouterOS implements several components that can be used to achieve said task:

- routing tables
- routing rules
- firewall mangle marking

Routing Tables

A router can have multiple routing tables with its own set of routes routing the same destination to different gateways.

Tables can be seen and configured from the `/routing/table` menu.

By default, RouterOS has only the '**main**' routing table:

```
[admin@rack1_b33_CCR1036] /routing/table> print
Flags: D - dynamic; X - disabled, I - invalid; U - used
0 D name="main" fib
```

If a custom routing table is required, it should be defined in this menu prior to using it anywhere in the configuration.

Let's consider a basic example where we have two gateways 172.16.1.1 and 172.16.2.1 and we want to resolve 8.8.8.8 only in the routing table named '**my Table**' to the gateway 172.16.2.1:

```
/routing table add name=myTable fib
/ip route add dst-address=8.8.8.8 gateway=172.16.1.1
/ip route add dst-address=8.8.8.8 gateway=172.16.2.1@main routing-table=myTable
```


For a user-created table to be able to resolve the destination, the main routing table should be able to resolve the destination too.

In our example, the **main** routing table should also have a route to destination 8.8.8.8 or at least a default route, since the default route is dynamically added by the DHCP for safety reasons it is better to add 8.8.8.8 also in the main table.

```
[admin@rack1_b33_CCR1036] /ip/route> print detail Flags: D - dynamic; X - disabled, I - inactive, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, y - cop
y;
H - hw-offloaded; + - ecmp
   DAd   dst-address=0.0.0.0/0 routing-table=main pref-src="" gateway=172.16.1.1
         immediate-gw=172.16.1.1%ether8 distance=1 scope=30 target-scope=10
         vrf-interface=ether8 suppress-hw-offload=no

 0  As   dst-address=8.8.8.8/32 routing-table=main pref-src="" gateway=172.16.1.1
         immediate-gw=172.16.1.1%ether8 distance=1 scope=30 target-scope=10 suppress-hw-offload=no

    DAc   dst-address=172.16.1.0/24 routing-table=main gateway=ether8 immediate-gw=ether8
         distance=0 scope=10 suppress-hw-offload=no local-address=172.16.1.2%ether8

    DAc   dst-address=172.16.2.0/24 routing-table=main gateway=ether7 immediate-gw=ether7
         distance=0 scope=10 suppress-hw-offload=no local-address=172.16.2.2%ether7

 1  As   dst-address=8.8.8.8/32 routing-table=myTable pref-src="" gateway=172.16.2.1
         immediate-gw=172.16.2.1%ether7 distance=1 scope=30 target-scope=10 suppress-hw-offload=no
```

But configuration above is not enough, we need a method to force the traffic to actually use our newly created table. RouterOS gives you two options to choose from:

- firewall mangle - it gives more control over the criteria to be used to steer traffic, for example, per connection or per packet balancing, etc. For more info on how to use mangle marking see Firewall Marking examples.
- routing rules - a basic set of parameters that can be used to quickly steer traffic. This is the method we are going to use for our example.

It is not recommended to use both methods at the same time or you should know exactly what you are doing. If you really do need to use both mangle and routing rules in the same setup then keep in mind that mangle has higher priority, meaning if the mangle marked traffic can be resolved in the table then route rules will never see this traffic.

Routing table count is limited to 4096 unique tables.

Routing Rules

Routing rules allow steering traffic based on basic parameters like a source address, a destination address, or in-interface as well as other parameters.

For our example, we want to select traffic with destination 8.8.8.8 and do not fall back to the **main** table:

```
/routing rule add dst-address=8.8.8.8 action=lookup-only-in-table table=myTable
```

Lets's say that we know that customer is connected to ether4 and we want only that customer to route 8.8.8.8 to a specific gateway. We can use the following rule:

```
/routing rule add dst-address=8.8.8.8 action=lookup-only-in-table table=myTable interface=ether4
```

If for some reason the gateway used in our table goes down, the whole lockup will fail and the destination will not be reachable. In active-backup setups we want the traffic to be able to fall back to the **main** table. To do that change the action from `lookup-only-in-table` to `lookup`.

Also, routing rules can be used as a very "basic firewall". Let's say we do not want to allow a customer connected to ether4 to be able to access the 192.168.1.0/24 network:

```
/routing rule add dst-address=192.168.1.0/24 interface=ether4 action=drop
```

List of all the parameters that can be used by routing rules:

Property	Description
action (*drop* | *lookup* | *lookup-only-in-table* | *unreachable*)	An action to take on the matching packet: • drop - silently drop the packet. • lookup - perform a lookup in routing tables. • lookup-only-in-table - perform lookup only in the specified routing table (see table parameter). • unreachable - generate ICMP unreachable message and send it back to the source.
comment (*string*)	
disabled (*yes* | *no*)	The disabled rule is not used.
dst-address()	The destination address of the packet to match.
interface (*string*)	Incoming interface to match.
min-prefix (*integer [0..4294967295]*)	Equivalent to Linux IP rule `suppress_prefixlength` . For example to suppress the default route in the routing decision set the value to 0.
routing-mark (*string*)	Match specific routing mark.
src-address (*string*)	The source address of the packet to match.
table (*name*)	Name of the routing table to use for lookup.

Virtual Routing and Forwarding (VRF)

Description

RouterOS allows to create multiple Virtual Routing and Forwarding instances on a single router. This is useful for BGP-based MPLS VPNs. Unlike BGP VPLS, which is OSI Layer 2 technology, BGP VRF VPNs work in Layer 3 and as such exchange IP prefixes between routers. VRFs solve the problem of overlapping IP prefixes and provide the required privacy (via separated routing for different VPNs).

It is possible to set up vrf-lite setups or use multi-protocol BGP with VPNv4 address family to distribute routes from VRF routing tables - not only to other routers, but also to different routing tables in the router itself.

Configuration

VRF table is created in **`/ip vrf`** menu. After the VRF config is created routing table mapping is added (a dynamic table with the same name is created). Each active VRF will always have a mapped routing table.

```
[admin@arm-bgp] /ip/vrf> print
Flags: X - disabled; * - builtin
 0  * name="main" interfaces=all

[admin@arm-bgp] /routing/table> print
Flags: D - dynamic; X - disabled, I - invalid; U - used
 0 D   name="main" fib
```

Note that the order of the added VRFs is significant. To properly match which interface will belong to the VRF care must be taken to place VRFs in the correct order (matching is done starting from the top entry, just like firewall rules).

Since each VRF has mapped routing table, count of max unique VRFs is also limited to 4096.

Let's look at the following example:

```
[admin@arm-bgp] /ip/vrf> print
Flags: X - disabled; * - builtin
 0  * name="main" interfaces=all
 1    name="myVrf" interfaces=lo_vrf
```

Since the first entry is matching all the interfaces, the second VRF will not have any interfaces added. To fix the problem order of the entries must be changed.

```
[admin@arm-bgp] /ip/vrf> move 1 0
[admin@arm-bgp] /ip/vrf> print
Flags: X - disabled; * - builtin
 0    name="myVrf" interfaces=lo_vrf
 1  * name="main" interfaces=all
```

Connected routes from the interfaces assigned to the VRF will be installed in the right routing table automatically.

When the interface is assigned to the VRF as well as connected routes it does not mean that RouterOS services will magically know which VRF to use just by specifying the IP address in the configuration. Each service needs VRF support to be added and explicit configuration. Whether the service has VRF support and has VRF configuration options refer to appropriate service documentation.

For example, let's make an SSH service to listen for connections on the interfaces belonging to the VRF:

```
[admin@arm-bgp] /ip/service> set ssh vrf=myVrf
[admin@arm-bgp] /ip/service> print
Flags: X, I - INVALID
Columns: NAME, PORT, CERTIFICATE, VRF
#   NAME     PORT  CERTIFICATE  VRF
0   telnet     23               main
1   ftp        21
2   www        80               main
3   ssh        22               myVrf
4 X www-ssl   443  none         main
5   api      8728               main
6   winbox   8291               main
7   api-ssl  8729  none         main
```

Adding routes to the VRF is as simple as specifying the routing-table parameter when adding the route and specifying in which routing table to resolve the gateway by specifying @name after the gateway IP:

```
/ip route add dst-address=192.168.1.0/24 gateway=172.16.1.1@myVrf routing-table=myVrf
```

Traffic leaking between VRFs is possible if the gateway is explicitly set to be resolved in another VRF, for example:

```
# add route in the myVrf, but resolve the gateway in the main table
/ip route add dst-address=192.168.1.0/24 gateway=172.16.1.1@main routing-table=myVrf

# add route in the main table, but resolve the gateway in the myVrf
/ip route add dst-address=192.168.1.0/24 gateway=172.16.1.1@myVrf
```


If the gateway configuration does not have an explicitly configured table to be resolved in, then it is considered, that gateway should be resolved in the "main" table.

VRF interfaces in firewall

Before RouterOS version 7.14, firewall filter rules with the property in/out-interface would apply to interfaces within a VRF instance. Starting from RouterOS version 7.14, these rules no longer target individual interfaces within a VRF, but rather the VRF interface as a whole.

Started from version 7.14 when interfaces are added in VRF - virtual VRF interface is created automatically. If it is needed to match traffic which belongs to VRF interface, VRF virtual interface should be used in firewall filters, for example:

```
/ip vrf add interfaces=ether5 name=vrf5
/ip firewall filter add chain=input in-interface=vrf5 action=accept
```

If there are several interfaces in one VRF but it is needed to match only one of these interfaces - marks should be used. For example:

```
/ip vrf add interface=ether15,ether16 vrf=vrf1516
/ip firewall mangle
add action=mark-connection chain=prerouting connection-state=new in-interface=ether15 new-connection-
mark=input_allow passthrough=yes
/ip firewall filter
add action=accept chain=input connection-mark=input_allow
```

Supported features

Different services can be placed in specific VRF on which the service is listening for incoming or creating outgoing connections. By default, all services are using the `main` table, but it can be changed with a separate `vrf` parameter or by specifying the VRF name separated by "@" at the end of the IP address.

Below is the list of supported services.

Feature	Support	Comment

BGP	+	`/routing bgp template`
		`add name=bgp-template1 vrf=vrf1`
		`/routing bgp vpls`
		`add name=bgp-vpls1 site-id=10 vrf=vrf1`
		`/routing bgp vpn`
		`add label-allocation-policy=per-vrf vrf=vrf1`

E-mail	+	`/tool e-mail` `set address=192.168.88.1 vrf=vrf1`

IP Services	+	VRF is supported for `telnet`, `www`, `ssh`, `www-ssl`, `api`, `winbox`, `api-ssl` services. The `ftp` service does not support changing the VRF.
		`/ip service`
		`set telnet vrf=vrf1`

L2TP Client	+	`/interface l2tp-client` `add connect-to=192.168.88.1@vrf1 name=l2tp-out1 user=l2tp-client`

MPLS	+	`/mpls ldp` `add vrf=vrf1`
Netwatch	+	`/tool netwatch` `add host=192.168.88.1@vrf1`

NTP	+	`/system ntp client`
		`set vrf=vrf1`
		`/system ntp server`
		`set vrf=vrf1`

OSPF	+	`/routing ospf instance` `add disabled=no name=ospf-instance-1 vrf=vrf1`
ping	+	`/ping 192.168.88.1 vrf=vrf1`
RADIUS	+	`/radius add address=192.168.88.1@vrf1` `/radius incoming set vrf=vrf1`
RIP	+	`/routing rip instance` `add name=rip-instance-1 vrf=vrf1`
RPKI	+	`/routing rpki` `add vrf=vrf1`
SNMP	+	`/snmp` `set vrf=vrf1`
EoIP	+	`/interface eoip` `add remote-address=192.168.1.1@vrf1`
IPIP	+	`/interface ipip` `add remote-address=192.168.1.1@vrf1`
GRE	+	`/interface gre` `add remote-address=192.168.1.1@vrf1`

SSTP-client	+	`/interface sstp-client` `add connect-to=192.168.1.1@vrf1`
OVPN-client	+	`/interface ovpn-client` `add connect-to=192.168.1.1@vrf1`
L2TP-ether	+	`/interface l2tp-ether` `add connect-to=192.168.2.2@vrf`
VXLAN	+	`/interface vxlan` `add vni=10 vrf=vrf1`
fetch		`/tool/fetch` `address=10.155.28.236@vrf1 mode=ftp src-path=my_file.pcap user=admin password=""`

Examples

Simple VRF-Lite setup

Let's consider a setup where we need two customer VRFs that require access to the internet:

```
/ip address
add address=172.16.1.2/24 interface=public
add address=192.168.1.1/24 interface=ether1
add address=192.168.2.1/24 interface=ether2

/ip route
add gateway=172.16.1.1

# add VRF configuration
/ip vrf
add name=cust_a interface=ether1 place-before 0
add name=cust_b interface=ether2 place-before 0

# add vrf routes
/ip route
add gateway=172.16.1.1@main routing-table=cust_a
add gateway=172.16.1.1@main routing-table=cust_b

# masquerade local source
/ip firewall nat add chain=srcnat out-interface=public action=masquerade
```

It might be necessary to ensure that packets coming in the "public" interface can actually reach the correct VRF.
This can be solved by marking new connections originated by the VRF customers and steering the traffic by routing marks of incoming packets on the "public" interface.

```
# mark new customer connections
/ip firewall mangle
add action=mark-connection chain=prerouting connection-state=new new-connection-mark=\
    cust_a_conn src-address=192.168.1.0/24 passthrough=no
add action=mark-connection chain=prerouting connection-state=new new-connection-mark=\
    cust_b_conn src-address=192.168.2.0/24 passthrough=no

# mark routing
/ip firewall mangle
add action=mark-routing chain=prerouting connection-mark=cust_a_conn \
    in-interface=public new-routing-mark=cust_a
add action=mark-routing chain=prerouting connection-mark=cust_b_conn \
    in-interface=public new-routing-mark=cust_b
```

Static inter-VRF routes

In general, it is recommended that all routes between VRF should be exchanged using BGP local import and export functionality. If that is not enough, static routes can be used to achieve this so-called route leaking.

There are two ways to install a route that has a gateway in a different routing table than the route itself.

The first way is to explicitly specify the routing table in the gateway field when adding a route. This is only possible when leaking a route and gateway from the "main" routing table to a different routing table (VRF). Example:

```
# add route to 5.5.5.0/24 in 'vrf1' routing table with gateway in the main routing table
add dst-address=5.5.5.0/24 gateway=10.3.0.1@main routing-table=vrf1
```

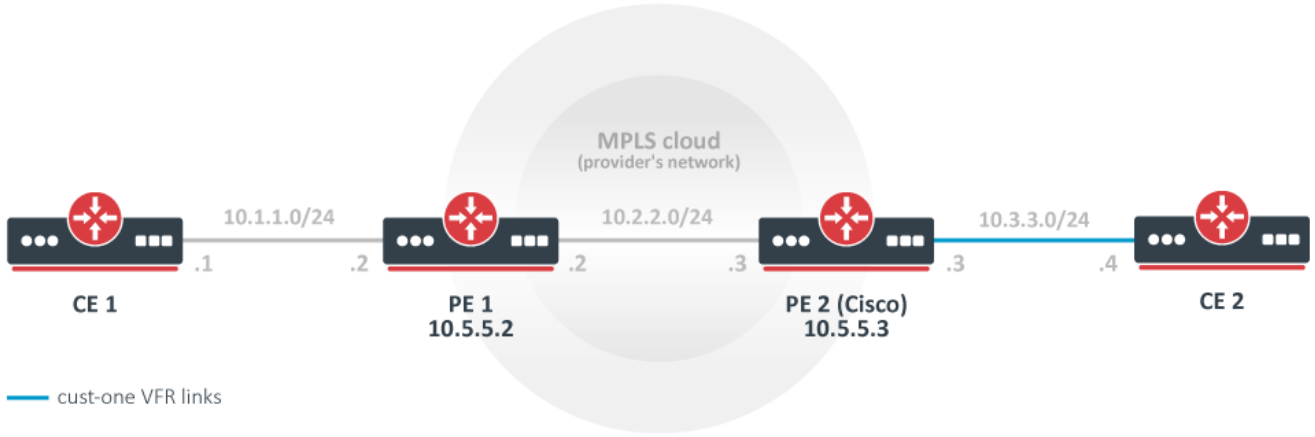
The second way is to explicitly specify the interface in the gateway field. The interface specified can belong to a VRF instance. Example:

```
# add route to 5.5.5.0/24 in the main routing table with gateway at 'ether2' VRF interface
add dst-address=5.5.5.0/24 gateway=10.3.0.1%ether2 routing-table=main
# add route to 5.5.5.0/24 in the main routing table with 'ptp-link-1' VRF interface as gateway
add dst-address=5.5.5.0/24 gateway=ptp-link-1 routing-table=main
```

As can be observed, there are two variations possible - to specify gateway as *ip_address%interface* or to simply specify an *interface*. The first should be used for broadcast interfaces in most cases. The second should be used for point-to-point interfaces, and also for broadcast interfaces, if the route is a connected route in some VRF. For example, if you have an address `1.2.3.4/24` on interface *ether2* that is put in a VRF, there will be a connected route to `1.2.3.0/24` in that VRF's routing table. It is acceptable to add a static route `1.2.3.0/24` in a different routing table with an interface-only gateway, even though *ether2* is a broadcast interface:

```
add dst-address=1.2.3.0/24 gateway=ether2 routing-table=main
```

The simplest MPLS VPN setup



In this example, a rudimentary MPLS backbone (consisting of two Provider Edge (PE) routers PE1 and PE2) is created and configured to forward traffic between Customer Edge (CE) routers CE1 and CE2 routers that belong to *cust-one* VPN.

CE1 Router

```
/ip address add address=10.1.1.1/24 interface=ether1
# use static routing
/ip route add dst-address=10.3.3.0/24 gateway=10.1.1.2
```

CE2 Router

```
/ip address add address=10.3.3.4/24 interface=ether1
/ip route add dst-address=10.1.1.0/24 gateway=10.3.3.3
```

PE1 Router

```
/interface bridge add name=lobridge
/ip address add address=10.1.1.2/24 interface=ether1
/ip address add address=10.2.2.2/24 interface=ether2
/ip address add address=10.5.5.2/32 interface=lobridge
/ip vrf add name=cust-one interfaces=ether1
/mpls ldp add enabled=yes transport-address=10.5.5.2 lsr-id=10.5.5.2
/mpls ldp interface add interface=ether2
/routing bgp template set default as=65000

/routing bgp vpn
add vrf=cust-one \
  route-distinguisher=1.1.1.1:111 \
  import.route-targets=1.1.1.1:111 \
  import.router-id=cust-one \
  export.redistribute=connected \
  export.route-targets=1.1.1.1:111 \
  label-allocation-policy=per-vrf
/routing bgp connection
add template=default remote.address=10.5.5.3 address-families=vpnv4 local.address=10.5.5.2

# add route to the remote BGP peer's loopback address
/ip route add dst-address=10.5.5.3/32 gateway=10.2.2.3
```

PE2 Router (Cisco)

```
ip vrf cust-one
rd 1.1.1.1:111
route-target export 1.1.1.1:111
route-target import 1.1.1.1:111
exit

interface Loopback0
ip address 10.5.5.3 255.255.255.255

mpls ldp router-id Loopback0 force
mpls label protocol ldp

interface FastEthernet0/0
ip address 10.2.2.3 255.255.255.0
mpls ip

interface FastEthernet1/0
ip vrf forwarding cust-one
ip address 10.3.3.3 255.255.255.0

router bgp 65000
neighbor 10.5.5.2 remote-as 65000
neighbor 10.5.5.2 update-source Loopback0
address-family vpnv4
neighbor 10.5.5.2 activate
neighbor 10.5.5.2 send-community both
exit-address-family
address-family ipv4 vrf cust-one
redistribute connected
exit-address-family

ip route 10.5.5.2 255.255.255.255 10.2.2.2
```

Results

Check that VPNv4 route redistribution is working:

```
[admin@PE1] /routing/route> print detail where afi="vpn4"
Flags: X - disabled, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, a - ldp-address, l - l
dp-mapping, g - slaac, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp, B - blackhole
 Ab   afi=vpn4 contribution=active dst-address=111.16.0.0/24&1.1.1.1:111 routing-table=main label=16
       gateway=111.111.111.4 immediate-gw=111.13.0.2%ether9 distance=200 scope=40 target-scope=30
       belongs-to="bgp-VPN4-111.111.111.4"
       bgp.peer-cache-id=*2C00011 .as-path="65511" .ext-communities=rt:1.1.1.1:111 .local-pref=100
       .atomic-aggregate=yes .origin=igp
       debug.fwp-ptr=0x202427E0

[admin@PE1] /routing/bgp/advertisements> print
 0 peer=to-pe2-1 dst=10.1.1.0/24 local-pref=100 origin=2 ext-communities=rt:1.1.1.1:111 atomic-aggregate=yes
```

Check that the 10.3.3.0 is installed in IP routes, in the cust-one route table:

```
[admin@PE1] > /ip route print where routing-table="cust-one"
Flags: D - DYNAMIC; A - ACTIVE; c, b, y - BGP-MPLS-VPN
Columns: DST-ADDRESS, GATEWAY, DISTANCE
# DST-ADDRESS     GATEWAY         DISTANCE
0 ADC 10.1.1.0/24 ether1@cust-one        0
1 ADb 10.3.3.0/24 10.5.5.3              20
```

Let's take a closer look at IP routes in cust-one VRF. The 10.1.1.0/24 IP prefix is a connected route that belongs to an interface that was configured to belong to cust-one VRF. The 10.3.3.0/24 IP prefix was advertised via BGP as a VPNv4 route from PE2 and is imported in this VRF routing table, because our configured **import-route-targets** matched the BGP extended communities attribute it was advertised with.

```
[admin@PE1] /routing/route> print detail where routing-table="cust-one"
Flags: X - disabled, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, a - ldp-address, l - l
dp-mapping, g - slaac, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp, B - blackhole
 Ac   afi=ip4 contribution=active dst-address=10.1.1.0/24 routing-table=cust-one
       gateway=ether1@cust-one immediate-gw=ether1 distance=0 scope=10 belongs-to="connected"
       local-address=10.1.1.2%ether1@cust-one
       debug.fwp-ptr=0x202420C0

 Ay   afi=ip4 contribution=active dst-address=10.3.3.0/24 routing-table=cust-one label=16
       gateway=10.5.5.3 immediate-gw=10.2.2.3%ether2 distance=20 scope=40 target-scope=30
       belongs-to="bgp-mpls-vpn-1-bgp-VPN4-10.5.5.3-import"
       bgp.peer-cache-id=*2C00011 .ext-communities=rt:1.1.1.1:111 .local-pref=100
       .atomic-aggregate=yes .origin=igp
       debug.fwp-ptr=0x20242840


[admin@PE1] /routing/route> print detail where afi="vpn4"
Flags: X - disabled, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, a - ldp-address, l - l
dp-mapping, g - slaac, y - bgp-mpls-vpn;
H - hw-offloaded; + - ecmp, B - blackhole
 Ay   afi=vpn4 contribution=active dst-address=10.1.1.0/24&1.1.1.1:111 routing-table=main label=19
       gateway=ether1@cust-one immediate-gw=ether1 distance=200 scope=40 target-scope=10
       belongs-to="bgp-mpls-vpn-1-connected-export"
       bgp.ext-communities=rt:1.1.1.1:1111 .atomic-aggregate=no .origin=incomplete
       debug.fwp-ptr=0x202426C0

 Ab   afi=vpn4 contribution=active dst-address=10.3.3.0/24&1.1.1.1:111 routing-table=main label=16
       gateway=10.5.5.3 immediate-gw=10.2.2.3%ether2 distance=200 scope=40 target-scope=30
       belongs-to="bgp-VPN4-10.5.5.3"
       bgp.peer-cache-id=*2C00011 .ext-communities=rt:1.1.1.1:111 .local-pref=100
       .atomic-aggregate=yes .origin=igp
       debug.fwp-ptr=0x202427E0
```

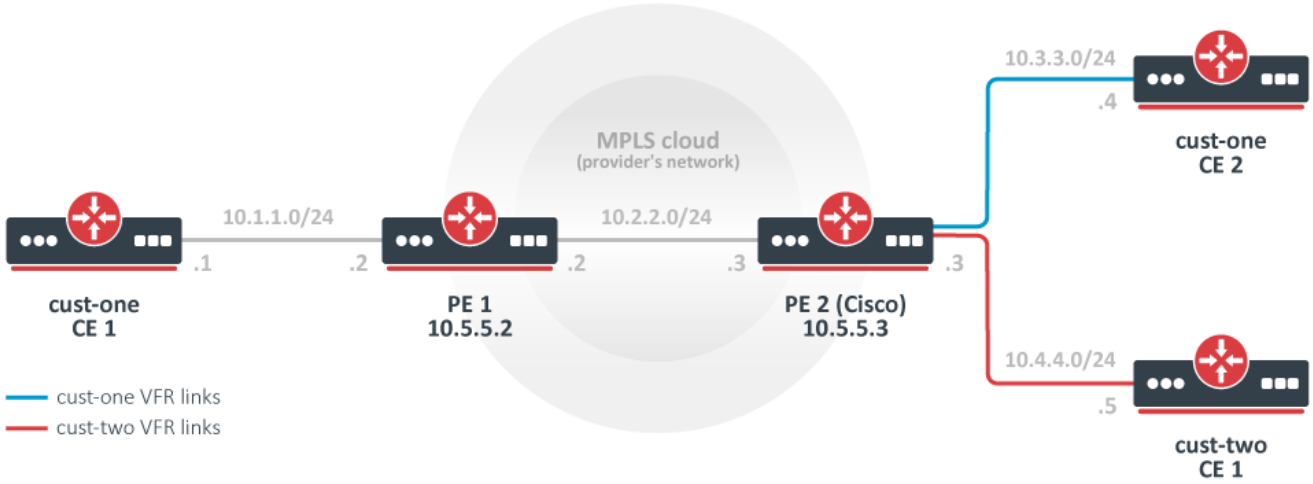
The same for Cisco:

```
PE2#show ip bgp vpnv4 all
BGP table version is 5, local router ID is 10.5.5.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete
Network Next Hop Metric LocPrf Weight Path
Route Distinguisher: 1.1.1.1:111 (default for vrf cust-one)
*>i10.1.1.0/24 10.5.5.2 100 0 ?
*> 10.3.3.0/24 0.0.0.0 0 32768 ?

PE2#show ip route vrf cust-one
Routing Table: cust-one
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
10.0.0.0/24 is subnetted, 1 subnets
B 10.1.1.0 [200/0] via 10.5.5.2, 00:05:33
10.0.0.0/24 is subnetted, 1 subnets
C 10.3.3.0 is directly connected, FastEthernet1/0
```

You should be able to ping from CE1 to CE2 and vice versa.

```
[admin@CE1] > /ping 10.3.3.4
10.3.3.4 64 byte ping: ttl=62 time=18 ms
10.3.3.4 64 byte ping: ttl=62 time=13 ms
10.3.3.4 64 byte ping: ttl=62 time=13 ms
10.3.3.4 64 byte ping: ttl=62 time=14 ms
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 13/14.5/18 ms
```

A more complicated setup (changes only)



As opposed to the simplest setup, in this example, we have two customers: cust-one and cust-two.

We configure two VPNs for them, cust-one and cust-two respectively, and exchange all routes between them. (This is also called "route leaking").

Note that this could be not the most typical setup, because routes are usually not exchanged between different customers. In contrast, by default, it should not be possible to gain access from one VRF site to a different VRF site in another VPN. (This is the "Private" aspect of VPNs.) Separate routing is a way to provide privacy, and it is also required to solve the problem of overlapping IP network prefixes. Route exchange is in direct conflict with these two requirements but may sometimes be needed (e.g. temp. solution when two customers are migrating to a single network infrastructure).

CE1 Router, *cust-one*

```
/ip route add dst-address=10.4.4.0/24 gateway=10.1.1.2
```

CE2 Router, *cust-one*

```
/ip route add dst-address=10.4.4.0/24 gateway=10.3.3.3
```

CE1 Router,*cust-two*

```
/ip address add address=10.4.4.5 interface=ether1
/ip route add dst-address=10.1.1.0/24 gateway=10.3.3.3
/ip route add dst-address=10.3.3.0/24 gateway=10.3.3.3
```

PE1 Router

```
# replace the old BGP VPN with this:
/routing bgp vpn
add vrf=cust-one \
  export.redistribute=connected \
  route-distinguisher=1.1.1.1:111 \
  import.route-targets=1.1.1.1:111,2.2.2.2:222  \
  export.route-targets=1.1.1.1:111
```

PE2 Router (Cisco)

```
ip vrf cust-one
rd 1.1.1.1:111
route-target export 1.1.1.1:111
route-target import 1.1.1.1:111
route-target import 2.2.2.2:222
exit

ip vrf cust-two
rd 2.2.2.2:222
route-target export 2.2.2.2:222
route-target import 1.1.1.1:111
route-target import 2.2.2.2:222
exit

interface FastEthernet2/0
ip vrf forwarding cust-two
ip address 10.4.4.3 255.255.255.0

router bgp 65000
address-family ipv4 vrf cust-two
redistribute connected
exit-address-family
```

Variation: replace the Cisco with another MT

PE2 Mikrotik config

```
/interface bridge add name=lobridge
/ip address
add address=10.2.2.3/24 interface=ether1
add address=10.3.3.3/24 interface=ether2
add address=10.4.4.3/24 interface=ether3
add address=10.5.5.3/32 interface=lobridge
/ip vrf
add name=cust-one interfaces=ether2
add name=cust-two interfaces=ether3
/mpls ldp add enabled=yes transport-address=10.5.5.3
/mpls ldp interface add interface=ether1

/routing bgp template set default as=65000
/routing bgp vpn
add vrf=cust-one \
  export.redistribute=connected \
  route-distinguisher=1.1.1.1:111 \
  import.route-targets=1.1.1.1:111,2.2.2.2:222 \
  export.route-targets=1.1.1.1:111 \
add vrf=cust-two \
  export.redistribute=connected \
  route-distinguisher=2.2.2.2:222 \
  import.route-targets=1.1.1.1:111,2.2.2.2:222 \
  export.route-targets=2.2.2.2:222 \

/routing bgp connection
add template=default remote.address=10.5.5.2 address-families=vpnv4 local.address=10.5.5.3

# add route to the remote BGP peer's loopback address
/ip route add dst-address=10.5.5.2/32 gateway=10.2.2.2
```

Results

The output of **/ip route print** now is interesting enough to deserve detailed observation.

```
[admin@PE2] /ip route> print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
# DST-ADDRESS PREF-SRC GATEWAY DISTANCE
0 ADb 10.1.1.0/24 10.5.5.2 recurs... 20
1 ADC 10.3.3.0/24 10.3.3.3 ether2 0
2 ADb 10.4.4.0/24 20
3 ADb 10.1.1.0/24 10.5.5.2 recurs... 20
4 ADb 10.3.3.0/24 20
5 ADC 10.4.4.0/24 10.4.4.3 ether3 0
6 ADC 10.2.2.0/24 10.2.2.3 ether1 0
7 A S 10.5.5.2/32 10.2.2.2 reacha... 1
8 ADC 10.5.5.3/32 10.5.5.3 lobridge 0
```

The route 10.1.1.0/24 was received from a remote BGP peer and is installed in both VRF routing tables.

The routes 10.3.3.0/24 and 10.4.4.0/24 are also installed in both VRF routing tables. Each is a connected route in one table and a BGP route in another table. This has nothing to do with their being advertised via BGP. They are simply being "advertised" to the local VPNv4 route table and locally reimported after that. Import and export **route-targets** determine in which tables they will end up.

This can be deduced from its attributes - they don't have the usual BGP properties. (Route 10.4.4.0/24.)

```
[admin@PE2] /routing/route> print detail where routing-table=cust-one
...
```

Leaking routes between VRFs

Currently, there is no mechanism to leak routes from one VRF instance to another within the same router.

As a workaround, it is possible to create a tunnel between two locally configure loopback addresses and assign each tunnel endpoint to its own VRF. Then it is possible to run either dynamic routing protocols or set up static routes to leak between both VRFs.

The downside of this approach is that tunnel must be created between each VRF where routes should be leaked (create a full mesh), which significantly complicates configuration even if there are just several VRFs, not to mention more complicated setups.

For example, to leak routes between 5 VRFs it would require n * (n – 1) / 2 connections, which will lead to the setup with 20 tunnel endpoints and 20 OSPF instances on one router.

Example config with two VRFs of this method:

```
/interface bridge
add name=dummy_custC
add name=dummy_custB
add name=lo1
add name=lo2

/ip address
add address=111.255.255.1 interface=lo1 network=111.255.255.1
add address=111.255.255.2 interface=lo2 network=111.255.255.2
add address=172.16.1.0/24 interface=dummy_custC network=172.16.1.0
add address=172.16.2.0/24 interface=dummy_custB network=172.16.2.0

/interface ipip
add local-address=111.255.255.1 name=ipip-tunnel1 remote-address=111.255.255.2
add local-address=111.255.255.2 name=ipip-tunnel2 remote-address=111.255.255.1

/ip address
add address=192.168.1.1/24 interface=ipip-tunnel1 network=192.168.1.0
add address=192.168.1.2/24 interface=ipip-tunnel2 network=192.168.1.0

/ip vrf
add interfaces=ipip-tunnel1,dummy_custC name=custC
add interfaces=ipip-tunnel2,dummy_custB name=custB

/routing ospf instance
add disabled=no name=i2_custB redistribute=connected,static,copy router-id=192.168.1.1 routing-table=custB
vrf=custB
add disabled=no name=i2_custC redistribute=connected router-id=192.168.1.2 routing-table=custC vrf=custC
/routing ospf area
add disabled=no instance=i2_custB name=custB_bb
add disabled=no instance=i2_custC name=custC_bb
/routing ospf interface-template
add area=custB_bb disabled=no networks=192.168.1.0/24
add area=custC_bb disabled=no networks=192.168.1.0/24
```

Result:

```
[admin@rack1_b36_CCR1009] /routing/ospf/neighbor> print
Flags: V - virtual; D - dynamic
 0  D instance=i2_custB area=custB_bb address=192.168.1.1 priority=128 router-id=192.168.1.2 dr=192.168.1.1
bdr=192.168.1.2
      state="Full" state-changes=6 adjacency=41m28s timeout=33s

 1  D instance=i2_custC area=custC_bb address=192.168.1.2 priority=128 router-id=192.168.1.1 dr=192.168.1.1
bdr=192.168.1.2
      state="Full" state-changes=6 adjacency=41m28s timeout=33s


[admin@rack1_b36_CCR1009] /ip/route> print where routing-table=custB
Flags: D - DYNAMIC; A - ACTIVE; c, s, o, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
     DST-ADDRESS       GATEWAY                         DISTANCE
  DAo 172.16.1.0/24     192.168.1.1%ipip-tunnel2@custB       110
  DAc 172.16.2.0/24     dummy_custB@custB                      0
  DAc 192.168.1.0/24    ipip-tunnel2@custB                     0


[admin@rack1_b36_CCR1009] > /ip route/print where routing-table=custC
Flags: D - DYNAMIC; A - ACTIVE; c, o, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
    DST-ADDRESS       GATEWAY                        DISTANCE
  DAc 172.16.1.0/24     dummy_custC@custC                      0
  DAo 172.16.2.0/24     192.168.1.2%ipip-tunnel1@custC       110
  DAc 192.168.1.0/24    ipip-tunnel1@custC                     0
```

References

RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs)

MPLS Fundamentals, chapter 7, *Luc De Ghein*, Cisco Press 2006

OSPF

Overview

OSPF is Interior Gateway Protocol (IGP) designed to distribute routing information between routers belonging to the same Autonomous System (AS).

The protocol is based on link-state technology that has several advantages over distance-vector protocols such as RIP:

- no hop count limitations;
- multicast addressing is used to send routing information updates;
- updates are sent only when network topology changes occur;
- the logical definition of networks where routers are divided into areas
- transfers and tags external routes injected into AS.

However, there are a few disadvantages:

- OSPF is quite CPU and memory intensive due to the SPF algorithm and maintenance of multiple copies of routing information;
- more complex protocol to implement compared to RIP;

RouterOS implements the following standards:

- RFC 2328 - OSPF Version 2
- RFC 3101 - The OSPF Not-So-Stubby Area (NSSA) Option
- RFC 3630 - Traffic Engineering (TE) Extensions to OSPF Version 2
- RFC 4577 - OSPF as the Provider/Customer Edge Protocol for BGP/MPLS IP Virtual Private Networks (VPNs)
- RFC 5329 - Traffic Engineering Extensions to OSPF Version 3
- RFC 5340 - OSPF for IPv6
- RFC 5643 - Management Information Base for OSPFv3
- RFC 6549 - OSPFv2 Multi-Instance Extensions
- RFC 6565 - OSPFv3 as a Provider Edge to Customer Edge (PE-CE) Routing Protocol
- RFC 6845 - OSPF Hybrid Broadcast and Point-to-Multipoint Interface Type
- RFC 7471 - OSPF Traffic Engineering (TE) Metric Extensions

OSPF Terminology

Before we move on let's familiarise ourselves with terms important for understanding the operation of the OSPF. These terms will be used throughout the article.

- **Neighbor** - connected (adjacent) router that is running OSPF with the adjacent interface assigned to the same area. Neighbors are found by Hello packets (unless manually configured).
- **Adjacency** - logical connection between a router and its corresponding DR and BDR. No routing information is exchanged unless adjacencies are formed.
- **Link** - link refers to a network or router interface assigned to any given network.
- **Interface** - physical interface on the router. The interface is considered a link when it is added to OSPF. Used to build link database.
- **LSA** - Link State Advertisement, data packet contains link-state and routing information, that is shared among OSPF Neighbors.
- **DR** - Designated Router, chosen router to minimize the number of adjacencies formed. The option is used in broadcast networks.
- **BDR** -Backup Designated Router, hot standby for the DR. BDR receives all routing updates from adjacent routers, but it does not flood LSA updates.
- **Area** - areas are used to establish a hierarchical network.
- **ABR** - Area Border Router, router connected to multiple areas. **ABRs** are responsible for **summarization** and **update suppression** between connected areas.
- **ASBR** - Autonomous System Boundary Router, router connected to an external network (in a different AS). If you import other protocol routes into OSPF from the router it is now considered ASBR.
- **NBMA** - Non-broadcast multi-access, networks allow multi-access but have no broadcast capability. Additional OSPF neighbor configuration is required for those networks.
- **Broadcast** - Network that allows broadcasting, for example, Ethernet.
- **Point-to-point** - Network type eliminates the need for DRs and BDRs
- **Router-ID** - IP address used to identify the OSPF router. If the OSPF Router-ID is not configured manually, a router uses one of the IP addresses assigned to the router as its Router-ID.
- **Link State** - The term link-state refers to the status of a link between two routers. It defines the relationship between a router's interface and its neighboring routers.
- **Cost** - Link-state protocols assign a value to each link called cost. the cost value depends on the speed of the media. A cost is associated with the outside of each router interface. This is referred to as interface output cost.
- **Autonomous System** - An autonomous system is a group of routers that use a common routing protocol to exchange routing information.

Basic Configuration Example

To start OSPF v2 and v3 instances, the first thing to do is to add the instance and the backbone area:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.2.3.4
add name=v3inst version=3 router-id=1.2.3.4
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=backbone_v3 area-id=0.0.0.0 instance=v3inst
```

At this point, we can add a template. The template is used to match interfaces on which OSPF should be running, it can be done either by specifying the network or interface directly.

```
/routing ospf interface-template
add networks=192.168.0.0/24 area=backbone_v2
add networks=2001:db8::/64 area=backbone_v3
add interfaces=ether1 area=backbone_v3
```

Routing Table Calculation

Link state database describes the routers and links that interconnect them and are appropriate for forwarding. It also contains the cost (metric) of each link. This metric is used to calculate the shortest path to the destination network.
Each router can advertise a different cost for the router's own link direction, making it possible to have asymmetric links (packets to the destination travel over one path, but the response travels a different path). Asymmetric paths are not very popular, because it makes it harder to find routing problems.
The value of the cost can be changed in the OSPF interface template configuration menu, for example, to add an ether2 interface with a cost of 100:

```
/routing ospf interface-template
add interfaces=ether2 cost=100 area=backbone_v2
```

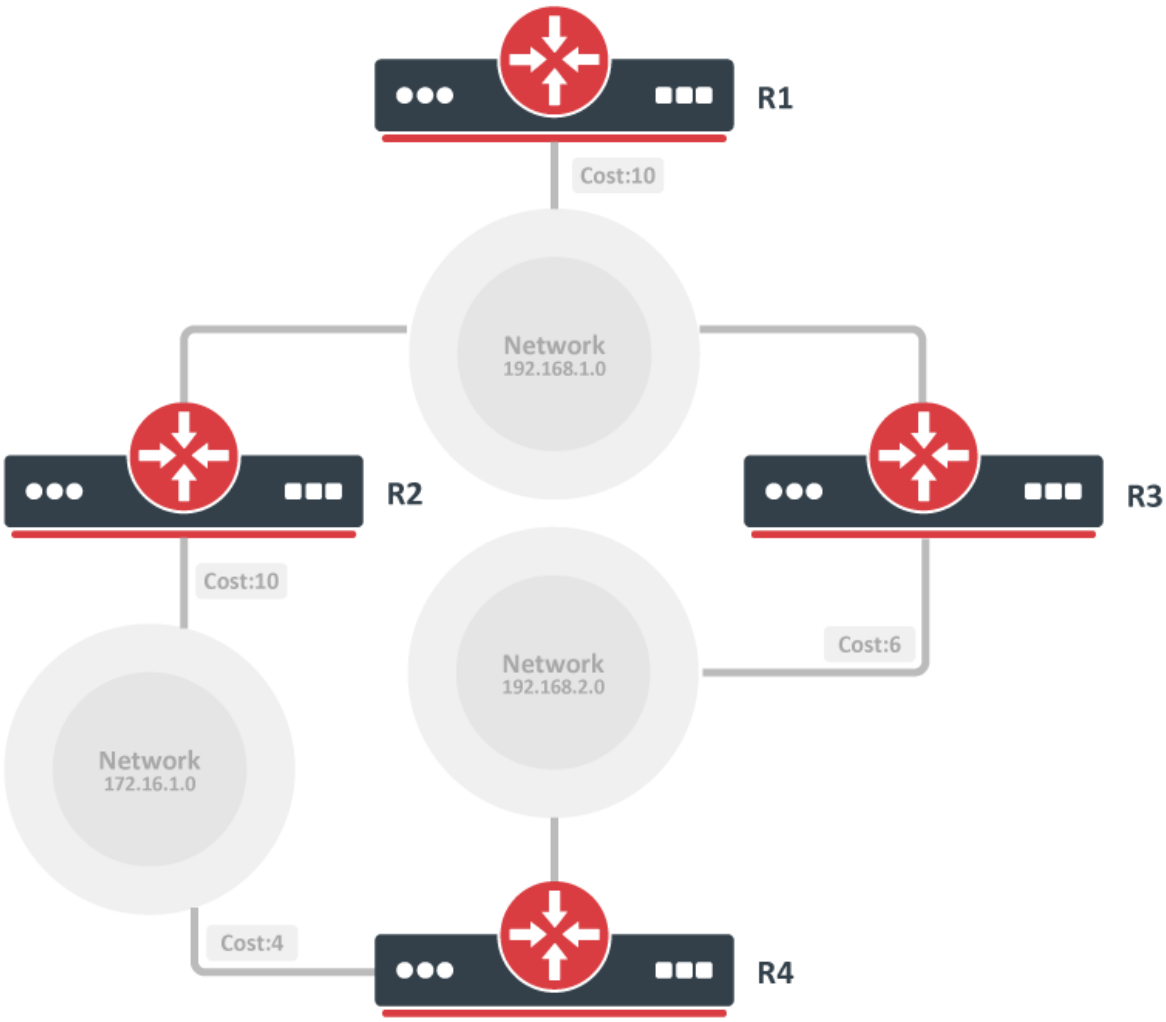
The cost of an interface on Cisco routers is inversely proportional to the bandwidth of that interface. A higher bandwidth indicates a lower cost. If similar costs are necessary on RouterOS, then use the following formula:

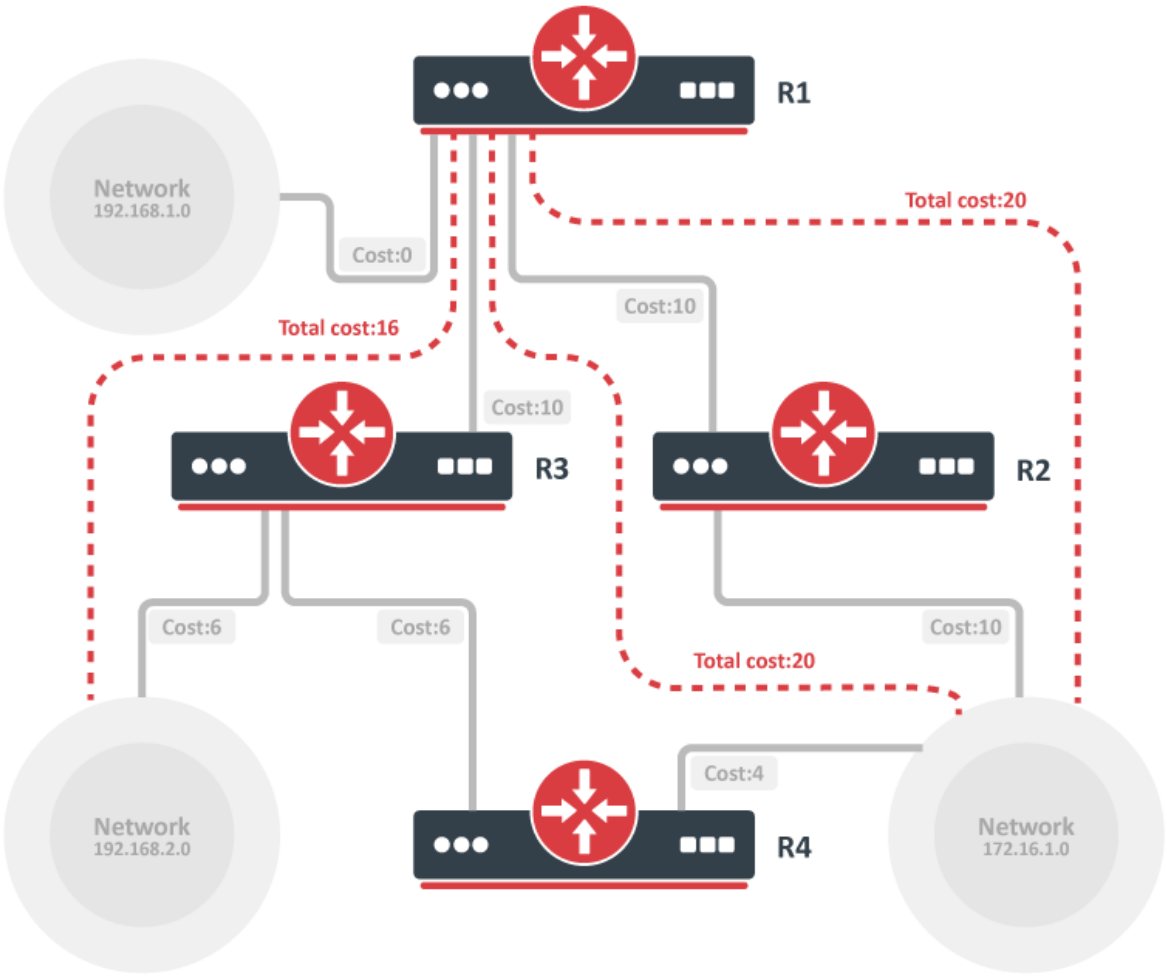
Cost = 100000000/bw in bps.

OSPF router is using Dijkstra's Shortest Path First (SPF) algorithm to calculate the shortest path. The algorithm places the router at the root of a tree and calculates the shortest path to each destination based on the cumulative cost required to reach the destination. Each router calculates its own tree even though all routers are using the same link-state database.

SPT Calculation

Assume we have the following network. The network consists of 4(four) routers. OSPF costs for outgoing interfaces are shown near the line that represents the link. In order to build the shortest-path tree for router R1, we need to make R1 the root and calculate the smallest cost for each destination.


R1
Cost:10
Network
192.168.1.0
R2
R3
Cost:10
Network
192.168.2.0
Cost:6
Network
172.16.1.0
Cost:4
R4



As you can see from the image above multiple shortest paths have been found to the 172.16.1.0 network, allowing load balancing of the traffic to that destination called equal-cost multipath (ECMP). After the shortest-path tree is built, a router starts to build the routing table accordingly. Networks are reached consequently to the cost calculated in the tree.

Routing table calculation looks quite simple, however, when some of the OSPF extensions are used or OSPF areas are calculated, routing calculation gets more complicated.

Forwarding Address

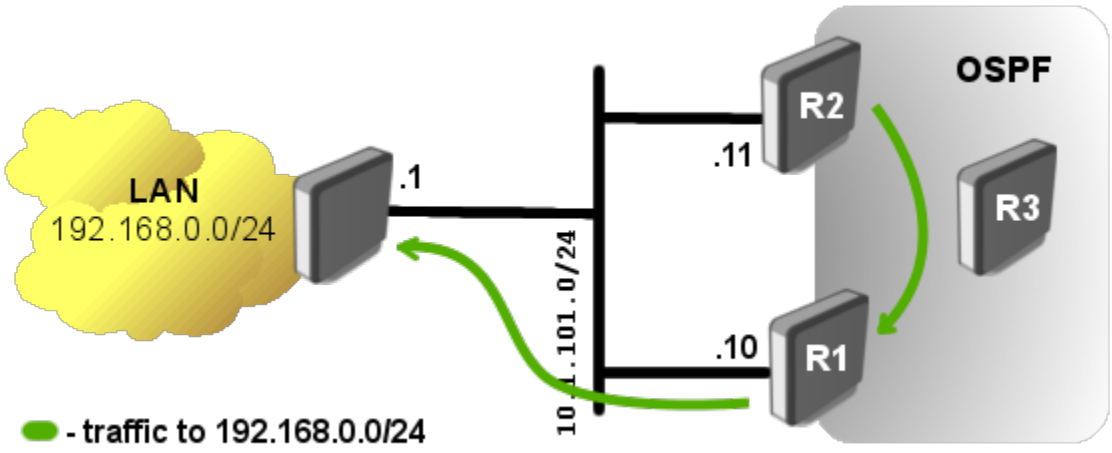
OSPF router can set the **forwarding-address** to something other than itself which indicates that an alternate next-hop is possible. Mostly forwarding address is set to **0.0.0.0** suggesting that the route is reachable only via the advertising router.

The forwarding address is set in LSA if the following conditions are met:

- OSPF must be enabled on the next-hop interface
- Next-hop address falls into the network provided by OSPF networks

A router that receives such LSA can use a forwarding address if OSPF is able to resolve the forwarding address. If forwarding address is not resolved directly - router sets nexthop for forwading address from LSA as a gateway, if forwarding address is not resolved at all - the gateway will be originator-id. Resolve happens only using OSPF instance routes, not the whole routing table.

Let's look at the example setup below:



Router **R1** has a static route to the external network *192.168.0.0/24*. OSPF is running between R1, R2, and R3, and the static route is distributed across the OSPF network.

The problem in such a setup is obvious, R2 can not reach the external network directly. Traffic going to the LAN network from **R2** will be forwarded over the router **R1**, but if we look at the network diagram we can see that more R2 can directly reach the router where the LAN network i located.

So knowing the forwarding address conditions, we can make router **R1** to set the forwarding address. We simply need to add 10.1.101.0/24 network to OSPF networks in the router's **R1** configuration:

```
/routing/ospf/interface-template add area=backbone_v2 networks=10.1.101.0/24
```

Now lets verify that forwarding address is actually working:

```
[admin@r2] /ip/route> print where dst-address=192.168.0.0/24
Flags: D - DYNAMIC; A - ACTIVE; o, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
    DST-ADDRESS       GATEWAY            DISTANCE
DAo 192.168.0.0/24    10.1.101.1%ether1       110
```

On all OSPF routers you will see LSA set with forwarding address other than 0.0.0.0

```
[admin@r2] /routing/ospf/lsa> print where id=192.168.0.0
Flags: S - self-originated, F - flushing, W - wraparound; D - dynamic

 1  D instance=default_ip4 type="external" originator=10.1.101.10 id=192.168.0.0
      sequence=0x80000001 age=19 checksum=0xF336 body=
        options=E
        netmask=255.255.255.0
        forwarding-address=10.1.101.1
        metric=10 type-1
        route-tag=0
```

⚠ OSPF adjacency between routers in the 10.1.101.0/24 network is not required

Neighbour Relationship and Adjacency

OSPF is a link-state protocol that assumes that the interface of the router is considered an OSPF link. Whenever OSPF is started, it adds the state of all the links in the local **link-state database**.

There are several steps before the OSPF network becomes fully functional:

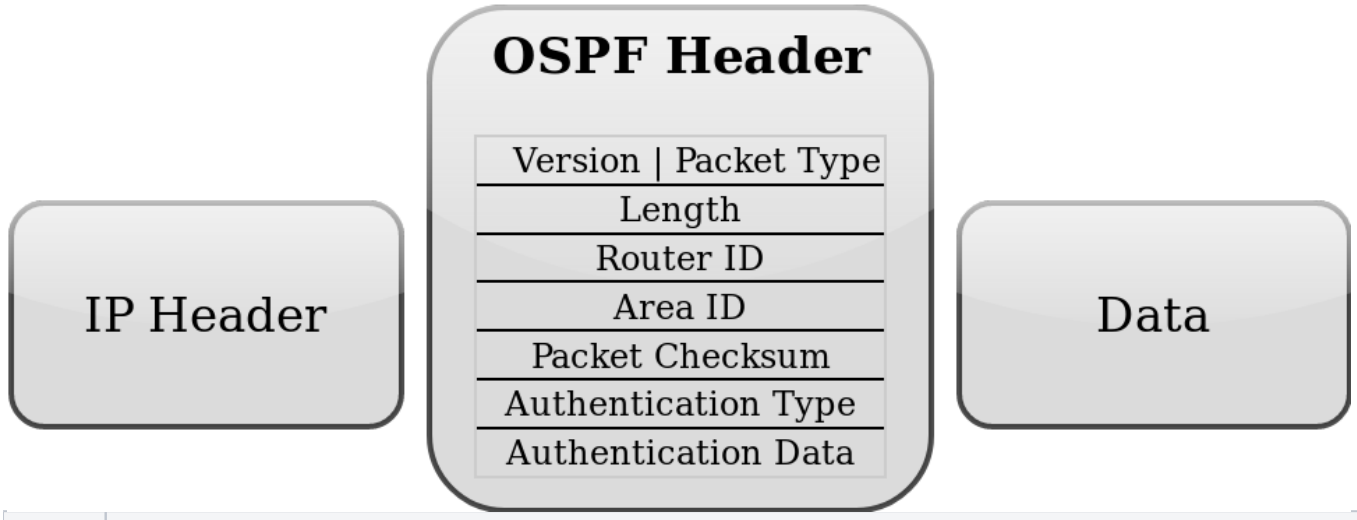
- Neighbors discovery
- Database Synchronization
- Routing calculation

Link-state routing protocols are distributing and replicating database that describes the routing topology. The link-state protocol's flooding algorithm ensures that each router has an identical link-state database and the routing table is calculated based on this database.

After all the steps above are completed link-state database on each neighbor contains full routing domain topology (how many other routers are in the network, how many interfaces routers have, what networks link between router connects, cost of each link, and so on).

Communication Between OSPF Routers

OSPF operates over the IP network layer using protocol number 89.
A destination IP address is set to the neighbor's IP address or to one of the OSPF multicast addresses AllSPFRouters (224.0.0.5) or AllDRRouters (224.0.0.6). The use of these addresses is described later in this article.
Every OSPF packet begins with a standard 24-byte header.



Field	Description
Packet type	There are several types of OSPF packets: Hello packet, Database Description (DD) packet, Link state request packet, Link State Update packet, and Link State Acknowledgement packet. All of these packets except the Hello packet are used in link-state database synchronization.
Router ID	one of the router's IP addresses unless configured manually
Area ID	Allows OSPF router to associate the packet to the proper OSPF area.
Checksum	Allows receiving router to determine if a packet was damaged in transit.
Authentication fields	These fields allow the receiving router to verify that the packet's contents were not modified and that packet really came from the OSPF router whose Router ID appears in the packet.

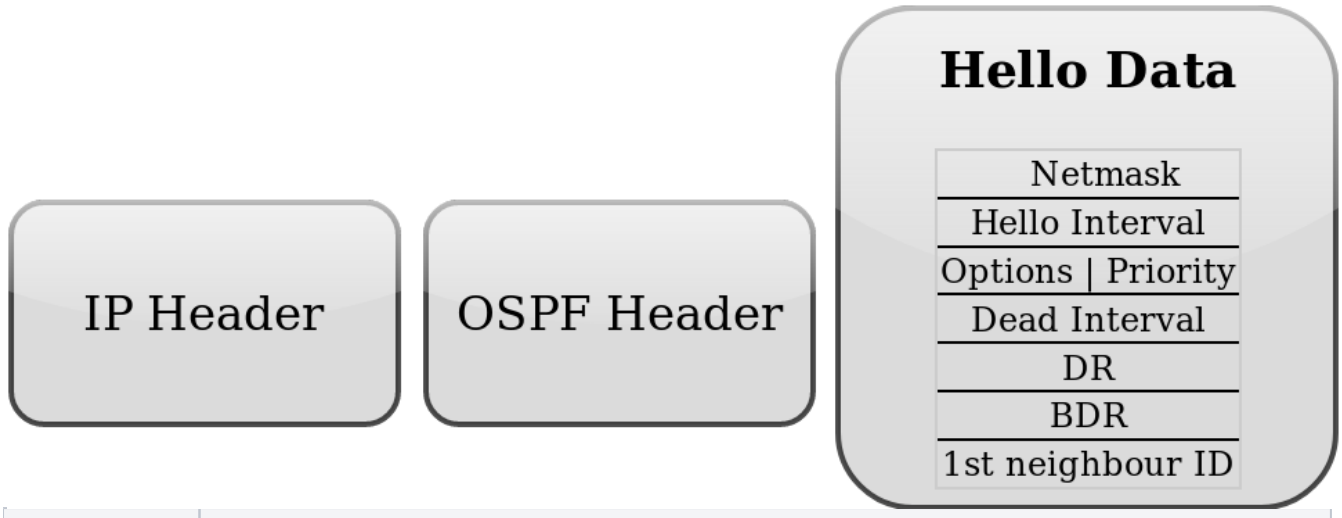
There are five different OSPF packet types used to ensure proper LSA flooding over the OSPF network.

- **Hello packet** - used to discover OSPF neighbors and build adjacencies.
- **Database Description (DD)** - check for Database synchronization between routers. Exchanged after adjacencies are built.
- **Link-State Request (LSR)** - used to request up-to-date pieces of the neighbor's database. Out-of-date parts of the routing database are determined after the DD exchange.
- **Link-State Update (LSU)** - carries a collection of specifically requested link-state records.
- **Link-State Acknowledgment (LSack)** - is used to acknowledge other packet types that way introducing reliable communication.

Neighbors Discovery

OSPF discovers potential neighbors by periodically sending Hello packets out of configured interfaces. By default *Hello packets* are sent out at 10-second intervals which can be changed by setting hello-interval in OSPF interface settings. The router learns the existence of a neighboring router when it receives the neighbor's Hello in return with matching parameters.

The transmission and reception of Hello packets also allow a router to detect the failure of the neighbor. If Hello packets are not received within dead-interval (which by default is 40 seconds) router starts to route packets around the failure. "Hello" protocol ensures that the neighboring routers agree on the Hello interval and Dead interval parameters, preventing situations when not in time received Hello packets mistakenly bring the link down.



Field	Description
network mask	The IP mask of the originating router's interface IP address.
hello interval	the period between Hello packets (default 10s)
options	OSPF options for neighbor information
router priority	an 8-bit value used to aid in the election of the DR and BDR. (Not set in p2p links)
router dead interval	time interval has to be received before considering the neighbor is down. (By default four times bigger than the Hello interval)
DR	the router-id of the current DR
BDR	the router-id of the current BDR
Neighbor router IDs	a list of router ids for all the originating router's neighbors

On each type of network segment Hello protocol works a little differently. It is clear that on point-to-point segments only one neighbor is possible and no additional actions are required. However, if more than one neighbor can be on the segment additional actions are taken to make OSPF functionality even more efficient.

Two routers do not become neighbors unless the following conditions are met.

- Two-way communication between routers is possible. Determined by flooding Hello packets.
- The interface should belong to the same area;
- The interface should belong to the same subnet and have the same network mask unless it has network-type configured as **point-to-point**;
- Routers should have the same authentication options, and have to exchange the same password (if any);
- Hello and Dead intervals should be the same in Hello packets;
- External routing and NSSA flags should be the same in Hello packets.

⚠ Network mask, Priority, DR, and BDR fields are used only when the neighbors are connected by a broadcast or NBMA network segment.

Discovery on Broadcast Subnets

The attached node to the broadcast subnet can send a single packet and that packet is received by all other attached nodes. This is very useful for auto-configuration and information replication. Another useful capability in broadcast subnets is multicast. This capability allows sending a single packet which will be received by nodes configured to receive multicast packets. OSPF is using this capability to find OSPF neighbors and detect bidirectional connectivity.

Consider the Ethernet network illustrated in the image below.

!!!!!!bilde!!!!!! OSPF Broadcast network

Each OSPF router joins the IP multicast group **AllSPFRouters** (224.0.0.5), then the router periodically multicasts its Hello packets to the IP address 224.0.0.5. All other routers that joined the same group will receive a multicasted Hello packet. In that way, OSPF routers maintain relationships with all other OSPF routers by sending a single packet instead of sending a separate packet to each neighbor on the segment.

This approach has several advantages:

Automatic neighbor discovery by multicasting or broadcasting Hello packets. Less bandwidth usage compared to other subnet types. On the broadcast segment, there are n*(n-1)/2 neighbor relations, but those relations are maintained by sending only n Hellos. If the broadcast has the multicast capability, then OSPF operates without disturbing non-OSPF nodes on the broadcast segment. If the multicast capability is not supported all routers will receive broadcasted Hello packets even if the node is not an OSPF router.

Discovery on NBMA Subnets

Non-broadcast multiaccess (NBMA) segments are similar to broadcast. Support more than two routers, the only difference is that NBMA does not support a data-link broadcast capability. Due to this limitation, OSPF neighbors must be discovered initially through configuration. On RouterOS static neighbor configuration is set in the `/routing ospf static-neighbor` menu. To reduce the amount of Hello traffic, most routers attached to the NBMA subnet should be assigned a Router Priority of 0 (set by default in RouterOS). Routers that are eligible to become Designated Routers should have priority values other than 0. It ensures that during the election of DR and BDR Hellos are sent only to eligible routers.

Discovery on PTMP Subnets

Point-to-MultiPoint treats the network as a collection of point-to-point links.

By design, PTMP networks should not have broadcast capabilities, which means that OSPF neighbors (the same way as for NBMA networks) must be discovered initially through configuration and all communication happens by sending unicast packets directly between neighbors. On RouterOS static neighbor configuration is set in the `/routing ospf static-neighbor` menu. Designated Routers and Backup Designated Routers are not elected on Point-to-multipoint subnets.

For PTMP networks that do support broadcast, a hybrid type named "ptmp-broadcast" can be used. This network type uses multicast Hellos to discover neighbors automatically and detect bidirectional communication between neighbors. After neighbor detection "ptmp-broacast" sends unicast packets directly to the discovered neighbors. This mode is compatible with the RouterOS v6 "ptmp" type.

Master-Slave Relation

Before database synchronization can begin, a hierarchy order of exchanging information must be established, which determines which router sends **Database Descriptor** (**DD**) packets first (**Master**). The master router is elected based on the **highest priority** and if priority is not set then the **router ID** will be used. Note that it is a router priority-based relation to arranging the exchanging data between neighbors which does not affect DR/BDR election (meaning that **DR** does not always have to be **Master**).

Database Synchronization

Link-state Database synchronization between OSPF routers is very important. Unsynchronized databases may lead to incorrectly calculated routing tables which could cause routing loops or black holes.

There are two types of database synchronizations:

- initial database synchronization
- reliable flooding.

When the connection between two neighbors first comes up, *initial database synchronization* will happen. OSPF is using explicit database download when neighbor connections first come up. This procedure is called **Database exchange**. Instead of sending the entire database, the OSPF router sends only its LSA headers in a sequence of OSPF **Database Description (DD)** packets. The router will send the next DD packet only when the previous packet is acknowledged. When an entire sequence of DD packets has been received, the router knows which LSAs it does not have and which LSAs are more recent. The router then sends **Link-State Request (LSR)** packets requesting desired LSAs, and the neighbor responds by flooding LSAs in **Link-State Update (LSU)** packets. After all the updates are received neighbors are said to be **fully adjacent**.

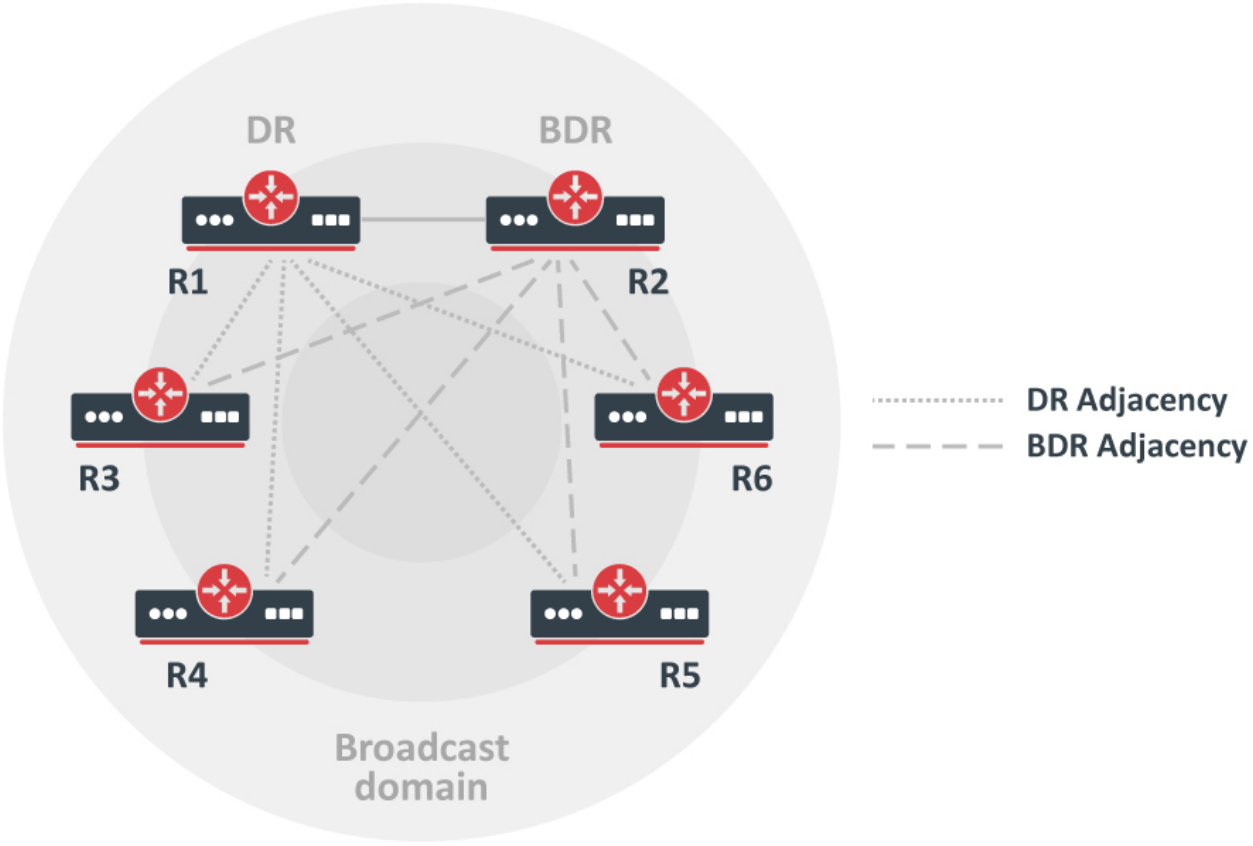
Reliable flooding is another database synchronization method. It is used when adjacencies are already established and the OSPF router wants to inform other routers about LSA changes. When the OSPF router receives such Link State Update, it installs a new LSA in the link-state database, sends an acknowledgment packet back to the sender, repackages LSA in new LSU, and sends it out to all interfaces except the one that received the LSA in the first place.

OSPF determines if LSAs are up to date by comparing sequence numbers. Sequence numbers start with 0×80000001, the larger the number, the more recent the LSA is. A sequence number is incremented each time the record is flooded and the neighbor receiving the update resets the Maximum age timer. LSAs are refreshed every 30 minutes, but without a refresh, LSA remains in the database for the maximum age of 60 minutes.

Databases are not always synchronized between all OSPF neighbors, OSPF decides whether databases need to be synchronized depending on the network segment, for example, on point-to-point links databases are always synchronized between routers, but on Ethernet networks databases are synchronized between certain neighbor pairs.

Synchronization on Broadcast Subnets

On the broadcast segment, there are n*(n-1)/2 neighbor relations, it will be a huge amount of Link State Updates and Acknowledgements sent over the subnet if the OSPF router will try to synchronize with each OSPF router on the subnet.



This problem is solved by electing one **Designated Router** and one **Backup Designated Router** for each broadcast subnet. All other routers are synchronizing and forming adjacencies only with those two elected routers. This approach reduces the number of adjacencies from n*(n-1)/2 to only 2n-3.

The image on the right illustrates adjacency formations on broadcast subnets. Routers R1 and R2 are Designated Routers and Backup Designated routers respectively. For example, if R3 wants to flood Link State Update (LSU) to both R1 and R2, a router sends LSU to the IP multicast address **AllDRouters** (224.0.0.6) and only DR and BDR listen to this multicast address. Then Designated Router sends LSU addressed to AllSPFRouters, updating the rest of the routers.

DR Election

DR and BDR routers are elected from data received in the Hello packet. The first OSPF router on a subnet is always elected as Designated Router, when a second router is added it becomes Backup Designated Router. When an existing DR or BDR fails new DR or BDR is elected to take into account configured router priority. The router with the highest priority becomes the new DR or BDR.

Being Designated Router or Backup Designated Router consumes additional resources. If Router Priority is set to 0, then the router is not participating in the election process. This is very useful if certain slower routers are not capable of being DR or BDR.

Synchronization on NBMA Subnets

Database synchronization on NBMA networks is similar to that on broadcast networks. DR and BDR are elected, databases initially are exchanged only with DR and BDR routers and flooding always goes through the DR. The only difference is that Link State Updates must be replicated and sent to each adjacent router separately.

Synchronization on PTMP Subnets

On PTMP subnets OSPF router becomes adjacent to all other routes with which it can communicate directly.

Understanding OSPF Areas

A distinctive feature of OSPF is the possibility to divide AS into multiple routing Areas which contain their own set of neighbors.
Imagine a large network with 300+ routers and multiple links between them. Whenever link flaps or some other topology change happens in the network, this change will be flooded to all OSPF devices in the network resulting in a quite heavy load on the network and even downtime since network convergence may take some time for such a large network.

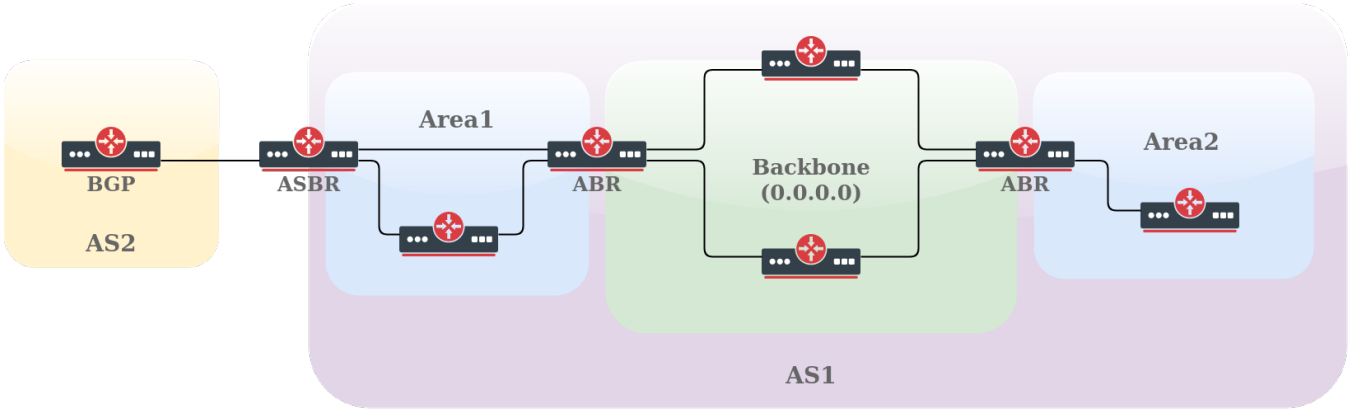
A large single-area network can produce serious issues:

- Each router recalculates the database every time whenever network topology change occurs, the process takes CPU resources.
- Each router holds an entire link-state database, which shows the topology of the entire network, it takes memory resources.
- A complete copy of the routing table and a number of routing table entries may be significantly greater than the number of networks, which can take even more memory resources.
- Updating large databases requires more bandwidth.

The introduction of areas allows for better resource management since topology change inside one area is not flooded to other areas in the network. The concept of areas enables simplicity in network administration as well as routing summarization between areas significantly reducing the database size that needs to be stored on each OSPF neighbor. This means that each area has its own link-state database and corresponding shortest-path tree.

The structure of an area is invisible to other areas. This isolation of knowledge makes the protocol more scalable if multiple areas are used; routing table calculation takes fewer CPU resources and routing traffic is reduced.

However, multi-area setups create additional complexity. It is not recommended to separate areas with fewer than 50 routers. The maximum number of routers in one area is mostly dependent on the CPU power you have for routing table calculation.



OSPF area has unique 32-bit identification (Area ID) and the area with an Area ID of 0.0.0.0 (called the Backbone area) is the main one where any other area should connect. Routers that connect to more than one area are called **ABR** (Area Border Routers), and their main responsibility is summarization and update suppression between connected areas. The router connecting to another routing domain is called **ASBR** (Autonomous System Boundary Router).

Each area has its own link-state database, consisting of router-LSAs and network-LSAs describing how all routers within that area are interconnected. Detailed knowledge of the area's topology is hidden from all other areas; router-LSAs and network-LSAs are not flooded beyond the area's borders. Area Border Routers (**ABR**s) leak addressing information from one area into another in OSPF summary-LSAs. This allows one to pick the best area border router when forwarding data to destinations from another area and is called **intra-area routing**.

Routing information exchange between areas is essentially a Distance Vector algorithm and to prevent algorithm convergence problems, such as counting to infinity, all areas are required to attach directly to the **backbone area** making a simple hub-and-spoke topology. The area-ID of the backbone area is always 0.0.0.0 and can not be changed.

RouterOS area configuration is done in the `/routing/ospf/area` menu. For example, a configuration of an ABR router with multiple attached areas, one Stub area, and one default area:

```
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=stub_area area-id=1.1.1.1 instance=v2inst type=stub
add name=another_area area-id=2.2.2.2 instance=v2inst type=default
```

OSPF can have 5 types of areas. Each area type defines what type of LSAs the area supports:

- standard/default - OSPF packets can normally be transmitted in this area, it supports types 1,2,3,4 and 5 LSAs

- backbone - as already mentioned this is the main area where any other area connects. It is basically the same as the standard area but identified with ID 0.0.0.0
- stub - this area does not accept any external routes
- totally stubby - a variation of the stub area
- not-so-stubby (NSSA) - a variation of the stub area

LSA Types

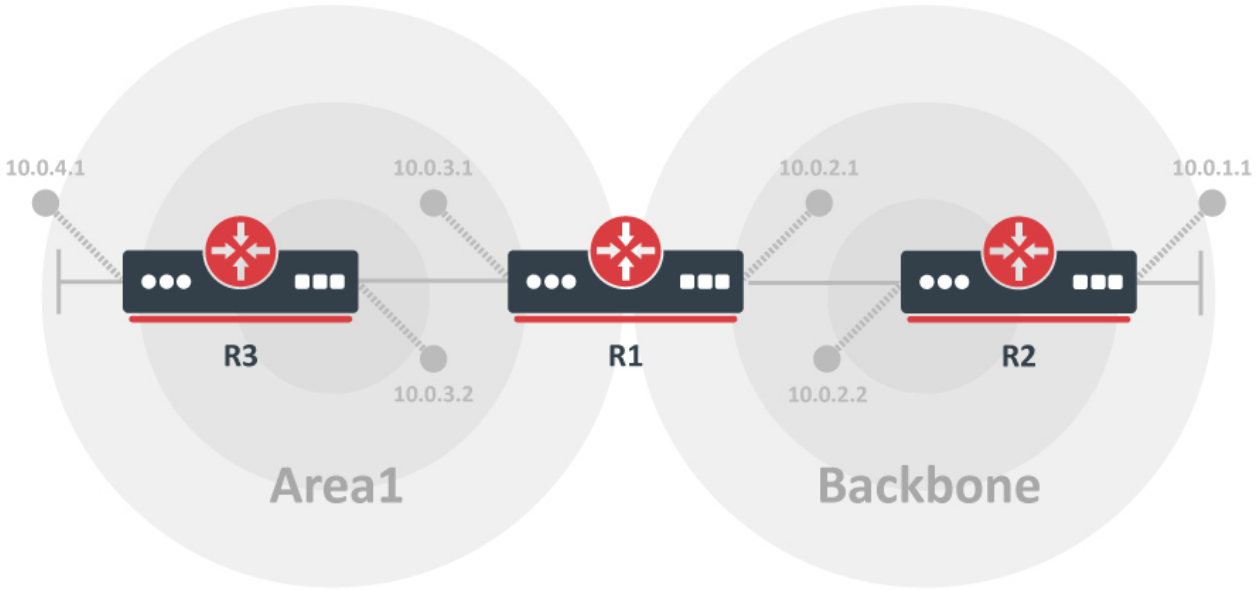
Before we continue a detailed look at each area type, let's get familiar with LSA types:

- ***type 1*** - (Router LSA) Sent by routers within the Area, including the list of directly attached links. Do not cross the ABR or ASBR.
- ***type 2*** - (Network LSA) Generated for every "transit network" within an area. A transit network has at least two directly attached OSPF routers. Ethernet is an example of a Transit Network. A Type 2 LSA lists each of the attached routers that make up the transit network and is generated by the DR.
- ***type 3*** - (Summary LSA) The ABR sends Type 3 Summary LSAs. A Type 3 LSA advertises any networks owned by an area to the rest of the areas in the OSPF AS. By default, OSPF advertises Type 3 LSAs for every subnet defined in the originating area, which can cause flooding problems, so it´s a good idea to use a manual summarization at the ABR.
- ***type 4*** - (ASBR-Summary LSA) It announces the ASBR address, it shows “where” the ASBR is located, announcing its address instead of its routing table.
- ***type 5*** - (External LSA) Announces the Routes learned through the ASBR, are flooded to all areas except Stub areas. This LSA divides into two sub-types: ***external type 1*** and ***external type 2***.
- ***type 6*** - (Group Membership LSA) This was defined for Multicast extensions to OSPF and is not used by RouterOS.
- ***type 7*** - type 7 LSAs are used to tell the ABRs about these external routes imported into the NSSA area. Area Border Router then translates these LSAs to **type 5** external LSAs and floods as normal to the rest of the OSPF network
- ***type 8*** - External Attributes LSA (OSPFv2) / link-local LSA (OSPFv3)
- ***type 9*** - Link-Local Scope Opaque (OSPFv2) / Intra Area Prefix LSA (OSPFv3). LSA of this type is not flooded beyond the local (sub)network.
- ***type 10*** - Area Local Scope Opaque. LSA of this type is not flooded beyond the scope of its associated area.
- ***type 11*** - Opaque LSA which is flooded throughout the AS (scope is the same as **type 5**). It is not flooded in stub areas and NSSAs.

> ⚠ If we do not have any ASBR, there are no LSA Types 4 and 5 in the network.

Standard Area

This area supports 1, 2, 3, 4, and 5 LSAs.



Simple multi-area network using default area. In this example, all networks from area1 are flooded to the backbone and all networks from the backbone are flooded to area1.
R1:

```
/ip address add address=10.0.3.1/24 interface=ether1
/ip address add address=10.0.2.1/24 interface=ether2
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.1
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=area1 area-id=1.1.1.1 type=default instance=v2inst
/routing ospf interface-template
add networks=10.0.2.0/24 area=backbone_v2
add networks=10.0.3.0/24 area=area1
```

R2:

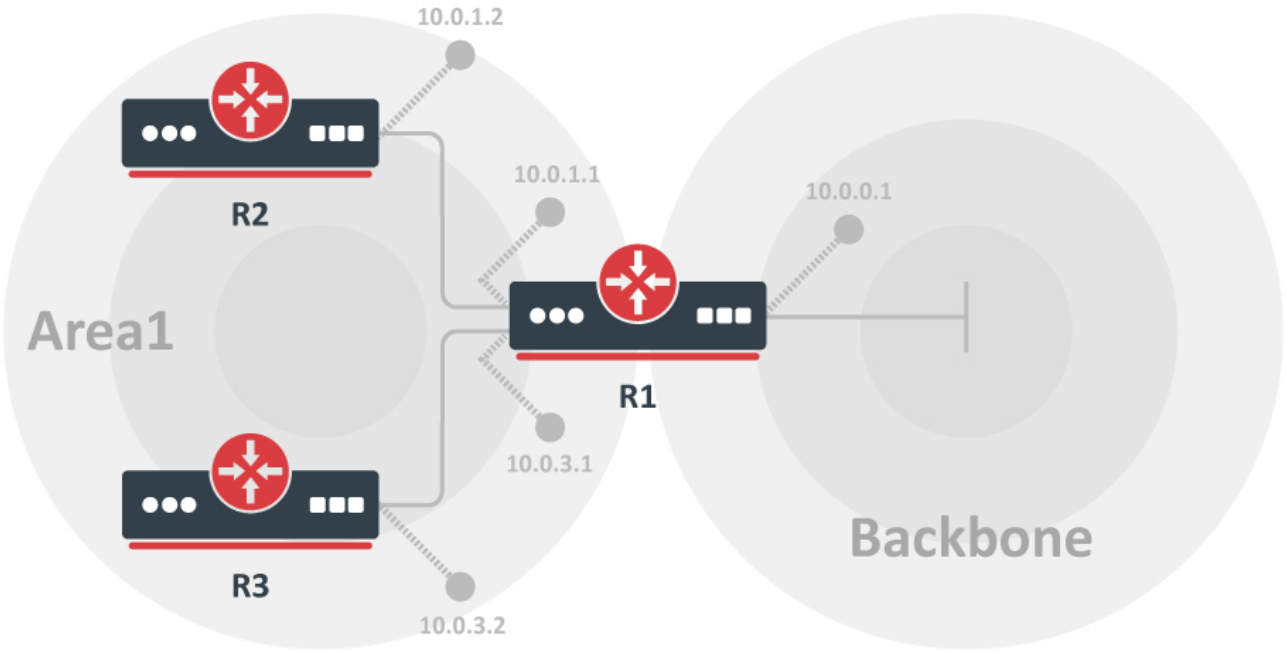
```
/ip address add address=10.0.1.1/24 interface=ether2
/ip address add address=10.0.2.2/24 interface=ether1
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.2
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0
/routing ospf interface-template
add networks=10.0.2.0/24 area=backbone_v2
add networks=10.0.1.0/24 area=backbone_v2
```

R3:

```
/ip address add address=10.0.3.2/24 interface=ether2
/ip address add address=10.0.4.1/24 interface=ether1
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.3
/routing ospf area
add name=area1 area-id=1.1.1.1 type=stub instance=v2inst
/routing ospf interface-template
add networks=10.0.3.0/24 area=area1
add networks=10.0.4.0/24 area=area1
```

Stub Area

The main purpose of stub areas is to keep such areas from carrying external routes. Routing from these areas to the outside world is based on a default route. A stub area reduces the database size inside an area and reduces the memory requirements of routers in the area.



The stub area has a few restrictions, ASBR routers cannot be internal to the area, stub area cannot be used as a transit area for virtual links. The restrictions are made because the stub area is mainly configured not to carry external routes.

This area supports 1, 2, and 3 LSAs.

Let's consider the example above. Area1 is configured as a stub area meaning that routers R2 and R3 will not receive any routing information from the backbone area except the default route.

R1:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.1
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=area1 area-id=1.1.1.1 type=stub instance=v2inst

/routing ospf interface-template
add networks=10.0.0.0/24 area=backbone_v2
add networks=10.0.1.0/24 area=area1
add networks=10.0.3.0/24 area=area1
```

R2:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.2
/routing ospf area
add name=area1 area-id=1.1.1.1 type=stub instance=v2inst
/routing ospf interface-template
add networks=10.0.1.0/24 area=area1
```

R3:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.3
/routing ospf area
```

```
add name=area1 area-id=1.1.1.1 type=stub instance=v2inst
/routing ospf interface-template
add networks=10.0.3.0/24 area=area1
```

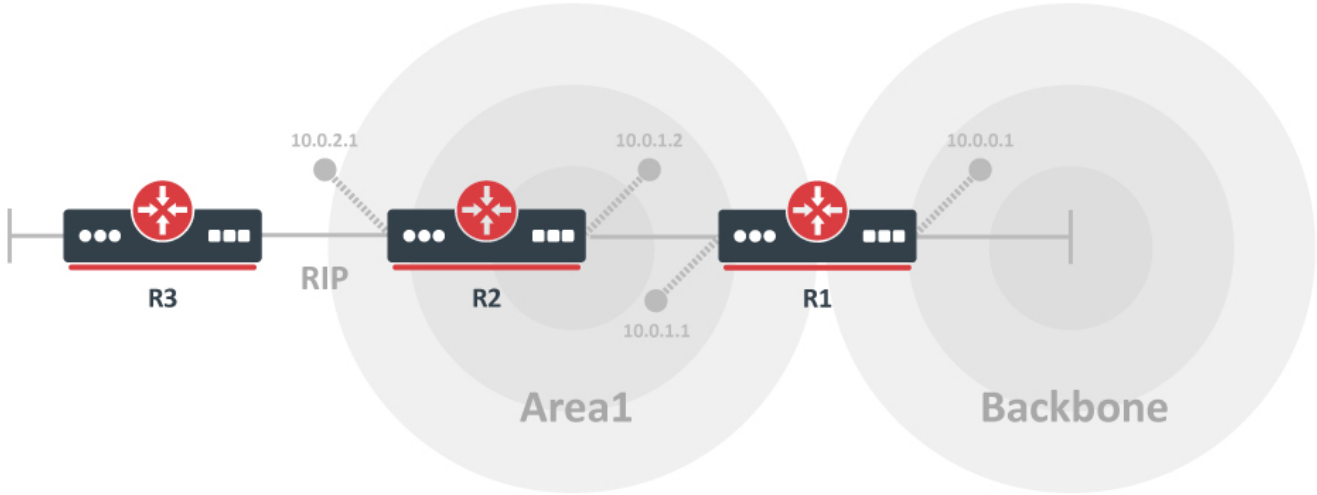
Totally Stubby Area

Totally stubby area is an extension of the stub area. A totally stubby area blocks external routes and summarized (inter-area) routes from going into the area. Only intra-area routes are injected into the area. Totally stubby area is configured as a stub area with an additional `no-summaries` flag. This area supports Type 1, Type 2 LSAs, and Type 3 LSAs with default routes.

```
/routing ospf area
add name=totally_stubby_area area-id=1.1.1.1 instance=v2inst type=stub no-summaries
```

NSSA

Not-so-stubby area (NSSA) is useful when it is required to inject external routes, but injection of type 5 LSA routes is not required.



The illustration shows two areas (backbone and area1) and RIP connection to the router located in "area1". We need "area1" to be configured as a stub area, but it is also required to inject external RIP routes in the backbone. Area1 should be configured as NSSA in this case.

The configuration example does not cover RIP configuration.

R1:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.1
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=area1 area-id=1.1.1.1 type=nssa instance=v2inst
/routing ospf interface-template
add networks=10.0.0.0/24 area=backbone_v2
add networks=10.0.1.0/24 area=area1
```

R2:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.0.0.2
/routing ospf area
add name=area1 area-id=1.1.1.1 type=nssa instance=v2inst
/routing ospf interface-template
add networks=10.0.1.0/24 area=area1
```

⚠ Virtual links cannot be used over NSSA areas.

External Routing Information and Default Route

On the edge of an OSPF routing domain, you can find routers called **AS boundary routers (ASBRs)** that run one of the other routing protocols. The job of those routers is to import routing information learned from other routing protocols into the OSPF routing domain. External routes can be imported at two separate levels depending on the metric type.

- type1 - OSPF metric is the sum of the internal OSPF cost and the external route cost
- type2 - OSPF metric is equal only to the external route cost.

External routes can be imported via the instance `redistribute` parameter. The example below will pick and redistribute all static and RIP routes:

```
/routing ospf instance
add name=v2inst version=2 router-id=1.2.3.4 redistribute=static,rip
```

Redistribution of default route is a special case where the `originate-default` the parameter should be used:

```
/routing ospf instance
set v2inst originate-default=if-installed
```

Since redistribution is controlled by "`originate-default`" and "`redistribute`" parameter, it introduces some corner-cases for default route filtering.

- if `redistribute` is enabled, then pick all routes matching redistribute parameters
- If `originate-default=never`, a default route will be rejected
- run selected routes through `out-select-chain` (if configured)
- run selected routes through `out-filter-chain` (if configured)
- if `originate-default` is set to `always` or `if-installed`:
    - OSPF creates a fake default route without attributes;
    - runs this route through `out-filter-chain` where attributes can be applied, but action is ignored (always accept);

For a complete list of redistribution values, see the reference manual.

Route Summarisation

Route summarization is a consolidation of multiple routes into one single advertisement. It is normally done at the area boundaries (Area Border Routers).

It is better to summarise in the direction of the backbone. That way the backbone receives all the aggregated routes and injects them into other areas already summarized. There are two types of summarization: inter-area and external route summarization.

Inter-area route summarization works on area boundaries (ABRs), it does not apply to external routes injected into OSPF via redistribution. By default, ABR creates a summary LSA for each route in a specific area and advertises it in adjacent areas.

Using ranges allows for creating only one summary LSA for multiple routes and sending only a single advertisement into adjacent areas, or suppressing advertisements altogether.

If a range is configured with the '`advertise`' parameter, a single summary LSA is advertised for each range if there are any routes under the range in the specific area. Otherwise (when '`advertise`' parameter disabled) no summary LSAs are created and advertised outside area boundaries at all.

Inter-area route summarization can be configured from the OSPF area range menu.

Let's consider that we have two areas backbone and area1, area1 has several /24 routes from the 10.0.0.0/16 range and there is no need to flood the backbone area with each /24 subnet if it can be summarized. On the router connecting area1 with the backbone we can set up the area range:

```
/routing ospf area range
add prefix=10.0.0.0/16 area=area1 advertise=yes cost=10
```

> ⚠ For an active range (i.e. one that has at least one OSPF route from the specified area falling under it), a route with the type 'blackhole' is created and installed in the routing table.

External route summarization can be achieved using routing filters. Let's consider the same example as above except that area1 has redistributed /24 routes from other protocols. To send a single summarised LSA, a blackhole route must be added and an appropriate routing filter to accept only summarised route:

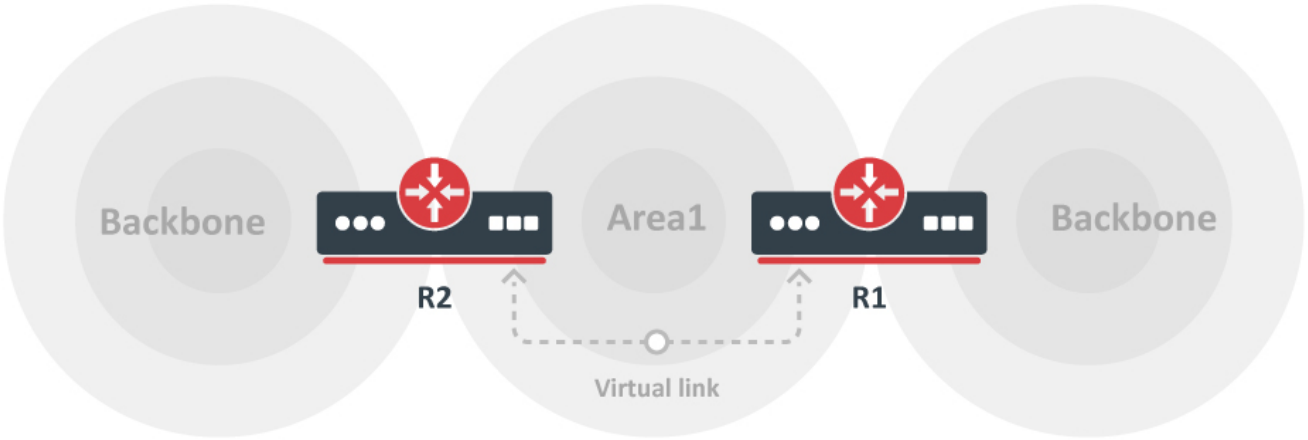
```
/ip route add dst-address=10.0.0.0/16 blackhole
/routing ospf instance
set v2inst out-filter-chain=ospf_out
/routing filter rule
add chain=ospf_out rule="if (dst == 10.0.0.0/16) {accept} else {reject}"
```

Virtual Link

As it was mentioned previously all OSPF areas have to be attached to the backbone area, but sometimes the physical connection is not possible. To overcome this, areas can be attached logically by using **virtual links**.

There are two common scenarios when virtual links can be used:

- to glue together the fragmented backbone area
- to connect remote are without direct connection to the backbone



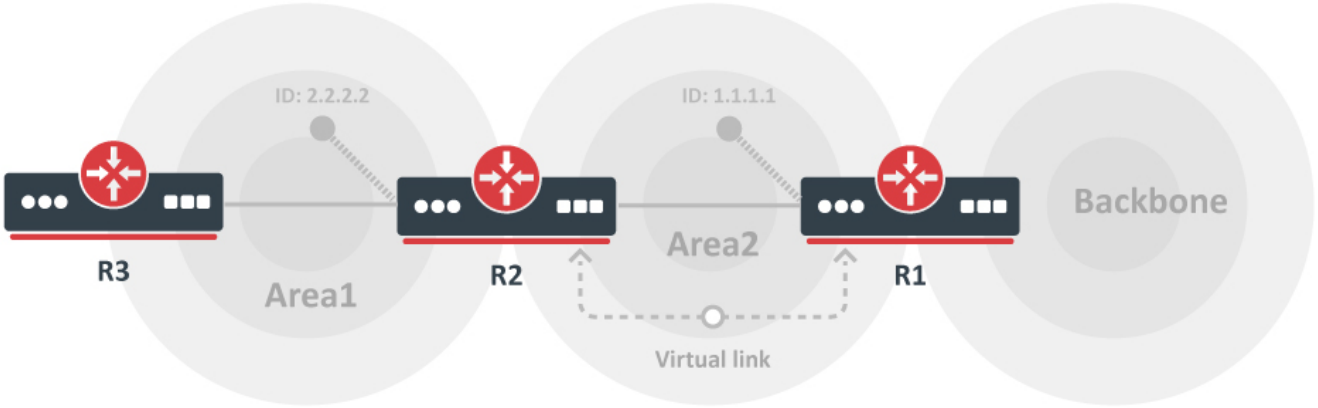
Partitioned Backbone

OSPF allows to linking of discontinuous parts of the backbone area using virtual links. This might be required when two separate OSPF networks are merged into one large network. Virtual links can be configured between separate ABRs that touch the backbone area from each side and have a common area.

The additional area could be created to become a transit area when a common area does not exist, it is illustrated in the image above.

Virtual Links are not required for non-backbone areas when they get partitioned. OSPF does not actively attempt to repair area partitions, each component simply becomes a separate area, when an area becomes partitioned. The backbone performs routing between the new areas. Some destinations are reachable via **intra-area** routing, the area partition requires **inter-area** routing.

However, to maintain full routing after the partition, an address range has not to be split across multiple components of the area partition.



No physical connection to a backbone

The area may not have a physical connection to the backbone, a virtual link is used to provide a logical path to the backbone of the disconnected area. A link has to be established between two ABRs that have a common area with one ABR connected to the backbone.

We can see that both R1 and R2 routers are ABRs and R1 is connected to the backbone area. Area2 will be used as a **transit area** and R1 is the **entry point** into the backbone area. A virtual link has to be configured on both routers.

Virtual link configuration is added in OSPF interface templates. If we take the example setup from the "no physical connection" illustration, then the virtual link configuration would look like this:

R1:

```
/routing ospf interface-template
add vlink-transit-area=area2 area=backbone_v2 type=virtual-link vlink-neighbor-id=2.2.2.2
```

R2:

```
/routing ospf interface-template
add vlink-transit-area=area2 area=backbone_v2 type=virtual-link vlink-neighbor-id=1.1.1.1
```

Property Reference

Instance

Sub-menu: `/routing/ospf/instance`

Property	Description
domain-id (*Hex | Address*)	MPLS-related parameter. Identifies the OSPF domain of the instance. This value is attached to OSPF routes redistributed in BGP as VPNv4 routes as BGP extended community attribute and used when BGP VPNv4 routes are redistributed back to OSPF to determine whether to generate inter-area or AS-external LSA for that route. By default Null domain-id is used, as described in RFC 4577.
domain-tag (*integer [0..4294967295]*)	if set, then used in route redistribution (as route-tag in all external LSAs generated by this router), and in route calculation (all external LSAs having this route tag are ignored). Needed for interoperability with older Cisco systems. By default not set.
in-filter (*string*)	name of the routing filter chain used for incoming prefixes

mpls-te-address (*string*)	the area used for MPLS traffic engineering. TE Opaque LSAs are generated in this area. No more than one OSPF instance can have mpls-te-area configured.
mpls-te-area (*string*)	the area used for MPLS traffic engineering. TE Opaque LSAs are generated in this area. No more than one OSPF instance can have mpls-te-area configured.
originate-default (*always | if-installed | never*; Default: **never**)	Specifies default route (0.0.0.0/0) distribution method.
out-filter-chain (*name*)	name of the routing filter chain used for outgoing prefixes filtering. Output operates only with "external" routes.
out-filter-select (*name*)	name of the routing filter select chain, used for output selection. Output operates only with "external" routes.
redistribute (*bgp, connected,copy,dhcp, fantasy,modem,ospf, rip,static,vpn*;)	Enable redistribution of specific route types.
router-id (*IP | name*; Default: **main**)	OSPF Router ID. Can be set explicitly as an IP address, or as the name of the router-id instance.
version (*2 | 3;* Default: **2**)	OSPF version this instance will be running (v2 for IPv4, v3 for IPv6).
vrf (*name of a routing table*; Default: **main**)	the VRF table this OSPF instance operates on
use-dn (*yes | no*)	Forces to use or ignore DN bit. Useful in some CE PE scenarios to inject intra-area routes into VRF. If a parameter is unset then the DN bit is used according to RFC. Available since v6rc12.

Notes

OSPF protocol supports two types of metrics:

- type1 - OSPF metric is the sum of the internal OSPF cost and the external route cost
- type2 - OSPF metric is equal only to the external route cost.

⚠ Type 1 external paths are always preferred over type 2 external paths. When all paths are type 2 external paths, the paths with the smallest advertised type 2 metric are always preferred. (RFC2328)

Area

Sub-menu: `/routing/ospf/area`

Property	Description
area-id (*IP address*; Default: **0.0.0.0**)	OSPF area identifier. If the router has networks in more than one area, then an area with area-id=0.0.0.0 (the backbone) must always be present. The backbone always contains all area border routers. The backbone is responsible for distributing routing information between non-backbone areas. The backbone must be contiguous, i.e. there must be no disconnected segments. However, area border routers do not need to be physically connected to the backbone - connection to it may be simulated using a virtual link.
default-cost (*integer*; unset)	Default cost of injected LSAs into the area. If the value is not set, then stub area type-3 default LSA will not be originated.
instance (*name*; mandatory)	Name of the OSPF instance this area belongs to.
no-summaries	Flag parameter, if set then the area will not flood summary LSAs in the stub area.

()	
name (*string*)	the name of the area
nssa-translate (*yes | no | candidate*)	The parameter indicates which ABR will be used as a translator from type7 to type5 LSA. Applicable only if area type is NSSA • yes - the router will be always used as a translator • no - the router will never be used as a translator • candidate - OSPF elects one of the candidate routers to be a translator
type (*default | nssa | stub*; Default: **default**)	The area type. Read more on the area types in the OSPF case studies.

Area Range

Sub-menu: `/routing/ospf/area/range`

Property	Description
advertise (*yes | no*; Default: yes)	Whether to create a summary LSA and advertise it to the adjacent areas.
area (*name*; mandatory)	the OSPF area associated with this range
cost (*integer [0..4294967295]*)	the cost of the summary LSA this range will create default - use the largest cost of all routes used (i.e. routes that fall within this range)
prefix (*IP prefix*; mandatory)	the network prefix of this range

Interface

Sub-menu: `/routing/ospf/interface`

Read-only matched interface menu

Interface Templates

Sub-menu: `/routing/ospf/interface-template`

The interface template defines common network and interface matches and what parameters to assign to a matched interface.

Matchers

Property	Description
interfaces (*name*)	Interfaces to match. Accepts specific interface names or the name of the interface list.
network (*IP prefix*)	the network prefix associated with the area. OSPF will be enabled on all interfaces that have at least one address falling within this range. Note that the network prefix of the address is used for this check (i.e. not the local address). For point-to-point interfaces, this means the address of the remote endpoint.

Assigned Parameters

Property	Description

area (*name*; mandatory)	The OSPF area to which the matching interface will be associated.
auth (*simple | md5 | sha1 | sha256 | sha384 | sha512*)	Specifies authentication method for OSPF protocol messages. • simple - plain text authentication • md5 - keyed Message Digest 5 authentication • sha - HMAC-SHA authentication RFC5709 If the parameter is unset, then authentication is not used.
auth-id (*integer*)	The key id is used to calculate message digest (used when MD5 or SHA authentication is enabled). The value should match all OSPF routers from the same region.
authentication-key (*string*)	The authentication key to be used, should match on all the neighbors of the network segment.
comment(*string*)	
cost(*integer [0..65535]*)	Interface cost expressed as link state metric.
dead-interval (*time*; Default: **40s**)	Specifies the interval after which a neighbor is declared dead. This interval is advertised in hello packets. This value must be the same for all routers on a specific network, otherwise, adjacency between them will not form
disabled(*yes | no*)	
hello-interval (*time*; Default: **10s**)	The interval between **HELLO** packets that the router sends out this interface. The smaller this interval is, the faster topological changes will be detected, the tradeoff is more OSPF protocol traffic. This value must be the same for all the routers on a specific network, otherwise, adjacency between them will not form.
instance-id (*integer [0..255]*; Default: **0**)	
passive ()	If enabled, then do not send or receive OSPF traffic on the matching interfaces
prefix-list (name)	Name of the address list containing networks that should be advertised to the v3 interface.
priority (*integer: 0..255*; Default: **128**)	Router's priority. Used to determine the designated router in a broadcast network. The router with the highest priority value takes precedence. Priority value 0 means the router is not eligible to become a designated or backup designated router at all. ROS v7 default value is 128 (defined in RFC), and the default value in ROS v6 was 1, keep this in mind when if you had strict priorities set for DR/BDR election.
retransmit-interval (*time*; Default: **5s**)	Time interval the lost link state advertisement will be resent. When a router sends a link state advertisement (LSA) to its neighbor, the LSA is kept until the acknowledgment is received. If the acknowledgment was not received in time (see transmit-delay), the router will try to retransmit the LSA.
transmit-delay (*time*; Default: **1s**)	Link-state transmit delay is the estimated time it takes to transmit a link-state update packet on the interface.
type (*broadcast | nbma | ptp | ptmp | ptp-unnumbered | virtual-link*; Default: **broadcast**)	the OSPF network type on this interface. Note that if interface configuration does not exist, the default network type is 'point-to-point' on PtP interfaces and 'broadcast' on all other interfaces. • broadcast - network type suitable for Ethernet and other multicast capable link layers. Elects designated router • nbma - Non-Broadcast Multiple Access. Protocol packets are sent to each neighbor's unicast address. Requires manual configuration of neighbors. Elects designated router • ptp - suitable for networks that consist only of two nodes. Do not elect designated router • ptmp - Point-to-Multipoint. Easier to configure than NBMA because it requires no manual configuration of a neighbor. Do not elect a designated router. This is the most robust network type and as such suitable for wireless networks, if 'broadcast' mode does not work well enough for them • ptp-unnumbered - works the same as ptp, except that the remote neighbor does not have an associated IP address to a specific PTP interface. For example, in case an IP unnumbered is used on Cisco devices. • virtual-link - for virtual link setups.
vlink-neighbor-id (*IP*)	Specifies the **router-id** of the neighbor which should be connected over the virtual link.
vlink-transit-area (*name*)	A non-backbone area the two routers have in common over which the virtual link will be established. Virtual links can

	not be established through stub areas.

Lsa

Sub-menu: `/routing/ospf/lsa`

Read-only list of all the LSAs currently in the LSA database.

Property	Description
age (*integer*)	How long ago (in seconds) the last update occurred
area (*string*)	The area this LSA belongs to.
body (*string*)	
checksum (*string*)	LSA checksum
dynamic (*yes* | *no*)	
flushing (*yes* | *no*)	
id (*IP*)	LSA record ID
instance (*string*)	The instance name this LSA belongs to.
link (*string*)	
link-instance-id (*IP*)	
originator (*IP*)	An originator of the LSA record.
self-originated (*yes* | *no*)	Whether LSA originated from the router itself.
sequence (*string*)	A number of times the LSA for a link has been updated.
type (*string*)	
wraparound (*string*)	

Neighbors

Sub-menu: `/routing/ospf/neighbor`

Read-only list of currently active OSPF neighbors.

Property	Description
address (*IP*)	An IP address of the OSPF neighbor router
adjacency (*time*)	Elapsed time since adjacency was formed
area (*string*)	
bdr (*string*)	An IP address of the Backup Designated Router
comment (*string*)	
db-summaries (*integer*)	
dr (*IP*)	An IP address of the Designated Router
dynamic (*yes* | *no*)	
inactive (*yes* | *no*)	
instance (*string*)	

ls-requests (*integer*)	
ls-retransmits (*integer*)	
priority (*integer*)	Priority configured on the neighbor
router-id (*IP*)	neighbor router's **RouterID**
state (*down | attempt | init | 2-way | ExStart | Exchange | Loading | full*)	• **Down** - No Hello packets have been received from a neighbor. • **Attempt** - Applies only to NBMA clouds. The state indicates that no recent information was received from a neighbor. • **Init** - Hello packet received from the neighbor, but bidirectional communication is not established (Its own RouterID is not listed in the Hello packet). • **2-way** - This state indicates that bi-directional communication is established. DR and BDR elections occur during this state, routers build adjacencies based on whether the router is DR or BDR, and the link is point-to-point or a virtual link. • **ExStart** - Routers try to establish the initial sequence number that is used for the packet information exchange. The router with a higher ID becomes the master and starts the exchange. • **Exchange** - Routers exchange database description (DD) packets. • **Loading** - In this state actual link state information is exchanged. Link State Request packets are sent to neighbors to request any new LSAs that were found during the Exchange state. • **Full** - Adjacency is complete, and neighbor routers are fully adjacent. LSA information is synchronized between adjacent routers. Routers achieve the full state with their DR and BDR only, an exception is P2P links.
state-changes (*integer*)	Total count of OSPF state changes since neighbor identification

Static Neighbour configuration

Sub-menu: `/routing/ospf/static-neighbor`

Static configuration of the OSPF neighbors. Required for non-broadcast multi-access networks.

Property	Description
address (*IP%iface*; mandatory)	The unicast IP address and an interface, that can be used to reach the IP of the neighbor. For example, `address=1.2.3.4%ether1` indicates that a neighbor with IP *1.2.3.4* is reachable on the *ether1* interface.
area (*name*; mandatory)	Name of the area the neighbor belongs to.
comment (*string*)	
disabled (*yes | no*)	
instance-id (*integer [0..255]*; Default: 0)	
poll-interval (*time*; Default: **2m**)	How often to send hello messages to the neighbors which are in a "down" state (i.e. there is no traffic from them)

RIP

Summary

MikroTik RouterOS implements RIP version 2 (RFC 2453). Version 1 (RFC 1058) is not supported.

RIP enables routers in an autonomous system to exchange routing information. It always uses the best path (the path with the fewest number of hops (i.e. routers)) available.

General

Sub-menu: `/routing rip instance`

Property	Description
name	name of the instance
vrf (Default: **main**)	which VRF to use
afi (*ipv4* / *ipv6*; Default:)	specifies which afi to use.
in-filter-chain (Default:)	input filter chain
out-filter-chain (Default:)	output filter chain
out-filter-select (Default:)	output filter select rule chain
redistribute (*bgp, bgp-mpls-vpn, connected, dhcp, fantasy, modem, ospf, rip, static, vpn*; Default:)	which routes to redistribute
originate-default (Default:)	whether to originate default route
routing-table (Default: main)	in which routing table the routes will be added
route-timeout (Default:)	route timeout
route-gc-timeout (Default:)	
update-interval (*time*; Default:)	specifies time interval after which the route is considered invalid

Note: The maximum metric of RIP route is 15. Metric higher than 15 is considered 'infinity' and routes with such metric are considered unreachable. Thus RIP cannot be used on networks with more than 15 hops between any two routers, and using redistribute metrics larger that 1 further reduces this maximum hop count.

Interface

Sub-menu: `/routing rip interface-template`

Property	Description
name	name of the instance
instance	which VRF to use
interfaces	specifies which afi to use.
source-addresses	input filter chain
cost (Default:)	output filter chain
split-horizon (*no| yes*)	
poison-reverse (*no| yes*)	
mode (*passive| strict*)	
key-chain (*name*)	name of key-chain
password	password

Sub-menu: `/routing rip interface`

Read-only properties:

Property	Description
instance (*name*)	name of the instance
address (*address%interface*)	IP address and interface name

Neighbor

Sub-menu: `/routing rip neighbor`

This submenu is used to define a neighboring routers to exchange routing information with. Normally there is no need to add the neighbors, if multicasting is working properly within the network. If there are problems with exchanging routing information, neighbor routers can be added to the list. It will force the router to exchange the routing information with the neighbor using regular unicast packets.

Read-only properties:

Property	Description
address (*IP address*)	IP address of neighboring router
routes	amount of routes
packets-total	amount of all packets
packets-bad	amount of bad packets
entries-bad	amount of bad entries
last-update (*time*)	time from last update

Sub-menu: `/routing rip static-neighbor`

Property	Description
instance (name)	name of used instance
address (*IP address*)	IP address of neighboring router

Keys

Sub-menu: `/routing rip keys`

MD5 authentication key chains.

Property	Description
chain (*string*; Default: **""**)	chain name to place this key in.
key (*string*; Default: **""**)	authentication key. Maximal length 16 characters
key-id (*integer:0..255*; Default:)	key identifier. This number is included in MD5 authenticated RIP messages, and determines witch key to use to check authentication for a specific message.
valid-from (*date and time*; Default: today's date and time: 00:00:00)	key is valid from this date and time
valid-till (*date and time*; Default: today's date and time: 00:00:00)	key is valid until this date and time

BGP

Summary

The Border Gateway Protocol (BGP) allows setting up an inter-domain dynamic routing system that automatically updates routing tables of devices running BGP in case of network topology changes.

BGP is an inter-autonomous system routing protocol based on the distance-vector algorithm. It is used to exchange routing information across the Internet and is the only protocol that is designed to deal with a network of the Internet's size and the only protocol that can deal well with having multiple connections to unrelated routing domains.

BGP is designed to allow for sophisticated administrative routing policies to be implemented. It does not exchange information about network topology but rather reachability information. As such, BGP is better suited to inter-AS environments and special cases like informational feeds. If you just need to enable dynamic routing in your network, consider OSPF instead.

The feature is not supported on SMIPS devices (hAP lite, hAP lite TC, and hAP mini).

Standards and Technologies:

- RFC 4271 Border Gateway Protocol 4
- RFC 4456 BGP Route Reflection
- RFC 5065 Autonomous System Confederations for BGP
- RFC 1997 BGP Communities Attribute
- RFC 8092 BGP Large Communities
- RFC 4360, 5668 BGP Extended Communities
- RFC 2385 TCP MD5 Authentication for BGPv4
- RFC 5492 Capabilities Advertisement with BGP-4
- RFC 2918 Route Refresh Capability
- RFC 4760 Multiprotocol Extensions for BGP-4
- RFC 2545 Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing
- RFC 4893 BGP Support for Four-octet AS Number Space
- RFC 4364 BGP/MPLS IP Virtual Private Networks (VPNs)
- RFC 4761 Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signalling
- RFC 6286 - AS-wide Unique BGP Identifier for BGP-4
- RFC 4273 - SNMP peer table monitoring (OID 1.3.6.1.2.1.15.3.1)
- RFC 6793 - 4-byte ASN support and Aggregator attribute.

BGP Terminology

- AS - Autonomous System
- ASN - Autonomous System Number
- NLRI - Network Layer Reachability Information is what is being exchanged between BGP peers and represents how to reach the prefixes.
- IGP - Interior Gateway Protocol

- EGP - Exterior Gateway protocol
- RR - Route reflector is the router in the BGP network that reflects advertisements to all the neighbors, avoiding the requirement for full BGP mesh.
- Route server - is the BGP router that does not participate in traffic forwarding. Routes are typically not even installed in the FIB.
- loopback address - a /32 address configured on a dummy bridge interface, that can act as a loopback.

BGP Basics

BGP routers exchange reachability information by means of a transport protocol, which in the case of BGP is TCP (port 179). Upon forming a TCP connection these routers exchange **OPEN** messages to negotiate and confirm supported capabilities.

After agreeing on capabilities to use, the session is considered to be established and peers can start to exchange NLRIs via **UPDATE** messages. This information contains an indication of what sequence of full paths (BGP AS numbers) the route should take in order to reach the destination network (NLRI prefix).

The peers initially exchange their full routing tables and after the initial exchange, incremental updates are sent as the routing tables change. Thus, BGP does not require a periodic refresh of the entire BGP routing table.

BGP maintains the routing table version number which must be the same between any two given peers for the duration of the connection.

KEEPALIVE messages are sent periodically to ensure that the connection is up and running, if **KEEPALIVE** messages are not received within the **Hold Time** interval, the connection will be closed.

To respond to errors or special conditions, **NOTIFICATION** messages can be generated and sent to the remote peer, notification message type also indicates whether the connection should be immediately closed.

There can be two types of BGP connections:

- **iBGP** - is an "internal" link connecting peers from the same AS
- **eBGP** - is an "external" link connecting peers belonging to two different AS-es

A particular AS might have multiple BGP speakers and provide transit service to other AS-es. This implies that BGP speakers must maintain a consistent view of routing within the AS. A consistent view of the routes exterior to the AS is provided by having all BGP routers within the AS establish direct iBGP connections with each other (full mesh) or by utilizing a Router Reflector setup.

Using a set of administrative policies BGP speakers within the AS come to an agreement as to which entry/exit point to use for a particular destination. This information is communicated to the interior routers of the AS using the interior routing protocol (IGP), for example, OSPF, RIP, or static routing. In certain setups, iBGP can take the IGP protocol role as well.

For certain BGP attributes handling behavior may change depending on what type of connection is set up, for example, the LOCAL-PREF attribute is not advertised to eBGP peers.

RouterOS divides configuration and session monitoring into three menus:

- connection menu (`/routing/bgp/connection`)
- sessions menu(`/routing/bgp/session`)
- template menu (`/routing/bgp/template`)

Connection Menu

Let's look at a very basic eBGP configuration example assuming, that Router1 IP is 192.168.1.1, AS 65531 and Router2 IP 192.168.1.2, AS 65532:

```
#Router1
/routing/bgp/connection
add name=toR2 remote.address=192.168.1.2 as=65531 local.role=ebgp
```

```
#Router2
/routing/bgp/connection
add name=toR1 remote.address=192.168.1.1 as=65532 local.role=ebgp
```

The BGP connection menu defines BGP outgoing connections as well as acts as a template matcher for incoming BGP connections.

`local.role` parameter is used to indicate that this connection will be the eBGP. Also, notice that the connection does not require a remote AS number to be specified, RouterOS can determine a remote AS number dynamically from the first received **OPEN** message.

The parameter equivalent to other vendors and older RouterOS "update-source" is "`local.address`". In most cases, it can be left unconfigured, and let the router determine the address.

When a local address is not specified, BGP will try to guess the local address depending on the current setup:

- if the peer is iBGP
  - if loopback available
    - pick the highest loopback address
  - if loopback is not available
    - pick any highest IP address on the router
- if the peer is eBGP
  - if a remote peer's IP is not from a directly connected network:
    - and multihop is not set, then throw an error
    - and multihop is enabled:
      - if loopback available
        - pick the highest loopback address
      - if loopback is not available
        - pick any highest IP address on the router
  - if a remote peer's IP is from a directly connected network:
    - and multihop is not set:
      - pick the local routers IP address from that connected network
    - and multihop is set:
      - if loopback available
        - pick the highest loopback address
      - if loopback is not available
        - pick any highest IP address on the router

In addition to connection-specific parameters, template-specific parameters are also directly exposed in this menu, for easier configuration in simple scenarios (when templates are not necessary).

A list of all connection-specific parameters can be seen in the table below:

Property	Description
name (*string*; Default:)	Name of the BGP connection
connect (*yes* / *no*; Default: yes)	Whether to allow the router to initiate the connection.
listen (*yes* / *no*; Default: yes)	Whether to listen for incoming connections.
local - a group of parameters associated with the local side of the connection	
.address (*IPv4/6*; Default: ::)	Local connection address.
.port(*integer [0..65535]*; Default:179)	Local connection port.
.role(ebgp | ebgp-customer | ebgp-peer | ebgp-provider | ebgp-rs | ebgp-rs-client | ibgp | ibgp-rr | ibgp-rr-client; Default:)	BGP role, in most common scenarios it should be set to iBGP or eBGP. More information on BGP roles can be found in the corresponding RFC draft https://datatracker.ietf.org/doc/draft-ietf-idr-bgp-open-policy/?include_text=1)
.ttl (*integer [1..255]*; Default:)	Time To Live (hop limit) that will be recorded in sent TCP packets.
remote - a group of parameters associated with the remote side of the connection	
.address (*IPv4/6*; Default: ::)	Remote address used to connect and/or listen to.
.port(*integer [0..65535]*; Default:179)	Local connection port.
.as(*integer []*; Default:)	Remote AS number. If not specified BGP will determine remote AS automatically from the OPEN message.
.allowed-as()	List of remote AS numbers that are allowed to connect. Useful for dynamic peer configuration.
.ttl (*integer [1..255]*; Default:)	Acceptable minimum Time To Live, the hop limit for this TCP connection. For example, if 'ttl=255' then only single-hop neighbors will be able to establish the connection. This property only affects EBGP peers.

tcp-md5-key (*string*; Default:)	The key used to authenticate the connection with TCP MD5 signature as described in RFC 2385. If not specified, authentication is not used.
templates (*name[,name]*; Default: default)	List of the template names, to inherit parameters from. Useful for dynamic BGP peers.

Session Menu

To see the actual active sessions with selected template parameters and negotiated capabilities refer to the BGP sessions menu:

```
[admin@MikroTik] /routing/bgp/session> print
Flags: E - established
 0 E name="toR2"
     remote.address=192.168.1.2 .as=65532 .id=192.168.1.1 .refused-cap-opt=no
     .capabilities=mp,rr,as4 .afi=ip,ipv6 .messages=43346 .bytes=3635916 .eor=""
     local.address=192.168.1.1 .as=65531 .id=192.168.44.2 .capabilities=mp,rr,gr,as4 .messages=2
     .bytes=71 .eor=""
     output.procid=97 .keep-sent-attributes=no
     .last-notification=ffffffffffffffffffffffffffffffff0015030601
     input.procid=97 .limit-process-routes=500000 ebgp limit-exceeded
     hold-time=3m keepalive-time=1m uptime=4s70ms
```

This menu shows read-only cached BGP session information. It will show the current status of the session, flags, last received notification, and negotiated session parameters.

Even if the BGP session is not active anymore, the cache can still be stored for some time. Routes received from a particular session are removed only if the cache expires, this allows mitigating extensive routing table recalculations if the BGP session is flapping.

Also, in this menu is located a session-specific set of commands.

Command	Description
clear	Clear the session flags. For example, to be able to re-establish a session after the prefix limit is reached "limit-exceeded" flag must be cleared. It can be done by specifying "`flag`" parameter, which is able to take the following values: • input-last-notification • limit-exceeded • output-last-notification • refused-cap-opt • stopped
dump-saved-advertisements	Dump saved advertisements from specified BGP session in the *.pcap file. The filename to store data is set by "`save-to`" parameter.
refresh	Send route refresh to a specified BGP session. Is used to trigger re-sending all the routes from the remote peer. "`address-family`" parameters allow specifying for which address family to send route refresh.
resend	Resend prefixes to a specified BGP session. The command takes two parameters: • "`address-family`" - parameters allow specifying for which address family to resend prefixes. • "`save-to`" - the name of the pcap file where to dump resent messages, can be used for debugging purposes.
reset	Reset specified BGP session.
stop	Stop specified BGP session.

Template Menu

The template contains all BGP protocol-related configuration options. It can be used as a template for dynamic peers and to apply a similar configuration to a group of peers. Note that this is not the same as peer groups on Cisco devices, where the group is more than just a common configuration.

List of available template parameters:

Property	Description
add-path-out (*all /none*; Default:)	
address-families (*ip / ipv6 / l2vpn / l2vpn-cisco / vpnv4*; Default: **ip**)	List of address families about which this peer will exchange routing information. The remote peer must support (they usually do) BGP capabilities optional parameter to negotiate any other families than IP.
as (*integer [0..4294967295]*; Default:)	32-bit BGP autonomous system number. Value can be entered in AS-Plain and AS-Dot formats. The parameter is also used to set up the BGP confederation, in the following format: *`confederation_as/as`*. For example, if your AS is 34 and your confederation AS is 43, then as configuration should be `as=43/34`.
as-override (*yes / no*; Default: **no**)	If set, then all instances of the remote peer's AS number in the BGP AS-PATH attribute are replaced with the local AS number before sending a route update to that peer. Happens before routing filters and prepending.
cisco-vpls-nlri-len-fmt (*auto-bits / auto-bytes / bits / bytes*; Default:)	VPLS NLRI length format type. Used for compatibility with Cisco VPLS. [[Read more>>]].
cluster-id (*IP address*; Default:)	In case this instance is a route reflector: the cluster ID of the router reflector cluster to this instance belongs. This attribute helps to recognize routing updates that come from another route reflector in this cluster and avoid routing information looping. Note that normally there is only one route reflector in a cluster; in this case, 'cluster-id' does not need to be configured and BGP router ID is used instead
disabled (*yes / no*; Default: **no**)	Whether the template is disabled.
hold-time (*time[3s..1h] / infinity*; Default: **3m**)	Specifies the BGP Hold Time value to use when negotiating with peers. According to the BGP specification, if the router does not receive successive **KEEPALIVE** and/or **UPDATE** and/or **NOTIFICATION** messages within the period specified in the Hold Time field of the **OPEN** message, then the BGP connection to the peer will be closed. The minimal hold-time value of both peers will be actually used (note that the special value 0 or 'infinity' is lower than any other value) • infinity - never expire the connection and never send keepalive messages.
input - a group of parameters associated with BGP input	
.accept-comunities (*string* ; Default:)	A quick way to filter incoming updates with specific communities. It allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered". Changes to be applied required session refresh.
.accept-ext-communities(*string*; Default:)	A quick way to filter incoming updates with specific extended communities. It allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered". Changes to be applied required session refresh.
.accept-large-comunities (*string*; Default:)	A quick way to filter incoming updates with specific large communities. It allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered". Changes to be applied required session refresh.

.accept-nlri(*string*; Default:)	Name of the ipv4/6 address-list. A quick way to filter incoming updates with specific NLRIs. It allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered". Changes to be applied required session restart.
.accept-unknown(*string*; Default:)	A quick way to filter incoming updates with specific "unknown" attributes. It allows filtering incoming messages directly before they are even parsed and stored in memory, that way significantly reducing memory usage. Regular input filter chain can only reject prefixes which means that it will still eat memory and will be visible in /routing route table as "not active, filtered". Changes to be applied required session refresh.
.affinity(afi | alone | instance | main | remote-as | vrf; Default:)	Configure input multi-core processing. Read more in Routing Protocol Multi-core Support article. • **alone** - input and output of each session are processed in its own process, most likely the best option when there are a lot of cores and a lot of peers • **afi, instance, vrf, remote-as** - try to run input/output of new session in process with similar parameters • **main** - run input/output in the main process (could potentially increase performance on single-core even possibly on multi-core devices with a small amount of cores) • **input** - run output in the same process as input (can be set only for output affinity)
.allow-as (*integer [0..10]*; Default:)	Indicates how many times to allow your own AS number in AS-PATH, before discarding a prefix.
.filter (*name*; Default:)	Name of the routing filter chain to be used on input prefixes. This happens after NLRIs are processed. If the chain is not specified, then BGP by default accepts everything.
.**ignore-as-path-len** (*yes* / *no*; Default: **no**)	Whether to ignore the **AS_PATH** attribute in the BGP route selection algorithm
.limit-nlri-diversity (*integer*; Default:)	
.limit-process-routes-ipv4 (*integer*; Default:)	Try to limit the amount of received IPv4 routes to the specified number. This number does not represent the exact number of routes going to be installed in the routing table by the peer. BGP session "clear" command must be used to reset the flag if the limit is reached.
.limit-process-routes-ipv6 (*integer*; Default:)	Try to limit the amount of received IPv6 routes to the specified number. This number does not represent the exact number of routes going to be installed in the routing table by the peer. BGP session "clear" command must be used to reset the flag if the limit is reached.
keepalive-time (*time [1s.. 30m]*; Default:3m)	How long to keep the BGP session open after the last received "keepalive" message.
multihop (*yes* / *no*; Default: **no**)	Specifies whether the remote peer is more than one hop away. This option affects outgoing next-hop selection as described in RFC 4271 (for EBGP only, excluding EBGP peers local to the confederation). It also affects: • whether to accept connections from peers that are not in the same network (the remote address of the connection is used for this check); • whether to accept incoming routes with NEXT_HOP attribute that is not in the same network as the address used to establish the connection; • the target-scope of the routes installed from this peer; routes from multi-hop or IBGP peers resolve their next-hops through IGP routes by default.
name (*string*; Default:)	Name of the BGP template
nexthop-choice (*default* | *force-self* | *propagate*; Default: **default**)	Affects the outgoing **NEXT_HOP** attribute selection. Note that next-hops set in filters always take precedence. Also note that the next-hop is not changed on route reflection, except when it's set in the filter. • default - select the next-hop as described in RFC 4271 • force-self - always use a local address of the interface that is used to connect to the peer as the next-hop; • propagate - try to propagate further the next-hop received; i.e. if the route has BGP **NEXT_HOP** attribute, then use it as the next-hop, otherwise, fall back to the default case
output - a group of parameters associated with BGP output	

	.affinity(afi | alone | instance | main | remote-as | vrf; Default:)	Configure output multicore processing. Read more in Routing Protocol Multi-core Support article. • **alone** - input and output of each session is processed in its own process, the most likely best option when there are a lot of cores and a lot of peers • **afi, instance, vrf, remote-as** - try to run input/output of new session in process with similar parameters • **main** - run input/output in the main process (could potentially increase performance on single-core even possibly on multicore devices with small amount of cores) • **input** - run output in the same process as input (can be set only for output affinity)
	.default-originate (*always | if-installed | never*; Default: **never**)	Specifies default route (0.0.0.0/0) distribution method.
	default-prepend (*integer [0..255]*; Default:)	
	.filter-chain (*name*; Default:)	Name of the routing filter chain to be used on the output prefixes. If the chain is not specified, then BGP by default accepts everything.
	.filter-select (*name*; Default:)	Name of the routing select chain to be used for prefix selection. If not specified, then default selection is used.
	.keep-sent-attributes (*yes | no*; Default: no)	Store in memory sent prefix attributes, required for "`dump-saved-advertisements`" command to work. By default, sent-out prefixes are not stored to preserve the router's memory. An option should be enabled only for debugging purposes when necessary to see currently advertised prefixes.
	.network(*name*; Default:)	Name of the address list used to send local networks. The network is sent only if a matching IGP route exists in the routing table.
	.no-client-to-client-reflection (yes | no; Default:)	Disable client-to-client route reflection in Route Reflector setups.
	.no-early-cut (yes | no; Default:)	The early cut is the mechanism, to guess (based on default RFC behavior) what would happen with the sent NPLRI when received by the remote peer. If the algorithm determines that the NLRI is going to be dropped, a peer will not even try to send it. However such behavior may not be desired in specific scenarios, then then this option should be used to disable the early cut feature.
	redistribute (bgp*, connected, bgp-mpls-vpn , dhcp, fantasy, modem, ospf, rip, static, vpn*; Default:)	Enable redistribution of specified route types.
remove-private-as (*yes | no*; Default: **no**		If set, then the BGP **AS-PATH** attribute is removed before sending out route updates if the attribute contains only private AS numbers. The removal process happens before routing filters are applied and before the local, AS number is prepended to the AS path.
router-id (*IP | name*; Default: main)		BGP Router ID to be used. Use the ID from the /routing/router-id configuration by specifying the reference name, or set the ID directly by specifying IP. Equal router-ids are also used to group peers into one instance.
routing-table (*string*; Default:)		Name of the routing table, to install routes in.
save-to (*string*; Default:)		Filename to be used to save BGP protocol-specific packet content (Exported PDU) into pcap file. This method allows much simpler peer-specific packet capturing for debugging purposes. Pcap files in this format can also be loaded to create virtual BGP peers to recreate conditions that happened at the time when packet capture was running.
templates (*name[,name]*; Default:)		List of template names from which to inherit parameters. Useful feature, to easily configure groups with overlapping configuration options.
use-bfd (*yes | no*; Default: **no**)		Whether to use the BFD protocol for faster connection state detection.
vrf (*name*; Default: main)		Name of the VRF BGP connections operates on. By default always use the "main" routing table.

Best-Path Selection

BGP routers can receive multiple copies of the global routing table from multiple providers.

There should be some way to compare those multiple BGP routing tables and select the best route to the destination, the solution is the BGP Best Path Selection Algorithm.

The route is evaluated by the algorithm only if it is valid. In general, the route is considered valid if:

- NEXT_HOP of the route is valid and reachable
- AS_PATH received from external peers does not contain the local AS
- the route is not rejected by routing filters

For more information read nexthop selection and validation.

The best path algorithm also compares routes received only by a **single BGP instance**. Routes installed by different BGP instances are compared by the general algorithm, i.e. route distances are compared and the route with a lower distance is preferred.

If all the criteria are met, then the following actions take place:

1. The first path received is automatically considered the 'best path'. Any further received paths are compared to the first received to determine if the new path is better.
2. Prefer the path with the highest **WEIGHT**.
   This parameter is not a part of the BGP standard, it is invented to quickly locally select the best route. A parameter is local to the router (assigned with routing filters in the BGP input) and cannot be advertised. A route without assigned WEIGHT has a default value of 0.
3. Prefer the path with the highest **LOCAL_PREF**.
   This attribute is used only within an AS. A path without the LOCAL_PREF attribute has a value of 100 by default.
4. Prefer the path with the shortest **AS_PATH**. (skipped if `input.ignore-as-path-len` set to **yes**).
   Each AS_SET counts as 1, regardless of the set size. The AS_CONFED_SEQUENCE and AS_CONFED_SET are not included in the AS_PATH length.
5. Prefer the path that was locally originated via aggregate or BGP network
6. Prefer the path with the lowest **ORIGIN** type.

   Interior Gateway Protocol (IGP) is lower than Exterior Gateway Protocol (EGP), and EGP is lower than INCOMPLETE

   in other words **IGP < EGP < INCOMPLETE**
7. Prefer the path with the lowest **multi-exit discriminator** (MED).

   The router compares the MED attribute only for paths that have the same neighboring (leftmost) AS. Paths without explicit MED value are treated with MED of 0
8. Prefer **eBGP** over **iBGP** paths
9. Prefer the route that comes from the BGP router with the lowest **router ID**. If a route carries the **ORIGINATOR_ID** attribute, then the **ORIGINATOR_ID** is used instead of the router ID.
10. Prefer the route with the shortest **route reflection cluster list**. Routes without a cluster list are considered to have a cluster list of length 0.
11. Prefer the path that comes from the lowest neighbor address

Routing Filter Notes

On BGP output routing filters are executed before BGP itself is modifying attributes, for example, if `nexthop-choice` is set to `force-self`, then the gateway set in the routing filters will be overridden.

On BGP input routing filters are applied to the received attributes, which means that, for example, setting the gateway will work no matter what `nexhop-choice` value is set.

Running More than One Instance

As we already know for best path selection to work properly, BGP routes must be received from the same instance. But in certain scenarios it is necessary to run multiple BGP instances with their own separate tables.
BGP determines whether sessions belongs to the same instance by comparing configured local router IDs.

For example config below will run each peer in its own BGP instance

```
/routing/bgp/connection
add name=inst1_peer remote.address=192.168.1.1 as=1234 local.role=ebgp router-id=1.1.1.1
add name=inst2_peer remote.address=192.168.1.2 as=5678 local.role=ebgp router-id=2.2.2.2
```

When `router-id` is not specified BGP will pick the "default" ID from `/routing id`.

VPLS

Sub Menu: `/routing/bgp/vpls`

This menu lists all the configured BGP-based VPLS instances. These instances allow the router to advertise VPLS BGP NLRI and indicate that the router belongs to a specific customer VPLS network.

MP-BGP-based autodiscovery and signaling (RFC 4761).

Cisco VPLS BGP-based auto-discovery (draft-ietf-l2vpn-signaling-08).

Support for multiple import/export route target extended communities for BGP-based VPLS (both, RFC 4761 and draft-ietf-l2vpn-signaling-08).

Property	Description
bridge (*name*)	The name of the bridge where dynamically created VPLS interfaces should be added as ports.
bridge-cost (*integer [0..4294967295]*)	
bridge-horizon (*none | integer [0..4294967295]*)	If set to **none** bridge horizon will not be used.
cisco-id ()	Unique identifier. A parameter must be set for cisco-style VPLS signaling. In most cases this should not be used, any modern software supports RFC 4761 style signaling (see site-id parameter). Parameter is a merge of l2-router-id and RD, for example: 10.155.155.1&6550:123
comment (*string*)	Short description of the item.
disabled (*yes | no*)	Defines whether an item is ignored or used.
export-route-target (list of RTs)	The setting is used to tag BGP NLRI with one or more route targets which on the remote side is used by `import-route-targets`.
import-route-targets (*list of RTs*)	The setting is used to determine if BGP NLRI is related to a particular VPLS, by comparing route targets received from BGP NLRI.

local-pref (*integer[0..4294967295]*)	
name (*string*; Default:)	
pw-control-word (*default | disabled | enabled*)	Enables/disables Control Word usage. Read more in the VPLS Control Word article.
pw-l2mtu (*integer [32..65535]*)	Advertised pseudowire MTU value.
pw-type (*raw-ethernet | tagged-ethernet | vpls*)	The parameter is available starting from v5.16. It allows choosing advertised encapsulation in NLRI used only for comparison. It does not affect the functionality of the tunnel. `See pw-type usage example >>`
rd (*string*)	Specifies the value that gets attached to VPLS NLRI so that receiving routers can distinguish advertisements that may otherwise look the same. This implies that a unique route-distinguisher for every VPLS must be used. It is not necessary to use the same route distinguisher for some VPLS on all routers forming that VPLS as distinguisher is not used for determining if some BGP NLRI is related to a particular VPLS (Route Target attribute is used for this), but it is mandatory to have different distinguishers for different VPLSes. Accepts 3 types of formats. Read more>>
site-id (*integer [0..65535]*)	Unique site identifier. Each site must have a unique site-id. A parameter must be set for RFC 4761 style VPLS signaling.
vrf (*name*)	Name of the VRF table.

VPN

Sub Menu: `/routing/bgp/vpn`

Route Distinguisher

Route Distinguisher is a 64-bit integer, which is divided into three parts: type (always 2 bytes), administrator, and value.

Currently, there are three format types defined.

2bytes	2bytes	2bytes	2bytes
Type1	ASN	4byte value	
Type2	4-byte IP		value
Type3	4-byte ASN		value

Properties

disabled (*yes* | *no*)	

export - a group of parameters associated with the vpnv4 export	
.filter-chain (*name*)	The name of the routing filter chain that is used to filter prefixes before exporting.
.filter-select(*name*)	The name of the select filter chain that is used to select prefixes to be exported exporting.
.redistribute(*bgp* | *connected* | *dhcp* | *fantasy* | *modem* | *ospf* | *rip* | *static* | *vpn*)	Enable redistribution of specified route types from VRF to VPNv4.
.route-targets(*rt[,rt]*)	List of route targets added when exporting VPNv4 routes. The accepted RT format is similar to the one for Route Distinguishers.
import - a group of parameters associated with the vpnv4 import	
.filter-chain (*name*)	The name of the routing filter chain that is used to filter prefixes during import.
.route-targets(*rt[,rt]*)	List of route targets that will be used to import VPNv4 routes. The accepted RT format is similar to the one for Route Distinguishers.
.router-id(*name* | *ip*)	The router ID of the BGP instance that will be used for the BGP best path selection algorithm.
label-allocation-policy (*per-prefix* | *per-vrf*)	
name	
route-distinguisher (rd)	Helps to distinguish between overlapping routes from multiple VRFs. Should be unique per VRF. Accepts 3 types of formats. Read more>>
vrf (*name*)	Name of the VRF table that this VPN instance will use.

RPKI

Overview

RouterOS implements the Resource Public Key Infrastructure (RPKI) to Router Protocol defined in RFC8210. RTR is a very lightweight low memory footprint protocol, to reliably get prefix validation data from RPKI validators.
More information on RPKI and how to set up validators can be found in the APNIC blog:

https://blog.apnic.net/2022/04/06/how-to-installing-an-rpki-validator-2/

Basic Example

Let's consider that we have our own RTR server on our network with IP address 192.168.1.1:

```
/routing/bgp/rpki
add group=myRpkiGroup address=192.168.1.1 port=8282 refresh-interval=20
```

If the connection is established and a database from the validator is received, we can check prefix validity:

```
[admin@rack1_b33_CCR1036] /routing> rpki-check group=myRpkiGroup prfx=70.132.18.0/24 origin-as=16509
    valid
```

Now the cached database can be used by routing filters to accept/reject prefixes based on RPKI validity. At first, we need to set up a filter rule which defines against which RPKI group performs the verification. After that filters are ready to match the status from the RPKI database. Status can have one of three values:

- **valid** - database has a record and origin AS is valid.
- **invalid** - the database has a record and origin AS is invalid.
- **unknown** - database does not have information of prefix and origin AS.
- **unverified** - set when none of the RPKI sessions of the RPKI group has synced database. This value can be used to handle the total failure of the RPKI.

```
/routing/filter/rule
add chain=bgp_in rule="rpki-verify myRpkiGroup"
add chain=bgp_in rule="if (rpki invalid) { reject } else { accept }"
```

Configuration Options

Sub-Menu: `/routing/rpki`

Property	Description
address (*IPv4/6*) mandatory	Address of the RTR server
disabled(*yes* | *no*; Default: **no**)	Whether the item is ignored.
expire-interval (*integer [600..172800]*; Default: 7200)	Time interval [s] polled data is considered valid in the absence of a valid subsequent update from the validator.
group (*string*) mandatory	Name of the group a database is assigned to.
port (*integer [0..65535]*; Default: 323)	Connection port number
preference (*integer [0..4294967295]*; Default: 0)	If there are multiple RTR sources, the preference number indicates a more preferred one. A lesser number is preferred.

refresh-interval (*integer [1..86400]*; Default: 3600)	Time interval [s] to poll the newest data from the validator.
retry-interval (*integer [1..7200]*; Default: 600)	Time Interval [s] to retry after the failed data poll from the validator.
vrf(*name*; Default: main)	Name of the VRF table used to bind the connection to.

Route Selection and Filters

Route Filtering

Filter Syntax

The routing filter rule implements script-like syntax. The example below is a quick demonstration of a routing filter that matches prefixes with a prefix length greater than 24 from subnet 192.168.1.0/24 and increments the default distance by 1. If there is no match then subtract the default distance by one.

```
/routing filter rule
  add chain=myChain \
  rule="if (dst in 192.168.1.0/24 && dst-len>24) {set distance +1; accept} else {set distance -1; accept}"
```

Filter rule may consist of multiple matchers and actions:

```
if ( [matchers] ) { [actions] } else { [actions] }
```

There are two types of properties:

- only readable - ones that value is only readable and cannot be rewritten, these properties can be used only by matchers
- readable/writable - ones that value is readable and writeable, used by filter actions, and also can be used by matchers

Readable properties can be matched by other readable properties (for numeric properties only) or constant values using boolean operators.

```
[matchers]:
[prop readable] [bool operator] [prop readable]

[actions]:
[action] [prop writeable] [value]
```

The boolean operator is not used if there is only one possible operation.

Example without boolean operator:

```
if ( protocol connected ) { accept }
```

Example with boolean operator:

```
if ( bgp-med < 30 ) { accept }
```

With readable flag properties, matcher is used without specified boolean operator and without value

```
if ( ospf-dn ) { reject }
```

> ⚠ Be aware that the default action of the routing filter chain is "reject"

Only Readable Properties

Property	Type	Description
Numeric properties		
dst-len		Destination prefix length
bgp-path-len		The current length of the BGP AS-PATH
bgp-input-local-as		AS number of the local peer to which the prefix was sent
bgp-input-remote-as		AS number of the remote peer from which the prefix was received
bgp-output-local-as		AS number of the peer that will advertise the prefix
bgp-output-remote-as		AS number of the peer to which the prefix will be advertised
ospf-metric		Current OSPF metric
ospf-tag		Current OSPF tag
rip-metric		Current RIP metric
rip-tag		Current RIP tag
Flag properties		
active		indicates whether the route is active
bgp-atomic-aggregate		
bgp-communities-empty		indicates if the BGP Communities attribute is empty
bgp-ext-communities-empty		indicates if the BGP Extended Communities attribute is empty
bgp-large-communities-empty		indicates if the BGP Large Communities attribute is empty
bgp-network		Indicates if the prefix is originated from BGP networks

ospf-dn		Indicates if the OSPF route has DN bit set.
Prefix properties		
dst		Destination
ospf-fwd		Current OSPF forwarding address
bgp-input-local-addr		The IP address of the local peer to which the prefix was sent
bgp-input-remote-addr		The IP address of the remote peer from which the prefix was received
bgp-output-local-addr		The IP address of the peer that will advertise the prefix
bgp-output-remote-addr		The IP address of the peer to which the prefix will be advertised
Other Properties		
afi	`ipv4 | ipv6 | l2vpn | l2vpn-cisco | vpnv4 | vpnv6`	The address family of the route.
bgp-as-path	numeric_regexp	AS path matching, read more>>
bgp-as-path-slow-legacy	string_regexp	**Deprecated**. Extremely slow old-style AS path matching. This parameter should be used only as a temporary matcher while migrating from an old ROS v6 config. Read more>>
chain	chain_name	
ospf-type	`ext1 | ext2 | inter | intra | nssa1 | nssa2`	Type of the OSPF route: • ext1 - external (Type 5 LSA) with type1 metric • ext2 - external (Type 5 LSA) with type2 metric • inter - inter-area-route (Type 3 LSA) • intra - intra-area-route (Type 4 LSA) • nssa1 - Type 7 LSA with type1 metric • nssa2 - Type 7 LSA with type1 metric
protocol	bgp | connected | dhcp | fantasy | modem | ospf | rip | static | vpn	Protocol type from which the route was imported.
rpki	invalid | unknown | valid | unverified	RPKI validation status of the prefix
rtab	routing_table_name	Name of the routing table the route was imported from
vrf	vrf_name	Name of the VRF the route was imported from

Writeable Properties

Property	Type	Description
Numeric properties		
distance		route distance
scope		
scope-target		target scope
bgp-weight		BGP WEIGHT attribute
bgp-med		BGP MED attribute is local to the router. It is also used in the output of iBGP peers.

bgp-out-med		BGP MED attribute to be sent to a remote peer. Should be used in the output chain of eBGP peers.
bgp-local-pref		BGP LOCALPREF attribute
bgp-igp-metric		BGP IGP METRIC
bgp-path-peer-prepend		Prepend last received remote peers ASN. If the prefix is originated from the router, then this parameter will not do anything on the router's output, because ASN does not exist yet. If used as a matcher in BGP input, it is possible to filter prefixes exceeding a certain number of prepends. For example, if a remote peer prepends its ASN 5 times, but we want to allow max 4 times prepended ASN, then we can use: "`if (bgp-path-peer-prepend > 4) {reject}`" This parameter also overrides any prepends received from the remote peer, for example, if the remote peer prepended it's AS 3 times, we can remove this prepend by setting "`bgp-path-peer-prepend` 1" in BGP input
bgp-path-prepend		Prepend routers ASN, should be used in BGP output.
ospf-ext-metric		OSPF External route metric
ospf-ext-tag		OSPF external route tag
rip-ext-metric		RIP External route metric
rip-ext-tag		RIP External route tag
Flag properties		
ospf-ext-dn		DN bit for external OSPF routes
blackhole		
suppress-hw-offload		Whether to suppress L3 HW offloading
use-te-nexthop		
Other properties		
gw	ipv4/6 address	IPv4/IPv6 address or interface name. In the case of BGP output, a gateway can be adjusted in the following setups: • is BGP reflector • nexthop-choice is set to propagate • is not eBGP and nexthop-choice=force-self is not set.
gw-interface	interface_name	Interface part of the gateway. Should be used if it is required to attach a specific interface for next-hop, like (1.2.3.4% ether1)
gw-check	*`none|arp|icmp|bfd|bfd-mh`*	
pref-src	ipv4/6 address	
bgp-origin	*`igp|egp|incomplete`*	

ospf-ext-fwd	ipv4/6 address	Forwarding address of External OSPF route
ospf-ext-type	*type1|type2*	OSPF External route type
comment	string	
bgp-communities	inline_community_set | community_list_name	BGP Communities attribute is defined in RFC 1997. Each community is 32-bit in size.
bgp-ext-communities	inline_ext_community_set | ext_community_list_name	BGP Extended Communities attribute is defined in RFC 4360. RouterOS parses site-of-origin (prefixed with soo:) and route-target (prefixed with rt:) extended communities. For example, "set bgp-ext-communities rt:1111:2.3.4.5;". It is possible to set/match RAW extended communities value in 64-bit hex, for example, "set bgp-ext-community 0x.....;"
bgp-large-communities	inline_large_community_set | large_community_list_name	BGP Large Communities attribute is defined in RFC 8092. Suitable for use with all ASNs including 32-bit ASNs. Each community is 12-bytes in length and consists of 3 parts: "global_admin:locap_part_1:local_part_2".

Commands

Command	Params	Description
accept		accept matched prefix
reject		reject matched prefix, the prefix will be stored in the memory as "filtered" and will not be the candidate to be selected as the best path.
return		return to the parent chain
jump	*jump chain_name*	jump to a specified chain
unset	*unset prop_name*	used to unset the value of the following properties: `pref-src|bgp-med|bgp-out-med|bgp-local-pref`
append		append at the end of the list or string. Following property values can be appended: `bgp-communities, bgp-ext-communities, bgp-large-communities, comment`
filter		Inverse of the delete action (Delete everything except the specified values). Values of the following properties can be filtered: `bgp-communities, bgp-ext-communities, bgp-large-communities`
delete		Delete the value of the specified property. Values of the following properties can be deleted: `bgp-communities, bgp-ext-communities, bgp-large-communities`
set	*set prop_writeable value*	The command is used to set a new value to writeable properties. Value can be set from other readable properties of matching types. For numeric properties, it is possible to prefix the value with +/- which will increment or decrement the current property value by a given amount. For example, "`set pref-src +1`" will increment current pref-src by one, or extract value from other readable num property, "`set distance +ospf-ext-metric`"
rpki-verify	*rpki-verify rpki_group_name*	Enable RPKI verification in the current chain from the specified RPKI group.

Operators

Matcher Operators

Operator	Description	Example
&&	Logical AND operator	`if (dst in 192.168.0.0/16 && dst-len in 16-32) {reject;}`
||	Logical OR operator	
not	Logical NOT operator	`if ( not bgp-network) {reject; }`

Num Prop Operators

Operator	Description
in	return true if the value is in provided numeric range. Numeric range can be written in following formats: {int..int}, {int-int}
==	return true if numeric values are equal
!=	return true if numeric values are not equal
>	return true if the left numeric value is greater than the right numeric value
<	return true if the left numeric value is less than the right numeric value
>=	return true if the left numeric value is greater than or equal to the right numeric value
<=	return true if the left numeric value is less than or equal to the right numeric value

Prefix Operators

Operator	Description
in	Return true if the prefix is the subnet of the provided network. If an operator is used to match prefixes from the address list (e.g "`dst in list_name`"), then it will match only the exact prefix.
!=	Return true if the prefix is not equal to the provided value
==	Return true if the prefix is equal to the provided value

BGP Community Operators

Operator	Description	Example
equal	return true if provided communities are equal to the routes property value	
equal-list	return true if communities from provided community-list are equal to the route's property value	
any	returns true if the route's property value contains at least one of provided communities	
any-list	returns true if the route's property value contains at least one community from the provided list	
includes	returns true if the route's property value includes specified communities	
includes-list	returns true if the route's property value includes all communities from the specified communities-list	
subset	returns true if route community subset matches communities from the list	1:1,3:3 will match 1:1,2:2,3:3
subset-list	the same as "subset", but matches communities form the community list.	
any-regexp	the same as "any", but matched by regexp	
subset-regexp	the same as "subset", but matched by regexp	

String Operators

Operator	Description
find	Check if provided substring is part of the property value

regexp	Match string regexp of the property value

Deleting BGP Communities

Routing filters allow to clear BGP communities by using "delete" command. Delete command accepts several parameters based on the type of the community type:

- **communities**:
  - "wk" - will match and remove well known communities
  - "other" - will match and remove other communities that are not well known
  - "regexp" - regexp pattern to match communities that should be deleted
  - "<community-list name>" - deletes communities from specified community-list
- **ext-communities**:
  - "rt" - will match and remove **RouteTarget**
  - "soo" - will match and remove **Site-of-Origin**
  - "other" - will match and remove other ext communities that are not RT or SSO
  - "regexp" - regexp pattern to match ext communities that should be deleted
  - "<community-ext-list name>" - deletes communities from specified community-ext-list
- **large-communities**:
  - "all" - removes everything
  - "regexp" - regexp pattern to match large communities that should be deleted
  - "<community-large-list name>" - deletes large communities from specified community-large-list

It is possible to specify multiple community types, for example delete all SSOs, other type of ext communities and specific RTs from the community-ext list:

```
/routing/filter/community-ext-list
add list=myRTList communities="rt:1.1.1.1:222"
/routing/filter/rule
add chain=myChain rule="delete bgp-ext-communities sso,other,myRTList;"
```

AS-PATH Regexp Matching

AS Path is the sequence of autonomous system numbers (ASNs), for example AS Path 123 456 789 would indicate, that route originated from AS with the number 789, and to reach the destination, the packet would need to travel through two autonomous systems: 456 and 789. To apply specific routing policies administrator might want to match specific AS numbers or set of numbers in the AS Path (for example, reject prefixes that travel through AS 456), which can be achieved using regular expression (regexp).

There are two common ways how to operate with AS Path data:

- convert whole AS path to string and let regexp operate on the string (ROS v6 or Cisco style)
- let regexp operate on each entry in the AS path as a number (ROS v7, Juniper style)

Basically, the first method is performing the match per character, the second method is performing the match per whole AS number. As you would imagine the latter method is much faster and less resource-intensive than the string matching approach.

This change would require administrators to implement new Regex strategies. Old Regex patterns from RouterOS v6 cannot be directly copied/pasted as they will result either in syntax errors or unexpected results.

Let us take a very basic AS Path filter rule.

```
/routing/filter/rule
add chain=myChain rule="if (bgp-as-path .1234.) {accept}"
```

In ROS v7 this Regex pattern will match ASN 1234 anywhere in the middle of the AS-path, the same pattern in ROS v6 would match any AS path that contains ASN consisting of at least 6 characters and contains a string of "1234". Obviously, if we directly copy/paste the Regex pattern from one implementation to another it will lead to unexpected/dangerous results. An equivalent pattern in ROS v6 would look something like this: "._1234_.".

Let's take another example from ROS v6, say we have a pattern "1234[5-9]" what it does is it matches 12345 to 12349 anywhere in the string, which means that valid matches are AS-path "12345 3434", "11 9123467 22" and so on. If you enter the same pattern in ROS v7 it will match AS path containing exact ASN 1234 followed by ASN in a range from 5 to 9 (matching AS-paths would be "1234 7 111", "111 1234 5 222" etc., it will not match "12345 3434").

> Do not copy Regex patterns directly from ROS v6 or Cisco configurations, they are not directly compatible. It can lead to unexpected or even dangerous configurations in some scenarios.

Regex Testing Tool

RouterOS now has a built-in regex checking tool to simplify the hard life of the administrators. This tool supports also num-list so now exact regex can be tested against any as-path before applying it to the routing filters.

```
/routing/filter/num-list add list=test range=100-1500

/routing/filter/test-as-path-regexp regexp="[[:test:]]5678\$" as-path="1234,5678"
```

Supported Operators

Operator	Description	Example	Example Explained	Example Matches
^	Represents the beginning of the path	^1234	will match AS-path starting with ASN 1234	
$	Represents the end of the path	1234$	will match AS-path of origin ASN 1234	
*	Zero or more occurrences of the listed ASN	^1234*$	will match Null as-path or as-path where ASN 1234 may or may not appear multiple times	**Match:** 1234 1234 1234 1234 Null path **No Match:** 1234 5678
+	One or more occurrences of the listed ASN	1234+	will match AS-path where ASN 1234 appears at least once	**Match:** 1234 3 1234 6 **No match:** 12345 678
?	Zero or one occurrence of the listed ASN	^1234? 5678	will match AS-path that may or may not start with ASN 1234 appearing once.	**Match**: 5678 1234 5678 **No match:** 1234 1234 5678 12345 5678

.	One occurrence of any ASN	^.$	will match any AS-path with the length of one.	**Match:** 12345 45678 **No match:** 1234 5678
|	Match one of two ASNs on each side	^(1234|5678)	will match AS-path starting with ASN 1234 or 5678	**Match**: 1234 5678 1234 5678 **No Match:** 91011
[ ] [^]	Represents the set of AS numbers where one AS number from the list must match. Use ^ after opening the bracket to negate the set. It is also possible to reference the pre-defined num-lists from num-list with [[:numset_name:]]	^[1234 5678 1-100]	will match the AS-path that starts with 1234 or 5678 or from the range of 1 to 100	**Match:** 1234 99 5678 **No Match:** 101
()	Group of regexp terms to match	^(1234$|5678)	will match AS-path that starts and ends with 1234 or AS-path that starts with 5678	**Match:** 1234 5678 9999 **No Match:** 1234 5678

⚠ Repetition ranges {} are not supported.

Community and Num Lists

A list of commonly used numbers can be configured from the `/routing/filter/num-list` menu. These lists of numbers can be used in the filter rules to simplify the filter setup process.

In a similar manner, you are allowed to define also community, extended community, and large community lists. Community sets can be used for matching, appending, and setting.

For example match communities from the list and clear the attribute:

```
/routing/filter/community-list
add communities=111:222 list=myCommunityList

/routing/filter/rule
add chain=myChain rule="if (bgp-communities equal-list myCommunityList) {delete bgp-communities wk,other;
accept;}"
```

`/routing/filter/community-list`

Property	Description
comment (*string*; Default:)	
communities (*list of communities*; Default:)	List of communities expressed either as **well-known** name or in the following format: "**as:number**", where each section can be integer [0..65535]. Accepted well known names: accept-own graceful-shutdown no-advertise no-llgr route-filter-6 accept-own-nh internet no-export no-peer route-filter-xlate-4 blackhole llgr-stale local-as route-filter-4 route-filter-xlate-6
disabled (yes / *no*)	
name (*integer [string*; Default:)	Reference name.
regexp (string)	Regexp matcher to match communities. The community set with only the regexp parameter cannot be used to append communities.

`/routing/filter/community-ext-list`

Property	Description
comment (*string*; Default:)	
communities (*list of ext communities*; Default:)	List of extended communities expressed as **raw** integer value or in the typed format: "**type:value**", where type can be: • **rt** - route-target • **soo** - site of origin Value depends on the type, for more info on RT and SoO values ask google.
disabled (yes / *no*)	
name (*integer [string*; Default:)	Reference name.
regexp (string)	Regexp matcher to match communities. The community set with only the regexp parameter cannot be used to append communities.

`/routing/filter/community-large-list`

Property	Description
comment (*string*; Default:)	
communities (*list of large communities*; Default:)	List of large communities expressed in following format: "**admin:value1:value2**", where each section can be integer [0..4294967295].
disabled (yes / *no*)	
name (*integer [string*; Default:)	Reference name.
regexp (string)	Regexp matcher to match communities. The community set with only the regexp parameter cannot be used to append communities.

Route Selection

Route selection rules allow controlling how output routes are selected from available candidate routes. By default, (if no selection rules are set) output always picks the best route.

For example, if we look at the routing table below, we can see that there are 2 candidate routes and one best route. By default when BGP selects which route to send out, it will pick the active route.

```
[admin@4] /routing/route> print where dst-address=1.0.0.0/24
Flags: A - ACTIVE; b, y - COPY
Columns: DST-ADDRESS, GATEWAY, AFI, DISTANCE, SCOPE, TARGET-SCOPE, IMMEDIATE-GW
   DST-ADDRESS  GATEWAY         AFI  DISTANCE  SCOPE  TARGET-SCOPE  IMMEDIATE-GW
 b 1.0.0.0/24   10.155.101.217  ip4        19     40            30  10.155.109.254%ether1
Ab 1.0.0.0/24   10.155.101.232  ip4        20     40            30  10.155.109.254%ether1
 b 1.0.0.0/24   10.155.101.231  ip4        20     40            30  10.155.109.254%ether1
```

But there might be cases where you would want preference for other routes, not the active ones, and here come in-play selection rules.

Selection rules in RouterOS are configured from `/routing/filter/select-rule` menu.

Select rules can also call routing filters where routes get selected based on filter rules. For example, to mimic default output selection we can set up the following rule sets:

```
/routing filter rule
add chain=get_active rule="if (active) {accept}"

/routing filter select-rule
add chain=my_select_chain do-where=get_active
```

Property Reference

/routing/filter/chain

Dynamic list of filter rule chains that can be referenced in BGP/OSPF configuration.

Read-only properties:

Property	Description
dynamic (*yes* / *no*)	
inactive (yes / *no*)	
name (*string*)	

/routing/filter/select-chain

Dynamic list of filter select chains that can be referenced in BGP/OSPF configuration.

Read-only properties:

Property	Description
dynamic (*yes* / *no*)	
inactive (yes / *no*)	
name (*string*)	

Multicast

In This Section:

Group Management Protocol

Introduction

The Group Management Protocol allows any of the interfaces to become a receiver for the multicast stream. It allows testing the multicast routing and switching setups without using dedicated IGMP or MLD clients. The option is available since RouterOS v7.4 and it supports IGMP v1, v2, v3 and MLD v1, v2 protocols.

Interfaces are using IGMP v3 and MLD v2 by default. In case IGMP v1, v2 or MLD v1 queries are received, the interfaces will fall back to the appropriate version. Once Group Management Protocol is created on the interface, it will send an unsolicited membership report (join) packet and respond to query messages. If the configuration is removed or disabled, the interface will send a leave message.

Configuration options

This section describes the Group Management Protocol configuration options.

Sub-menu: `/routing gmp`

Property	Description
groups (*IPv4* | *IPv6*; Default:)	The multicast group address to be used by the interface, multiple group addresses are supported.
interfaces (*name*; Default:)	Name of the interface, multiple interfaces and interface lists are supported.
exclude (Default:)	When `exclude` is set, the interface expects to reject multicast data from the configured `sources`. When this option is not used, the interfaces will emit source specific join for the configured `sources`.
sources (*IPv4* | *IPv6*; Default:)	The source address list used by the interface, multiple source addresses are supported. This setting has an effect when IGMPv3 or MLDv2 protocols are active.

Examples

This example shows how to configure a simple multicast listener on the interface.

First, add an IP address on the interface:

```
/ip address
add address=192.168.10.10/24 interface=ether1 network=192.168.10.0
```

Then configure Group Management Protocol on the same interface:

```
/routing gmp
add groups=229.1.1.1 interfaces=ether1
```

It is now possible to check your multicast network to see if routers or switches have created the appropriate multicast forwarding entries and whether multicast data is being received on the interface (see the interface stats, or use a Packet Sniffer and Torch).

IGMP Proxy

Summary

Internet Group Management Protocol (IGMP) proxy can implement multicast routing. It is forwarding IGMP frames and is commonly used when there is no need for a more advanced protocol like PIM.

IGMP proxy features:

- The simplest way how to do multicast routing;
- Can be used in topologies where PIM-SM is not suitable for some reason;
- It takes slightly less resources than PIM-SM;
- Ease of configuration.

On the other hand, IGMP proxy is not well suited for complicated multicast routing setups. Compared to PIM-based solutions, IGMP proxy does not support more than one upstream interface and routing loops are not detected or avoided.

By default, IGMP proxy upstream interface will send IGMPv3 membership reports and it will detect what IGMP version the upstream device (e.g. multicast router) is using based on received queries. In case IGMPv1/v2 queries are received, the upstream port will fall back to the lower IGMP version. It will convert back to IGMPv3 when IGMPv1/v2 querier present timer (400s) expires. Downstream interfaces of IGMP proxy will only send IGMPv2 queries.

RouterOS v7 has IGMP proxy configuration available in the main **system** package. Older RouterOS versions need an additional **multicast** package installed in order to use IGMP proxy. See more details about Packages.

Configuration options

General IGMP proxy configuration.

Sub-menu: `/routing igmp-proxy`

Property	Description
query-interval (*time: 1s ..1h*; Default: **2m5s**)	How often to send out IGMP Query messages over downstream interfaces.
query-response-interval (*time: 1s..1h*; Default: **10s**)	How long to wait for responses to an IGMP Query message.
quick-leave	Specifies action on IGMP Leave message. If quick-leave is on, then an IGMP Leave message is sent upstream as soon as a leave message is received from the first client on the downstream interface. Use **yes** only in case there is only one subscriber behind the proxy.

Configure what interfaces will participate as IGMP proxy interfaces on the router. If an interface is not configured as an IGMP proxy interface, then all IGMP traffic received on it will be ignored.

Sub-menu: `/routing igmp-proxy interface`

Property	Description
alternative-subnets (*IP/Mask*; Default:)	By default, only packets from directly attached subnets are accepted. This parameter can be used to specify a list of alternative valid packet source subnets, both for data or IGMP packets. Has an effect only on the upstream interface. Should be used when the source of multicast data often is in a different IP network.

interface (*name*; Default: **all**)	Name of the interface.
threshold (*integer: 0..4294967295*; Default: **1**)	Minimal TTL. Packets received with a lower TTL value are ignored
upstream (*yes / no*; Default: **no**)	The interface is called "upstream" if it's in the direction of the root of the multicast tree. An IGMP forwarding router must have exactly one upstream interface configured. The upstream interface is used to send out IGMP membership requests.

It is possible to get detailed status information for each interface using the `print status` command.

```
[admin@MikroTik] /routing igmp-proxy interface print status
Flags: X - disabled, I - inactive, D - dynamic; U - upstream
 0  U interface=ether2 threshold=1 alternative-subnets="" upstream=yes source-ip-address=192.168.10.10 rx-
bytes=3018487500 rx-packets=2012325 tx-bytes=0 tx-packets=0

 1    interface=ether3 threshold=1 alternative-subnets="" upstream=no querier=yes source-ip-address=192.
168.20.10 rx-bytes=0 rx-packets=0 tx-bytes=2973486000 tx-packets=1982324

 2    interface=ether4 threshold=1 alternative-subnets="" upstream=no querier=yes source-ip-address=192.
168.30.10 rx-bytes=0 rx-packets=0 tx-bytes=152019000 tx-packets=101346
```

Property	Description
querier (*read-only; yes|no*)	Whether the interface is acting as an IGMP querier.
source-ip-address (*read-only; IP address*)	The detected source IP for the interface.
rx-bytes (*read-only; integer*)	The total amount of received multicast traffic on the interface.
rx-packet (*read-only; integer*)	The total amount of received multicast packets on the interface.
tx-bytes (*read-only; integer*)	The total amount of transmitted multicast traffic on the interface.
tx-packet (*read-only; integer*)	The total amount of transmitted multicast packets on the interface.

Multicast forwarding cache (MFC) status.

Sub-menu: `/routing igmp-proxy mfc`

Property	Description
active-downstream-interfaces (*read-only: name*)	The packet stream is going out of the router through this interface.
bytes (*read-only: integer*)	The total amount of received multicast traffic.
group (*read-only: IP address*)	IGMP group address.
packets (*read-only: integer*)	The total amount of received multicast packets.
source (*read-only: IP address*)	The multicast data originator address.
upstream-interface (*read-only: name*)	The packet stream is coming into the router through this interface.
wrong-packets (*read-only: integer*)	The total amount of received multicast packets that arrived on a wrong interface, for example, a multicast stream that is received on a downstream interface instead of an upstream interface.

RouterOS support static multicast forwarding rules for IGMP proxy. If a static rule is added, all dynamic rules for that group will be ignored. These rules will take effect only if IGMP-proxy interfaces are configured (upstream and downstream interfaces should be set) or these rules won't be active.

Property	Description
downstream-interfaces (*name*; Default:)	The received stream will be sent out to the listed interfaces only.

group (*read-only: IP address*)	The multicast group address this rule applies.
source (*read-only: IP address*)	The multicast data originator address.
upstream-interface (*read-only: name*)	The interface that is receiving stream data.

Examples

To forward all multicast data coming from the ether2 interface to the downstream bridge interface, where subscribers are connected, use the configuration below. Both interfaces should have an IP address.

```
/routing igmp-proxy interface
add interface=ether2 upstream=yes
add interface=bridge1

[admin@MikroTik] /routing igmp-proxy interface print
Flags: U - UPSTREAM
Columns: INTERFACE, THRESHOLD
#   INTERFACE  THRESHOLD
0 U ether2             1
1   bridge1            1
```

You may also need to configure `alternative-subnets` on the upstream interface in case the multicast sender address is in an IP subnet that is not directly reachable from the local router:

```
/routing igmp-proxy interface
set [find upstream=yes] alternative-subnets=192.168.50.0/24,192.168.60.0/24
```

To enable `quick-leave`, use the setting below:

```
/routing igmp-proxy
set quick-leave=yes
```

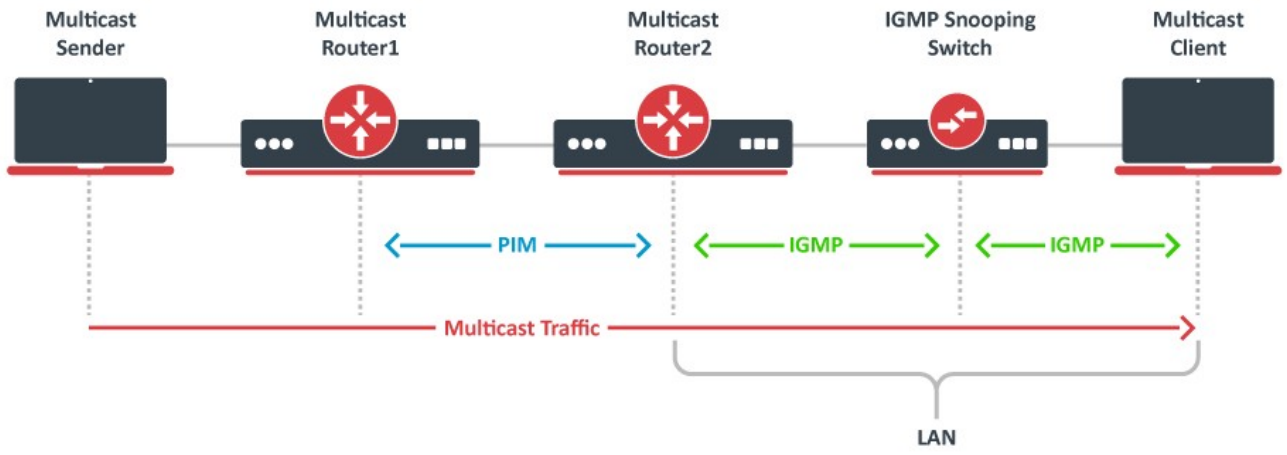
PIM-SM

Summary

IP Multicast is a technology that allows data to be efficiently shared with many recipients over the Internet. Senders transmit their data to a specific multicast IP address, and receivers indicate their interest in receiving data sent to that address. The network then takes care of delivering the data from senders to receivers.

If both the sender and receiver for a multicast group are on the same local network segment, routers are not required for the process. Communication can happen directly, and this can be enhanced with the use of IGMP snooping switches. However, if the sender and receiver are on different network segments, a multicast routing protocol must be used to establish the path for data transmission between them.

Protocol Independent Multicast - Sparse Mode (PIM-SM or PIM) enables RouterOS to support multicast streaming over the network area. PIM stands for Platform Independent Multicast, meaning it's not tied to any particular unicast routing. SM stands for Sparse-Mode, which means that specific control messages ensure that data is delivered only to network segments where there are receivers that want it. In addition to the routing protocols that manage data transmission between network segments, routers need a way to discover local receivers on their directly connected network segment. For IPv4, this is achieved through the Internet Group Management Protocol (IGMP), and Multicast Listener Discovery (MLD) for IPv6.

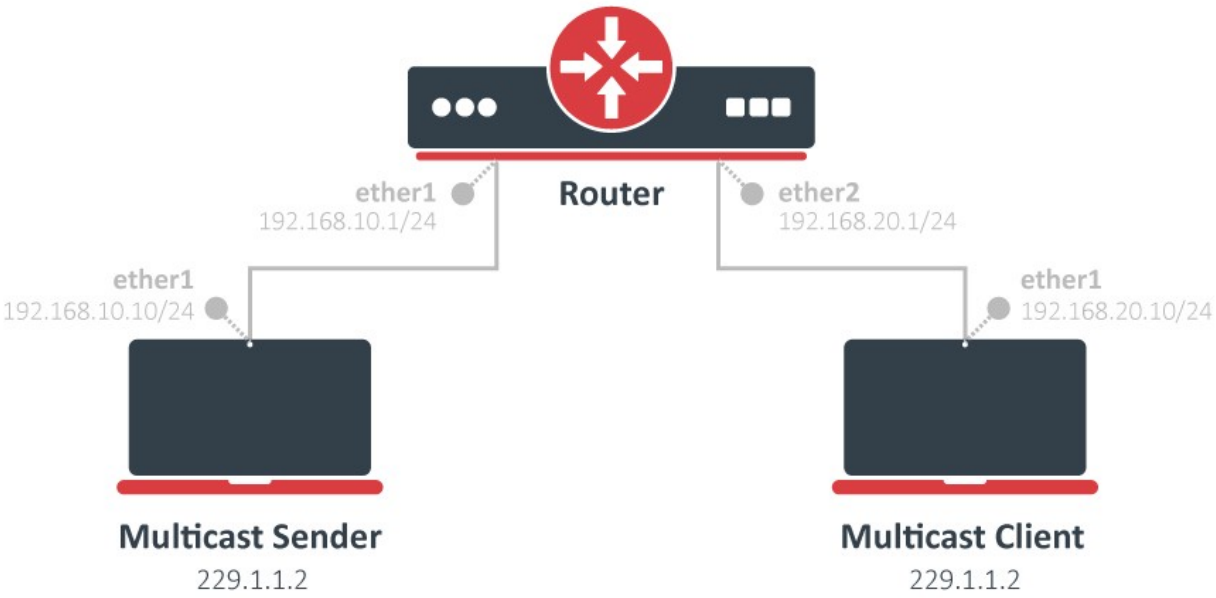


RouterOS v7 has PIM-SM configuration available in the main **system** package. Older RouterOS versions need an additional **multicast** package installed in order to use PIM-SM. See more details about Packages.

The feature is not supported on SMIPS devices (hAP lite, hAP lite TC and hAP mini).

Basic multicast routing on single device

Picture this scenario, you have got a router with two interfaces, namely ether1 and ether2, and each of them is set up in separate networks. Normally, the router will create connected routes and hosts on both networks will be able to communicate using unicast traffic. However, if you want to enable multicast communication between these networks, you'll need to configure multicast routing separately because it won't work otherwise. In this scenario, we are going to create a simple configuration. This involves creating a PIM instance and configuring the required interfaces.



Begin by ensuring that IP addresses are set up on the router's interfaces.

```
/ip address
add address=192.168.10.1/24 interface=ether1 network=192.168.10.0
add address=192.168.20.1/24 interface=ether2 network=192.168.20.0
```

Configure PIM instance. For this example, the default settings should work fine.

```
/routing pimsm instance
add name=pimsm-instance-1
```

Last, add interfaces and specify the PIM instance you created earlier.

```
/routing pimsm interface-template
add interfaces=ether1,ether2 instance=pimsm-instance-1
```

Now router starts listening to IGMP membership reports (client join messages) and will route multicast traffic to clients interested in receiving it.

To test the configuration, you can configure a multicast sender using RouterOS traffic-generator and IGMP client using GMP.

```
# Multicast Sender
/ip address
add address=192.168.10.10/24 interface=ether1 network=192.168.10.0
/tool traffic-generator packet-template
add interface=ether1 ip-dst=229.1.1.2 mac-dst=01:00:5E:01:01:02/FF:FF:FF:FF:FF:FF name=multicast
/tool traffic-generator quick tx-template=multicast mbps=10

# Multicast Client
/ip address
add address=192.168.20.10/24 interface=ether1 network=192.168.20.0
/routing gmp
add disabled=no groups=229.1.1.2 interfaces=ether1
```

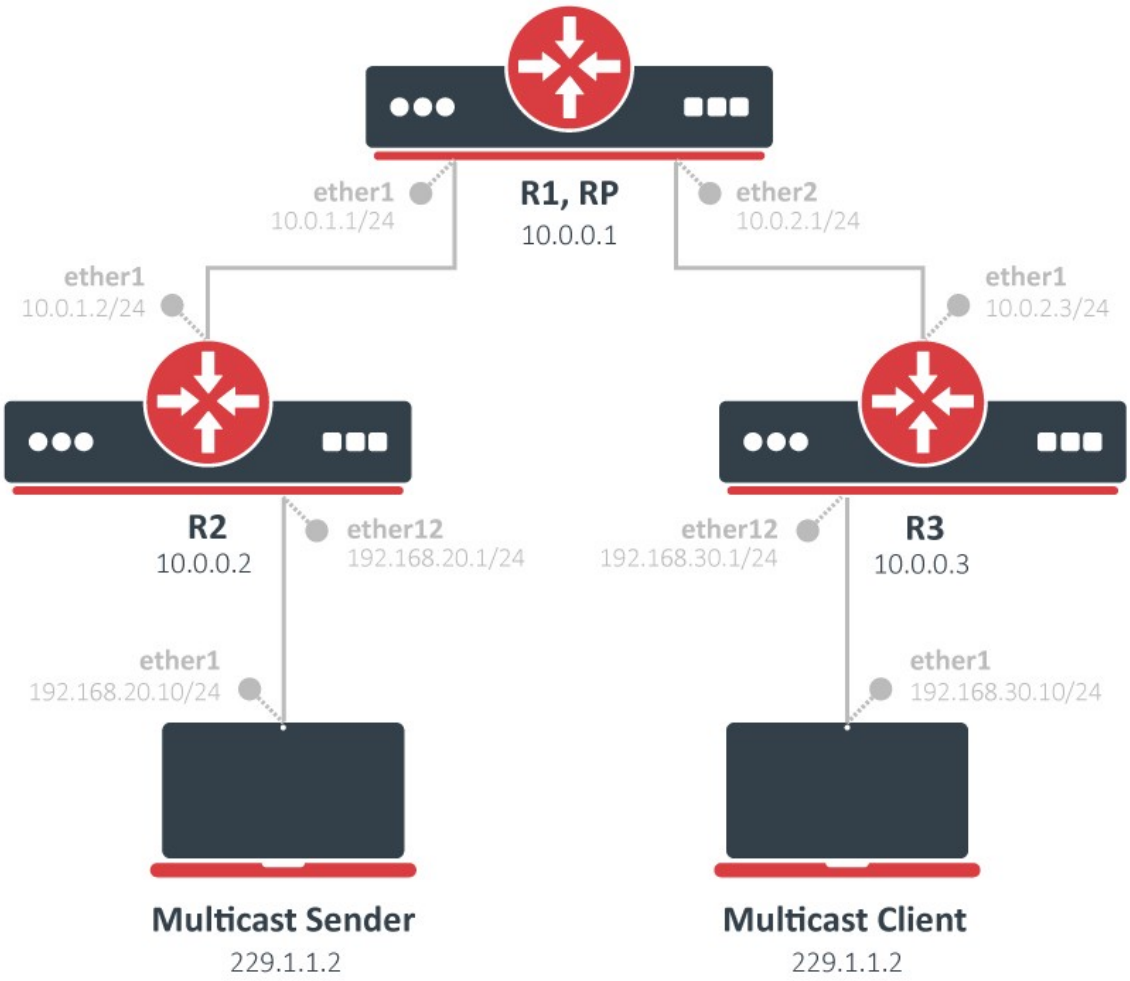
To verify whether multicast traffic is being properly routed, monitor the received packet counters on the client interface or use tools like Torch or a Packet Sniffer.

It is also possible to monitor active multicast group on router:

```
/routing pimsm uib-g print
Columns: INSTANCE, GROUP
# INSTANCE          GROUP
0 pimsm-instance-1  229.1.1.2

/routing pimsm uib-sg print
Flags: K - KEEPALIVE; S - SPT-BIT
Columns: INSTANCE, GROUP, SOURCE
#    INSTANCE          GROUP      SOURCE
0 KS pimsm-instance-1  229.1.1.2  192.168.10.10
```

Multicast routing with static RP

In the upcoming example, we'll be working with multiple PIM routers, as shown in the diagram below. PIM-SM uses shared trees and to make this work, we need to designate a specific node as the multicast root distribution point. In PIM, this router is called the Rendezvous Point, or RP. There are various methods for selecting an RP in PIM, such as the Bootstrap Router (BSR) method. However, for this example, we'll be using a straightforward approach known as static RP configuration. This means that the administrator can manually specify one or more RPs for specific multicast groups.



To get started, we'll need to configure IP addresses and set up unicast routing. In this example, we'll use OSPF to exchange routing information between the routers. See more details about OSFP.

```
# R1 Rendezvous Point:
/interface bridge
add name=loopback
/ip address
add address=10.0.0.1 interface=loopback network=10.0.0.1
add address=10.0.1.1/24 interface=ether2 network=10.0.1.0
add address=10.0.2.1/24 interface=ether3 network=10.0.2.0
/routing ospf instance
add disabled=no name=ospf-instance-1 router-id=10.0.0.1
/routing ospf area
add disabled=no instance=ospf-instance-1 name=ospf-area-1
/routing ospf interface-template
add area=ospf-area-1 disabled=no interfaces=loopback,ether2,ether3

# R2:
/interface bridge
add name=loopback
/ip address
add address=10.0.0.2 interface=loopback network=10.0.0.2
add address=10.0.1.2/24 interface=ether1 network=10.0.1.0
add address=192.168.20.1/24 interface=ether12 network=192.168.20.0
/routing ospf instance
add disabled=no name=ospf-instance-1 router-id=10.0.0.2
/routing ospf area
add disabled=no instance=ospf-instance-1 name=ospf-area-1
/routing ospf interface-template
add area=ospf-area-1 disabled=no interfaces=loopback,ether1,ether12

# R3:
/interface bridge
add name=loopback
/ip address
add address=10.0.0.3 interface=loopback network=10.0.0.3
add address=10.0.2.3/24 interface=ether1 network=10.0.2.0
add address=192.168.30.1/24 interface=ether12 network=192.168.30.0
/routing ospf instance
add disabled=no name=ospf-instance-1 router-id=10.0.0.3
/routing ospf area
add disabled=no instance=ospf-instance-1 name=ospf-area-1
/routing ospf interface-template
add area=ospf-area-1 disabled=no interfaces=loopback,ether1,ether12
```

As in the previous example with a single router, we need to configure the PIM instance and add the necessary interfaces on all routers.

```
# R1 Rendezvous Point:
/routing pimsm instance
add disabled=no name=pimsm-instance-1
/routing pimsm interface-template
add instance=pimsm-instance-1 interfaces=loopback,ether2,ether3

# R2:
/routing pimsm instance
add disabled=no name=pimsm-instance-1
/routing pimsm interface-template
add instance=pimsm-instance-1 interfaces=loopback,ether1,ether12

# R3:
/routing pimsm instance
add name=pimsm-instance-1
/routing pimsm interface-template
add instance=pimsm-instance-1 interfaces=loopback,ether1,ether12
```

Now, let's take a look at our PIM neighbors and their current statuses. On R1, there are two neighbors, while on R2 and R3, there's only one neighbor each.

```
# R1 Rendezvous Point:
/routing pimsm neighbor print
Flags: R - DESIGNATED-ROUTER; J - JOIN-TRACKING
Columns: INSTANCE, ADDRESS, PRIORITY
#    INSTANCE          ADDRESS          PRIORITY
0 RJ pimsm-instance-1  10.0.1.2%ether2  1
1 RJ pimsm-instance-1  10.0.2.3%ether3  1

# R2:
/routing pimsm neighbor print
Flags: R - DESIGNATED-ROUTER; J - JOIN-TRACKING
Columns: INSTANCE, ADDRESS, PRIORITY
#    INSTANCE          ADDRESS          PRIORITY
0  J pimsm-instance-1  10.0.1.1%ether1  1

# R3:
/routing pimsm neighbor print
Flags: J - JOIN-TRACKING
Columns: INSTANCE, ADDRESS, PRIORITY
#   INSTANCE          ADDRESS          PRIORITY
0 J pimsm-instance-1  10.0.2.1%ether1  1
```

Finally, we will select one router to act as our Rendezvous Point (RP). We will configure the R1 loopback IP address on all PIM routers. It's important to ensure that each router has the correct routing information to reach the R1 loopback address.

```
# R1 Rendezvous Point:
/routing pimsm static-rp
add address=10.0.0.1 instance=pimsm-instance-1

# R2:
/routing pimsm static-rp
add address=10.0.0.1 instance=pimsm-instance-1

# R3:
/routing pimsm static-rp
add address=10.0.0.1 instance=pimsm-instance-1
```

Property Reference

Instance

The instance menu defines the main PIM-SM settings. The instance is then used for all other PIM-related configurations like interface-template, static RP, and Bootstrap Router.

Sub-menu: `/routing pimsm instance`

Property	Description
afi (*ipv4* | *ipv6*; Default: **ipv4**)	Specifies address family for PIM.
bsm-forward-back (*yes* | *no*; Default:)	Currently not implemented.
crp-advertise-contained (*yes* | *no*; Default:)	Currently not implemented.
name (*text*; Default:)	Name of the instance.
rp-hash-mask-length (*integer: 0..4294967295*; Default: **30** (IPv4), or **126** (IPv6))	The hash mask allows changing how many groups to map to one of the matching RPs.

rp-static-override (*yes* | *no*; Default: **no**)	Changes the selection priority for static RP. When disabled, the bootstrap RP set has a higher priority. When enabled, static RP has a higher priority.
ssm-range (*IPv4* | *IPv6*; Default:)	Currently not implemented.
switch-to-spt (*yes* | *no*; Default: **yes**)	Whether to switch to Shortest Path Tree (SPT) if multicast data bandwidth threshold is reached. The router will not proceed from protocol phase one (register encapsulation) to native multicast traffic flow if this option is disabled. It is recommended to enable this option.
switch-to-spt-bytes (*integer: 0..4294967295*; Default: **0**)	Multicast data bandwidth threshold. Switching to Shortest Path Tree (SPT) happens if this threshold is reached in the specified time interval. If a value of 0 is configured, switching will happen immediately.
switch-to-spt-interval (*time*; Default:)	Time interval in which to account for multicast data bandwidth, used in conjunction with `switch-to-spt-bytes` to determine if the switching threshold is reached.
vrf (*name*; Default: main)	Name of the VRF.

Interface template

The interface template menu defines which interfaces will participate in PIM and what per-interface configuration will be used.

Sub-menu: `/routing pimsm interface-template`

Property	Description
hello-delay (*time*; Default: **5s**)	Randomized interval for the initial Hello message on interface startup or detecting new neighbor.
hello-period (*time*; Default: **30s**)	Periodic interval for Hello messages.
instance (*name*; Default:)	Name of the PIM instance this interface template belongs to.
interfaces (*name*; Default: **all**)	List of interfaces that will participate in PIM.
join-prune-period (*time*; Default: **1m**)	
join-tracking-support (*yes* | *no*; Default: **yes**)	Sets the value of a Tracking (T) bit in the LAN Prune Delay option in the Hello message. When enabled, a router advertises its willingness to disable Join suppression. it is possible for upstream routers to explicitly track the join membership of individual downstream routers if Join suppression is disabled. Unless all PIM routers on a link negotiate this capability, explicit tracking and the disabling of the Join suppression mechanism are not possible.
override-interval (*time*; Default: **2s500ms**)	Sets the maximum time period over which to randomize when scheduling a delayed override Join message on a network that has join suppression enabled.
priority (*integer: 0..4294967295*; Default: **1**)	The Designated Router (DR) priority. A single Designated Router is elected on each network. The priority is used only if all neighbors have advertised a priority option. Numerically largest priority is preferred. In case of a tie or if priority is not used - the numerically largest IP address is preferred.

propagation-delay (*time*; Default: **500ms**)	Sets the value for a prune pending timer. It is used by upstream routers to figure out how long they should wait for a Join override message before pruning an interface that has join suppression enabled.
source-addresses (*IPv4 | IPv6*; Default:)	

Interface

The interface menu shows all interfaces that are currently participating in PIM and their statuses. This menu contains dynamic and read-only entries that get created by defined interface templates.

Sub-menu: `/routing pimsm interface`

Property	Description
address (*IP%interface@vrf*)	Shows IP address, interface, and VRF.
designated-router (*yes | no*)	
dr (*yes | no*)	
dynamic (*yes | no*)	
instance (*name*)	Name of the PIM instance this interface template belongs to.
join-tracking (*yes | no*)	
override-interval (*time*)	
priority (*integer: 0..4294967295*)	
propagation-delay (*time*)	

Neighbor

The neighbor menu shows all detected neighbors that are running PIM and their statuses. This menu contains dynamic and read-only entries.

Sub-menu: `/routing pimsm neighbor`

Property	Description
address (*IP%interface*)	Shows the neighbor's IP address and local interface the neighbor is detected on.
designated-router (*yes | no*)	Shows whether the neighbor is elected as Designated Router (DR).
instance (*name*)	Name of the PIM instance this neighbor is detected on.
join-tracking (*yes | no*)	Indicates the neighbor's value of a Tracking (T) bit in the LAN Prune Delay option in the Hello message.
override-interval (*time*)	Indicates the neighbor's value of the override interval in the LAN Prune Delay option in the Hello message.
priority (*integer: 0..4294967295*)	Indicates the neighbor's priority value.
propagation-delay (*time*)	Indicates the neighbor's value of the propagation delay in the LAN Prune Delay option in the Hello message.
timeout (*time*)	Shows the reminding time after the neighbor is removed from the list if no new Hello message is received. The hold time equals to neighbor's `hello-period` * 3.5.

Static RP

The static-rp menu allows manually defining the multicast group to RP mappings. Such a mechanism is not robust to failures but does at least provide a basic interoperability mechanism.

Sub-menu: `/routing pimsm static-rp`

Property	Description
address (*IPv4* | *IPv6*; Default:)	The IP address of the static RP.
group (*IPv4* | *IPv6*; Default: **224.0.0.0/4**)	The multicast group that belongs to a specific RP.
instance (*name*; Default:)	Name of the PIM instance this static RP belongs to.

Upstream Information Base

The upstream information base menus show the any-source multicast (*,G) and source-specific multicast (S,G) groups and their statuses. These menus contain only read-only entries.

Sub-menu: `/routing pimsm uib-g`

Property	Description
group (*IPv4* | *IPv6*)	The multicast group address.
instance (*name*)	Name of the PIM instance the multicast group is created on.
rp (*IPv4* | *IPv6*)	The address of the Rendezvous Point for this group.
rp-local (*yes* | *no*)	Indicates whether the multicast router itself is RP.
rpf (*IP%interface*)	The Reverse Path Forwarding (RPF) indicates the router address and outgoing interface that a Join message for that group is directed to.

Sub-menu: `/routing pimsm uib-sg`

Property	Description
group (*IPv4* | *IPv6*)	The multicast group address.
instance (*name*)	Name of the PIM instance the multicast group is created on.
keepalive (*yes* | *no*)	
register (*join* | *join-pending* | *prune*)	
rpf (*IP%interface*)	The Reverse Path Forwarding (RPF) indicates the router address and outgoing interface that a Join message for that group is directed to.
source (*IPv4* | *IPv6*)	The source IP address of the multicast group.
spt-bit (*yes* | *no*)	The Shortest Path Tree (SPT) bit indicates whether forwarding is taking place on the (S,G) Shortest Path Tree or on the (*,G) tree. A router can have an (S,G) state and still be forwarding on a (*,G) state during the interval when the source-specific tree is being constructed. When SPT bit is false, only the (*,G) forwarding state is used to forward packets from S to G. When SPT bit is true, both (*,G) and (S,G) forwarding states are used.

Routing Debugging Tools

Routing stats.

/routing/stats/origin

/routing/stats/process

This menu allows to monitor debugging information of all the routing processes.

```
[admin@rack1_b35_CCR1036] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME,
KERNEL-TIME, CUR-BUSY, MAX-BUSY, CUR-CALC, MAX-CALC
 # TASKS                          PRIVATE-  SHARED-ME  PSS        RSS     VMS      RETIRED  ID       PID  RPID
PROCESS  KERNEL-TIME  CUR-BUSY  MAX-BUSY  CUR-CALC  MAX-CALC
 0 routing tables                 768.0KiB  1792.0KiB  2399.0KiB  6.4MiB  22.1MiB       34  main     317     0
2s260ms  1s940ms      10ms      170ms     20ms      1s210ms

rib

 1 fib                            0         0          2263.0KiB  6.2MiB  22.3MiB           fib      351     1
250ms    1s720ms                 1s210ms             1s210ms
 2 ospf                           256.0KiB  256.0KiB   2559.0KiB  6.6MiB  22.3MiB           ospf     384     1
4s710ms  5s210ms                 20ms                20ms
 3 pimsm                          256.0KiB  0          2252.0KiB  5.8MiB  22.3MiB           pim      386     1
200ms    450ms                   10ms                10ms
 4 fantasy                        0         0          2031.0KiB  5.1MiB  22.3MiB           fantasy  388     1
270ms    390ms                   10ms                10ms
 5 configuration and reporting    0         512.0KiB   2351.0KiB  6.4MiB  22.3MiB           static   389     1
310ms    430ms                   10ms                10ms
 6 ldp                            256.0KiB  256.0KiB   2455.0KiB  6.4MiB  22.3MiB           mpls     387     1
340ms    350ms                   40ms                40ms

Copy

 7 rip                            256.0KiB  0          2230.0KiB  5.7MiB  22.3MiB           rip      377     1
230ms    380ms                   10ms                10ms
 8 routing policy configuration   512.0KiB  512.0KiB   2355.0KiB  5.6MiB  22.3MiB           policy   358     1
240ms    390ms                   10ms                10ms
 9 BGP service                    512.0KiB  0          2592.0KiB  6.3MiB  22.3MiB           bgp      364     1
360ms    600ms                   10ms                10ms
10 BFD service                    256.0KiB  0          2206.0KiB  5.7MiB  22.3MiB           12       371     1
230ms    370ms                   10ms                10ms
11 BGP Input 111.11.0.1           512.0KiB  512.0KiB   2560.0KiB  6.4MiB  22.3MiB        1  22       679     1
140ms    350ms                   10ms                10ms
   BGP Output
111.11.0.1

12 Global memory                             256.0KiB                                         global     0     0
```

/routing/stats/step

/routing/fantasy

Fantasy menu is a fancy way to generate large amount of routes for testing purposes. Main benefits of this approach compared to script is the generation speed and simplicity. It is easy to remove all fantasy generated routes just by disabling fantasy rule.

Fantasy uses random generator from hashed route sequence number, seed and other parameters.

Configuration Options

Property	Description
comment (*string*)	
count (*integer:[0..4294967295]*)	How many routes to generate.
dealer-id (*start-[end]:: integer: [0..4294967295]*)	
disabled (*yes* | *no*)	ID reference is not used.
dst-address(*Prefix*)	Prefix from which route will be generated.
gateway (string)	
instance-id (*start-[end]:: integer: [0..4294967295]*)	
name (*string*)	Reference name
offset (*integer:[0..4294967295]*)	Route sequence number offset
prefix-length (start-[end]:: *integer:[0..4294967295]*)	Prefix length for generated route (can be specified as integer range). For example dst-address 192.168.0.0/16 and prefix-length 24 will generate /24 routes from 192.168.0.0/16 subnet.
priv-offset (*start-[end]:: integer: [0..4294967295]*)	
priv-size (*start-[end]:: integer: [0..100000]*)	
scope (*start-[end]:: integer: [0..255]*)	Scope to be set, can be set as range
seed (*string*)	Random generator seed
target-scope (*start-[end]:: integer: [0..255]*)	Target scope to be set, can be set as range
use-hold (*yes* | *no*)	

/routing/route

A read-only table that lists routes from all the address families as well as all filtered routes with all possible route attributes.

Default example output of the table with various route types:

```
[admin@MikroTik] /routing/route> print
Flags: A - ACTIVE; c, s, a, l, y - COPY; H - HW-OFFLOADED
Columns: DST-ADDRESS, GATEWAY, AFI, DISTANCE, SCOPE, TARGET-SCOPE, IMMEDIATE-GW
    DST-ADDRESS                 GATEWAY           AFI   D  SCOPE  TA  IMMEDIATE-GW
 lH 10.0.0.0/8                                    ip4   0
;;; defconf
As  10.0.0.0/8                 10.155.130.1      ip4   1     30  10  10.155.130.1%ether1
 lH 10.155.130.0/25                               ip4   0
Ac  10.155.130.0/25            ether1            ip4   0     10      ether1
 aH 10.155.130.12/32                              ip4   0
 lH 111.13.0.0/24                                 ip4   0
Ac  111.13.0.0/24              ether2            ip4   0     10      ether2
 aH 111.13.0.1/32                                 ip4   0
Ac  111.111.111.2/32           loopback@vrfTest  ip4   0     10      loopback
Ac  2111:4::/64                ether2            ip6   0     10      ether2
Ac  fe80::%ether1/64           ether1            ip6   0     10      ether1
Ac  fe80::%ether2/64           ether2            ip6   0     10      ether2
Ac  fe80::%ether3/64           ether3            ip6   0     10      ether3
Ac  fe80::%ether4/64           ether4            ip6   0     10      ether4
Ac  3333::2/128                loopback@vrfTest  ip6   0     10      loopback
Ac  fe80::%loopback/64         loopback@vrfTest  ip6   0     10      loopback
Ay  111.111.111.2/32&65530:100 loopback@vrfTest  vpn4  0     10   5  loopback
Ay  3333::2/128&65530:100      loopback@vrfTest  vpn6  0     10   5  loopback
A H ether1                                       link  0
A H ether2                                       link  0
A H ether3                                       link  0
A H ether4                                       link  0
A H loopback                                     link  0
```

Detailed example output with some BGP, OSPF, and other routes:

```
[admin@MikroTik] /routing/route> print detail
Flags: X - disabled, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem, a - ldp-address, l - ldp-
mapping, y - copy; H - hw-offloaded;
+ - ecmp, B - blackhole
  o   afi=ip4 contribution=best-candidate dst-address=0.0.0.0/0 routing-table=main gateway=10.155.101.1%ether1
immediate-gw=10.155.101.1%ether1
       distance=110 scope=20 target-scope=10 belongs-to="OSPF route"
       ospf.metric=2 .tag=111 .type=ext-type-1
       debug.fwp-ptr=0x203425A0

 Ad + afi=ip4 contribution=active dst-address=0.0.0.0/0 routing-table=main pref-src="" gateway=10.155.101.1
immediate-gw=10.155.101.1%ether1
       distance=1 scope=30 target-scope=10 vrf-interface=ether1 belongs-to="DHCP route"
       debug.fwp-ptr=0x20342060

 As + afi=ip4 contribution=active dst-address=0.0.0.0/0 routing-table=main pref-src="" gateway=10.155.101.1
immediate-gw=10.155.101.1%ether1
       distance=1 scope=30 target-scope=10 belongs-to="Static route"
       debug.fwp-ptr=0x20342060

 Fb   afi=ip4 contribution=filtered dst-address=1.0.0.0/24 routing-table=main gateway=10.155.101.1 immediate-
gw=10.155.101.1%ether1 distance=20
       scope=40 target-scope=10 belongs-to="BGP IP routes from 10.155.101.217" rpki=valid
       bgp.peer-cache-id=*B000002 .aggregator="13335:172.68.180.1" .as-path="65530,100,9002,13335" .atomic-
aggregate=yes .origin=igp
       debug.fwp-ptr=0x20342960
```

Property		Description
active (*yes* | *no*)		A flag indicates whether the route is elected as Active and eligible to be added to the FIB.
afi (*ip4* | *ip6* | *link*)		Address family this route belongs to.
belongs-to (*string*)		Descriptive info showing from where the route was received.
bgp (*yes* | *no*)		A flag indicates whether this route was added by the BGP protocol.
bgp - a group of parameters associated with the BGP protocol		
	.as-path(*string*)	value of the AS_PATH BGP attribute
	.aggregator (*string*)	
	.atomic-aggregate (*yes* | *no*)	
	.cluster-list (*string*)	
	.communities (*string*)	value of the COMMUNITIES BGP attribute
	.ext-communities (*string*)	value of the EXTENDED_COMMUNITIES BGP attribute
	.igp-metric(*string*)	value of the IGP_METRIC BGP attribute

	.large-communities (*string*)	value of the LARGE_COMMUNITIES BGP attribute
	.local-pref (*string*)	value of the LOCAL_PREF BGP attribute
	.med (*string*)	value of the MED BGP attribute
	.nexthop (*string*)	
	.origin (*string*)	
	.originator-id (*string*)	
	.out-nexthop(*string*)	
	.peer-cache-id (*string*)	The ID of the BGP session that installed the route. See `/routing/bgp/session` menu.
	.unknown (*string*)	hex blob of unknown BGP attributes
	.weight (*string*)	
blackhole (*yes* | *no*)		A flag indicates whether it is a blackhole route
check-gateway (*ping* | *arp* | *bfd*)		Currently used check-gateway option.
comment (*string*)		
connect (*yes* | *no*)		A flag indicates whether it is a connected network route.
contribution (*string*)		Shows the route status contributing to the election process, e.g "filtered, active, candidate"
copy (*yes* | *no*)		A flag indicates a copy of the route to be redistributed as the L3VPN route. VPNv4/6 related attributes are attached to this "copy" route.
create-time (*string*)		
debug - a group of debugging parameters		
dhcp (*yes* | *no*)		A flag indicates whether the route was added by the DHCP service.
disabled (*yes* | *no*)		A flag indicates whether the route is disabled.
distance (*integer*)		
dst-address (*prefix*)		Route destination.
ecmp (*yes* | *no*)		A flag indicates whether the route is added as an Equal-Cost Multi-Path route in the FIB. Read more>>
filtered (*yes* | *no*)		A flag indicates whether the route was filtered by routing filters and excluded from being used as the best route.
gateway (*string*)		Configured gateway, for the actually resolved gateway, see `immediate-gw` parameter.
hw-offloaded (*yes* | *no*)		Indicates whether the route is eligible to be hardware offloaded on supported hardware.
immediate-gw (*string*)		Shows actual (resolved) gateway and interface that will be used for packet forwarding. Displayed in format `[ip% interface]`.
label (*integer*)		
ldp-address (*yes* | *no*)		A flag indicates whether the route entry is an LDP address.
ldp-mapping (*yes* | *no*)		A flag indicates whether the route entry is the LDP mapping

ldp - a group of parameters associated with the LDP protocol	
.label (*integer*)	LDP mapped MPLS label.
.peer-id ()	
local-address (*IP*)	Local IP address of the connected network.
modem (*yes* | *no*)	A flag indicates whether the route is added by the LTE or 3g modems.
mpls - group of generic parameters associated with the MPLS	
.in-label ()	Mapped MPLS ingress label
.labels ()	
.out-label ()	Mapped MPLS egress label
nexthop-id ()	
ospf (*yes* | *no*)	A flag indicates whether the route was added by the OSPF routing protocol.
ospf - group of parameters associated with the OSPF protocol	
.metric (*integer*)	
.type (*string*)	
pref-src ()	
received-from ()	
rip (*yes* | *no*)	A flag indicates whether the route was added by the RIP routing protocol
rip - group of parameters associated with the RIP protocol	
.metric ()	
.route-tag ()	
route-cost ()	
routing-table ()	Routing table this route belongs to.
rpki *(valid | invalid | unknown)*	Current status of the prefix from the RPKI validation process.
scope (*integer*)	Scope used in the next-hop lookup process. Read more>>
static (*yes* | *no*)	A flag indicates statically added routes.
target-scope (*integer*)	Target scope used in next-hop lookup process. Read more>>
te-tunnel-id ()	Traffic Engineering tunnel ID
total-cost ()	
unreachable (*yes* | *no*)	A flag indicates whether the route next-hop is unreachable.
update-time ()	
ve-block-offset	
ve-block-size	
ve-id	
vpn (*yes* | *no*)	A flag indicates whether the route was added by one of the VPN protocols (PPPoE, L2TP, SSTP, etc.)
vrf-interface ()	Internal use only parameter which allows identifying to which VRF route should be added. Used by services that add routes dynamically, for example, DHCP client. Shown for debugging purposes.

Routing Reference

/routing/id

Global Router ID election configuration. ID can be configured explicitly or set to be elected from one of the Routers IP addresses.

For each VRF table RouterOS adds dynamic ID instance, that elects the ID from one of the IP addresses belonging to a particular VRF:

```
[admin@rack1_b33_CCR1036] /routing/id> print
Flags: D - DYNAMIC, I - INACTIVE
Columns: NAME, DYNAMIC-ID, SELECT-DYNAMIC-ID, SELECT-FROM-VRF
#   NAME   DYNAMIC-ID      SELECT-D   SELE
0 D main   111.111.111.2   only-vrf   main
```

Configuration Options

Property	Description
comment (*string*)	
disabled (*yes* | *no*)	ID reference is not used.
id(*IP*)	Parameter to explicitly set the Router ID. If ID is not explicitly specified, then it can be elected from one of the configured IP addresses on the router. See parameters select-dynamic-id and select-from-vrf.
name (*string*)	Reference name
select-dynamic-id(*any* | *lowest* | *only-active* | *only-loopback* | *only-static* | *only-vrf*)	States what IP addresses to use for the ID election: • any - any address found on the router can be elected as the Router ID. • lowest - pick the lowest IP address. • only-active - pick an ID only from active IP addresses. • only-loopback - pick an ID only from loopback addresses. • only-vrf - pick an ID only from selected VRF. Works with select-from-vrf property.
select-from-vrf (*name*)	VRF from which to select IP addresses for the ID election.

Read-only Properties

Property	Description
dynamic (*yes* | *no*)	
dynamic-id (*IP*)	Currently selected ID.
inactive (*yes* | *no*)	If there was a problem to get a valid ID, then item can become inactive.

BFD

Summary

Bidirectional Forwarding Detection (BFD) is a low-overhead and short-duration protocol intended to detect faults in the bidirectional path between two forwarding engines, including physical interfaces, sub-interfaces, data link(s), and to the extent possible the forwarding engines themselves, with potentially very low latency. It operates independently of media, data protocols, and routing protocols.

BFD is basically a hello protocol for checking bidirectional neighbor reachability. It provides sub-second link failure detection support. BFD is not routing protocol specific, unlike protocol hello timers or such.

BFD Control packets are transmitted in UDP packets with destination port 3784, BFD also uses port 4784 for multihop paths. The source port is in the range 49152 through 65535. And BFD Echo packets are encapsulated in UDP packets with destination port 3785.

Standards and Technologies:

- RFC 5880 Bidirectional Forwarding Detection (BFD)
- RFC 5881 BFD for IPv4 and IPv6
- RFC 5882 Generic Application of BFD
- RFC 5883 Bidirectional Forwarding Detection (BFD) for Multihop Paths

Features not yet supported

- echo mode
- enabling BFD for ip route gateways
- authentication

Configuration

Allowing or forbidding BFD sessions can be done from the `/routing bfd configuration` menu. For example:

```
/routing bfd configuration
add interfaces=sfp12 forbid-bfd=yes
add interfaces=static
```

Configuration entries are order sensitive, which means that in the example above we are forbidding BFD sessions explicitly on the "sfp12" interface and allowing on the rest of the interfaces belonging to the "static" interface list.

To be able to filter multi-hop sessions, `addresses` or `address-list` properties can be used to match the destination, as well as the appropriate VRF, if a session is not running in the "main" VRF.

```
/ip firewall address-list
add address=10.155.255.183 list=bgp_allow_bfd
add address=10.155.255.217 list=bgp_allow_bfd

/routing bfd configuration
add addresses=111.111.0.0/16 vrf=vrf1
add address-list=bgp_allow_bfd
```

Everything else that is not explicitly listed in the configuration by default is forbidden.

BFD with BGP

To enable the use of BFD for BGP sessions, enable `use-bfd` for required entries in `/routing bgp connection` menu.

A useful feature is that the BGP session will show that the BFD session for that particular BGP session is down:

```
[admin@dr_02_BGP_MUM] /routing/bgp/session> print
Flags: E - established
 0 E ;;; BFD session down
     name="ovpn_test1-1"
     remote.address=111.111.11.11@vrf1 .as=65530 .id=10.155.101.217
     .capabilities=mp,rr,as4 .hold-time=infinity .messages=40717
     .bytes=3436281 .eor=""
     local.address=111.111.11.12@vrf1 .as=555 .id=111.111.11.12
     .capabilities=mp,rr,gr,as4 .messages=1 .bytes=19 .eor=""
     output.procid=20
     input.procid=20 .filter=bgp-in ebgp
     hold-time=infinity use-bfd=yes uptime=3s210ms
     last-started=2023-05-19 09:54:04 prefix-count=3853
```

BFD with OSPF

To enable the use of BFD for OSPF neighbors, enable `use-bfd` for required entries in `/routing ospf interface-template` menu.

Session Status

The status of the currently available sessions can be observed from `/routing bfd session` menu:

```
[admin@dr_02_BGP_MUM] /routing/bfd/session> print
Flags: U - up, I - inactive
 0 I ;;; BFD forbidden for destination address
     multihop=yes remote-address=10.155.101.183 local-address="" desired-tx-interval=0ms required-min-rx=0ms
     multiplier=0

 1   multihop=no remote-address=111.111.11.11%ovpn-out1@vrf1 local-address=111.111.11.12@vrf1 state=down
     state-changes=0 desired-tx-interval=200ms required-min-rx=200ms remote-min-rx=1us multiplier=5
     packets-rx=0 packets-tx=7674
```

BFD is picking the highest value between the local tx interval and remote minimum rx interval as desired transmit interval. If the session is not established then desired minimum tx interval is set to 1 second.

IS-IS

Overview

The IS-IS (Intermediate System - Intermediate System) protocol is an Interior Gateway Protocol (IGP) used to distribute IP routing information throughout a single Autonomous System.

It was originally developed as a routing protocol for CLNP but later extended to include IP routing when IP became popular. An extended version is sometimes referred to as Integrated IS-IS.

IS-IS belongs to the link-state protocol family, which exchanges topology information between nearest neighbors and floods it throughout the AS. The main advantage is that complete knowledge of the network topology allows one to choose the best path to the destination. It can also be useful for traffic engineering purposes.

Neighbours periodically exchange **Hello** packets, forms adjacency and selects designated IS based on the negotiation. Hello packets are sent individually for **Level-1** and **Level-2**.

Standards and Technologies:

- RFC 1195 Use of OSI IS-IS for Routing in TCP/IP and Dual Environments
- RFC 5302 Domain-Wide Prefix Distribution with Two-Level IS-IS
- RFC 5303 Three-Way Handshake for IS-IS Point-to-Point Adjacencies
- RFC 5305 IS-IS Extensions for Traffic Engineering (only wide metric support)
- RFC 5308 Routing IPv6 with IS-IS

IS-IS Terminology

- **IS** - Intermediate System is a router capable of forwarding traffic between distantly located hosts.
- **LSP** - Link State PDU contains information on the router's local state (usable interfaces, reachable neighbours, and the cost of the interfaces)
- **SPF** - Shortest-path-first algorithm
- **DIS** - designated intermediate system. DIS ensures that all routes in the network maintain synchronised database. Separate DISs are elected for L1 and L2 routing. Election of the DIS is based on the highest interface priority.
- **Level-1 (L1) routing** - Controls distribution of routing information within an IS-IS area. L1 routing is based on system ID.
- **Level-2 (L2) routing** - Controls distribution of routing information between IS-IS areas. L2 routing is based on area ID.
- **IS-IS Adjacency** - link between IS-IS neighbours. The type of adjacency formed depends on the parameters exchanged in the IS-IS Hello packets. Each of the the adjacent routers runs the DIS election process to determine whether it is eligible to be an L1 or L2 DIS on the broadcast network.

Basic Configuration Example

Basic IS-IS setup between three routers.

R1:

```
/routing isis instance
add afi=ip areas=49.2222 disabled=no name=isis-instance-1 system-id=90ab.cdef.0001
/routing isis interface-template
add instance=isis-instance-1 interfaces=ether1 levels=l1,l2


[] /routing/isis/neighbor> print
 0 instance=isis-instance-1 interface=ether1 level-type=l2 snpa=08:00:27:22:B4:A2 srcid="1111.2222.aded"
state=up

 1 instance=isis-instance-1 interface=ether1 level-type=l2 snpa=D4:CA:6D:78:2F:2E srcid="1111.2222.cded"
state=up

 2 instance=isis-instance-1 interface=ether1 level-type=l1 snpa=08:00:27:22:B4:A2 srcid="1111.2222.aded"
state=up

 3 instance=isis-instance-1 interface=ether1 level-type=l1 snpa=D4:CA:6D:78:2F:2E srcid="1111.2222.cded"
state=up


[] /routing/route> print where is-is
Flags: A - ACTIVE; i - IS-IS
Columns: DST-ADDRESS, GATEWAY, AFI, DISTANCE, SCOPE, TARGET-SCOPE, IMMEDIATE-GW
   DST-ADDRESS        GATEWAY               AFI  DISTANCE  SCOPE  TARGET-SCOPE  IMMEDIATE-GW
 i 0.0.0.0/0          10.155.101.214%ether1  ip4       115     20            10  10.155.101.214%ether1
 i 10.155.101.0/24    10.155.101.216%ether1  ip4       115     20            10  10.155.101.216%ether1
Ai 10.255.255.162/32  10.155.101.216%ether1  ip4       115     20            10  10.155.101.216%ether1
```

R2:

```
/routing isis instance
add afi=ip areas=49.2222 disabled=no l1.originate-default=always l2.originate-default=always name=isis-instance-
1 \
    system-id=1111.2222.cded
/routing isis interface-template
add instance=isis-instance-1 interfaces=sfp12 levels=l1,l2
add instance=isis-instance-1 interfaces=lo levels=l2


[] /routing/isis/neighbor> print
 0 instance=isis-instance-1 interface=sfp12 level-type=l1 snpa=08:00:27:22:B4:A2 srcid="1111.2222.aded"
state=up

 1 instance=isis-instance-1 interface=sfp12 level-type=l1 snpa=C4:AD:34:43:EA:5C srcid="90ab.cdef.0001"
state=up

 2 instance=isis-instance-1 interface=sfp12 level-type=l2 snpa=08:00:27:22:B4:A2 srcid="1111.2222.aded"
state=up

 3 instance=isis-instance-1 interface=sfp12 level-type=l2 snpa=C4:AD:34:43:EA:5C srcid="90ab.cdef.0001"
state=up
```

R3 Cisco:

```
interface Loopback0
 ip address 10.255.255.162 255.255.255.255
 ip router isis
!
interface GigabitEthernet1
 ip address dhcp
 ip router isis
 negotiation auto
!
router isis
 net 49.2222.1111.2222.aded.00
!



# show isis neighbors

Tag null:
System Id      Type Interface   IP Address      State Holdtime Circuit Id
90AB.CDEF.0001 L1   Gi1         10.155.101.183  UP    27       1111.2222.CDED.01
90AB.CDEF.0001 L2   Gi1         10.155.101.183  UP    27       1111.2222.CDED.01
1111.2222.CDED L1   Gi1         10.155.101.214  UP    9        1111.2222.CDED.01
1111.2222.CDED L2   Gi1         10.155.101.214  UP    9        1111.2222.CDED.01

# show ip route

i*L1  0.0.0.0/0 [115/11] via 10.155.101.214, 4w5d, GigabitEthernet1
      10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
C        10.155.101.0/24 is directly connected, GigabitEthernet1
L        10.155.101.216/32 is directly connected, GigabitEthernet1
i L2     10.155.255.214/32 [115/10] via 10.155.101.183, 2w3d, GigabitEthernet1
```

Scripting

Scripting language manual

This manual provides an introduction to RouterOS's built-in powerful scripting language.

Scripting host provides a way to automate some router maintenance tasks by means of executing user-defined scripts bounded to some event occurrence.

Scripts can be stored in the Script repository or can be written directly to the console. The events used to trigger script execution include, but are not limited to the System Scheduler, the Traffic Monitoring Tool, and the Netwatch Tool generated events.

If you are already familiar with scripting in RouterOS, you might want to see our Tips & Tricks.

Line structure

The RouterOS script is divided into a number of command lines. Command lines are executed one by one until the end of the script or until a runtime error occurs.

Command-line

The RouterOS console uses the following command syntax:

```
[prefix] [path] command [uparam] [param=[value]] .. [param=[value]]
```

- [prefix] - ":" or "/" character which indicates if a command is ICE or path. It may not be required.
- [path] - relative path to the desired menu level. It may not be required.
- command - one of the commands available at the specified menu level.
- [uparam] - unnamed parameter, must be specified if the command requires it.
- [params] - a sequence of named parameters followed by respective values

The end of the command line is represented by the token *";"* or *NEWLINE*. Sometimes *";"* or *NEWLINE* is not required to end the command line.

Single command inside `(), [] or {}` does not require any end-of-command character. The end of the command is determined by the content of the whole script

```
:if ( true ) do={ :put "lala" }
```

Each command line inside another command line starts and ends with square brackets "[ ]" (command concatenation).

```
:put [/ip route get [find gateway=1.1.1.1]];
```

Notice that the code above contains three command lines:

- :put
- /ip route get
- find gateway=1.1.1.1

Command-line can be constructed from more than one physical line by following line joining rules.

Physical Line

A physical line is a sequence of characters terminated by an end-of-line (EOL) sequence. Any of the standard platform line termination sequences can be used:

- **Unix** – ASCII LF;
- **Windows** – ASCII CR LF;
- **mac** – ASCII CR;

Standard C conventions for newline characters can be used (the \n character).

Comments

The following rules apply to a comment:

- A comment starts with a hash character (#) and ends at the end of the physical line.
- RouterOS does not support multiline comments.
- If a **#** character appears inside the string it is not considered a comment.

Example

```
# this is a comment
# next line comment
:global a; # another valid comment

:global myStr "part of the string # is not a comment"
```

Line joining

Two or more physical lines may be joined into logical lines using the backslash character (\).

The following rules apply to using backslash as a line-joining tool:

- A line ending in a backslash cannot carry a comment.
- A backslash does not continue a comment.
- A backslash does not continue a token except for string literals.
- A backslash is illegal elsewhere on a line outside a string literal.

Example

```
:if ($a = true \
        and $b=false) do={ :put "$a $b"; }
:if ($a = true \ # bad comment
        and $b=false) do={ :put "$a $b"; }
# comment \
        continued - invalid (syntax error)
```

Whitespace between tokens

Whitespace can be used to separate tokens. Whitespace is necessary between two tokens only if their concatenation could be interpreted as a different token. Example:

```
{
        :local a true; :local b false;
# whitespace is not required
        :put (a&&b);
# whitespace is required
        :put (a and b);
}
```

Whitespace characters are not allowed

- between '<parameter>='
- between 'from=' 'to=' 'step=' 'in=' 'do=' 'else='

Example:

```
#incorrect:
:for i from = 1 to = 2 do = { :put $i }
#correct syntax:
:for i from=1 to=2 do={ :put $i }
:for i from= 1 to= 2 do={ :put $i }

#incorrect
/ip route add gateway = 3.3.3.3
#correct
/ip route add gateway=3.3.3.3
```

Scopes

Variables can be used only in certain regions of the script called scopes. These regions determine the visibility of the variable. There are two types of scopes - global and local. A variable declared within a block is accessible only within that block and blocks enclosed by it, and only after the point of declaration.

Global scope

Global scope or root scope is the default scope of the script. It is created automatically and can not be turned off.

Local scope

User can define their own groups to block access to certain variables, these scopes are called local scopes. Each local scope is enclosed in curly braces ("{ }").

```
{
        :local a 3;
        {
                :local b 4;
                :put ($a+$b);
        } #line below will show variable b in light red color since it is not defined in scope
        :put ($a+$b);
}
```

In the code above variable, b has local scope and will not be accessible after a closing curly brace.

⚠ Each line written in the terminal is treated as local scope

So for example, the defined local variable will not be visible in the next command line and will generate a syntax error

```
[admin@MikroTik] > :local myVar a;
[admin@MikroTik] > :put $myVar
syntax error (line 1 column 7)
```

⛔ Do not define global variables inside local scopes.

Note that even variable can be defined as global, it will be available only from its scope unless it is not referenced to be visible outside of the scope.

```
{
        :local a 3;
        {
                :global b 4;
        }
        :put ($a+$b);
}
```

The code above will output 3, because outside of the scope b is not visible.

The following code will fix the problem and will output 7:

```
{
        :local a 3;
        {
                :global b 4;
        }
        :global b;
        :put ($a+$b);
}
```

ⓘ Do not use global or local variables with the same names as the arguments used in the console.

Example:

Incorrect will return all entries.

```
:local name aa; /file/print where name=$name
```

Correct.

```
:local nm aa; /file/print where name=$nm
```

Keywords

The following words are keywords and cannot be used as variable and function names:

```
and       or       in
```

Delimiters

The following tokens serve as delimiters in the grammar:

```
()  []  {}  :   ;   $   /
```

Data types

RouterOS scripting language has the following data types:

Type	Description
num (number)	- 64bit signed integer, possible hexadecimal input;
bool (boolean)	- values can bee `true` or `false`;
str (string)	- character sequence;
ip	- IP address;
ip-prefix	- IP prefix;
ip6	- IPv6 address
ip6-prefix	- IPv6 prefix
id (internal ID)	- hexadecimal value prefixed by '*' sign. Each menu item has an assigned unique number - internal ID;
time	- date and time value;
array	- sequence of values organized in an array;
nil	- default variable type if no value is assigned;

Constant Escape Sequences

Following escape sequences can be used to define certain special characters within a string:

\"	Insert double quote
****	Insert backslash
\n	Insert newline
\r	Insert carriage return
\t	Insert horizontal tab
\$	Output $ character. Otherwise, $ is used to link the variable.
\?	~~Output ? character. Otherwise ? is used to print "help" in the console.~~ Removed since v7.1rc2
_	- space
\a	- BEL (0x07)
\b	- backspace (0x08)
\f	- form feed (0xFF)

\v	Insert vertical tab
\xx	A print character from hex value. Hex numbers should use capital letters.

Example

```
:put "\48\45\4C\4C\4F\r\nThis\r\nis\r\na\r\ntest";
```

which will show on the display
```
HELLO
This
is
a
test
```

Operators

Arithmetic Operators

Usual arithmetic operators are supported in the RouterOS scripting language

Operator	Description	Example
"+"	binary addition	`:put (3+4);`
"-"	binary subtraction	`:put (1-6);`
"*"	binary multiplication	`:put (4*5);`
"/"	binary division	`:put (10 / 2); :put ((10)/2)`
"%"	modulo operation	`:put (5 % 3);`
"-"	unary negation	`{ :local a 1; :put (-a); }`

Note: for the division to work you have to use braces or spaces around the dividend so it is not mistaken as an IP address

Relational Operators

Operator	Description	Example
"<"	less	`:put (3<4);`
">"	greater	`:put (3>4);`
"="	equal	`:put (2=2);`
"<="	less or equal	
">="	greater or equal	
"!="	not equal	

Logical Operators

Operator	Description	Example
"!"	logical NOT	`:put (!true);`
"&&", "and"	logical AND	`:put (true&&true)`
"||", "or"	logical OR	`:put (true||false);`
"in"		`:put (1.1.1.1/32 in 1.0.0.0/8);`

Bitwise Operators

Bitwise operators are working on number, IP, and IPv6 address data types.

Operator	Description	Example
“~”	bit inversion	`:put (~0.0.0.0)` `:put (~::ffff)`
“|”	bitwise OR. Performs logical OR operation on each pair of corresponding bits. In each pair the result is “1” if one of the bits or both bits is “1”, otherwise the result is “0”.	`:put (192.168.88.0|0.0.0.255)` `:put (2001::1|::ffff)`
“^”	bitwise XOR. The same as OR, but the result in each position is “1” if two bits are not equal, and “0” if the bits are equal.	`:put (1.1.1.1^255.255.0.0)` `:put (2001::ffff:1^::ffff:0)`
“&”	bitwise AND. In each pair, the result is “1” if the first and second bit is “1”. Otherwise, the result is “0”.	`:put (192.168.88.77&255.255.255.0)` `:put (2001::1111&ffff::)`
“<<”	left shift by a given amount of bits, not supported for IPv6 address data type	`:put (192.168.88.77<<8)`
“>>”	right shift by a given amount of bits, not supported for IPv6 address data type	`:put (192.168.88.77>>24)`

Calculate the subnet address from the given IP and CIDR Netmask using the "&" operator:

```
{
:local IP 192.168.88.77;
:local CIDRnetmask 255.255.255.0;
:put ($IP&$CIDRnetmask);
}
```

Get the last 8 bits from the given IP addresses:

```
 :put (192.168.88.77&0.0.0.255);
```

Use the "|" operator and inverted CIDR mask to calculate the broadcast address:

```
{
:local IP 192.168.88.77;
:local Network 192.168.88.0;
:local CIDRnetmask 255.255.255.0;
:local InvertedCIDR (~$CIDRnetmask);
:put ($Network|$InvertedCIDR)
}
```

Concatenation Operators

Operator	Description	Example
“.”	concatenates two strings	`:put ("concatenate" . " " . "string");`
“,”	concatenates two arrays or adds an element to the array	`:put ({1;2;3} , 5 );`

It is possible to add variable values to strings without a concatenation operator:

```
:global myVar "world";

:put ("Hello " . $myVar);
# next line does the same as above
:put "Hello $myVar";
```

By using $[] and $() in the string it is possible to add expressions inside strings:

```
:local a 5;
:local b 6;
:put " 5x6 = $($a * $b)";

:put " We have $[ :len [/ip route find] ] routes";
```

Other Operators

Operator	Description	Example
"[]"	command substitution. Can contain only a single command line	`:put [ :len "my test string"; ];`
"()"	subexpression or grouping operator	`:put ( "value is " . (4+5));`
"$"	substitution operator	`:global a 5; :put $a;`
"~"	the binary operator that matches value against POSIX extended regular expression	Print all routes whose gateway ends with 202 `/ip route print where gateway~"^[0-9 \\.]*202\$"`

"->"	Get an array element by key	`[admin@x86] >:global aaa {a=1;b=2}`
		`[admin@x86] > :put ($aaa->"a")`
		`1`
		`[admin@x86] > :put ($aaa->"b")`
		`2`

Variables

The scripting language has two types of variables:

- global - accessible from all scripts created by the current user, defined by global keyword;
- local - accessible only within the current scope, defined by local keyword.

There can be **undefined** variables. When a variable is undefined, the parser will try to look for variables set, for example, by DHCP lease-script or Hotspot on-login

Every variable, except for built-in RouterOS variables, must be declared before usage by local or global keywords. Undefined variables will be marked as undefined and will result in a compilation error. Example:

```
# following code will result in compilation error, because myVar is used without declaration
:set myVar "my value";
:put $myVar
```

Correct code:

```
:local myVar;
:set myVar "my value";
:put $myVar;
```

The exception is when using variables set, for example, by DHCP lease-script

```
/system script
add name=myLeaseScript policy=\
        ftp,reboot,read,write,policy,test,winbox,password,sniff,sensitive,api \
        source=":log info \$leaseActIP\r\
        \n:log info \$leaseActMAC\r\
        \n:log info \$leaseServerName\r\
        \n:log info \$leaseBound"

/ip dhcp-server set myServer lease-script=myLeaseScript
```

Valid characters in variable names are letters and digits. If the variable name contains any other character, then the variable name should be put in double quotes. Example:

```
#valid variable name
:local myVar;
#invalid variable name
:local my-var;
#valid because double quoted
:global "my-var";
```

If a variable is initially defined without value then the variable data type is set to *nil*, otherwise, a data type is determined automatically by the scripting engine. Sometimes conversion from one data type to another is required. It can be achieved using data conversion commands. Example:

```
#convert string to array
:local myStr "1,2,3,4,5";
:put [:typeof $myStr];
:local myArr [:toarray $myStr];
:put [:typeof $myArr]
```

Variable names are case-sensitive.

```
:local myVar "hello"
# following line will generate error, because variable myVAr is not defined
:put $myVAr
# correct code
:put $myVar
```

Set command without value will un-define the variable (remove from environment, new in v6.2)

```
#remove variable from environment
:global myVar "myValue"
:set myVar;
```

Use quotes on the full variable name when the name of the variable contains operators. Example:

```
:local "my-Var";
:set "my-Var" "my value";
:put $"my-Var";
```

Reserved variable names

All built-in RouterOS properties are reserved variables. Variables that will be defined the same as the RouterOS built-in properties can cause errors. To avoid such errors, use custom designations.

For example, the following script will not work:

```
{
:local type "ether1";
/interface print where name=$type;
}
```

But will work with different defined variables:

```
 {
:local customname "ether1";
/interface print where name=$customname;
}
```

Commands

Global commands

Every global command should start with the *":"* token, otherwise, it will be treated as a variable.

Command	Syntax	Description	Example
/		go to the root menu	
..		go back by one menu level	
?		list all available menu commands and brief descriptions	
global	`:global <var> [<value>]`	define a global variable	`:global myVar "something"; :put $myVar;`
local	`:local <var> [<value>]`	define the local variable	`{ :local myLocalVar "I am local"; :put $myVar; }`
beep	`:beep <freq> <length>`	beep built-in speaker	

convert	`:convert from=[arg] to=[arg]`	Converts specified value from one format to another. By default uses an automatically parsed value, if the "from" format is not specified (for example, "001" becomes "1", "10.1" becomes "10.0.0.1", etc.).
		
		from specifies the format of the value - *base32, base64, hex, raw, url.*
		
		to specifies the format of the output value - *base32, base64, hex, raw, url.*
			`:put [:convert 001 to=hex ]`
			
			`31`
			
			`:put [:convert [/ip dhcp-client /option/get hostname raw-value] from=hex to=raw ]`
			
			`MikroTik`

delay	`:delay <time>`	do nothing for a given period of time	
environment	`:environment print <start>`	print initialized variable information	`:global myVar true; :environment print;`
error	`:error <output>`	Generate console error and stop executing the script	

execute	`:execute <expression>`	Execute the script in the background. The result can be written in the file by setting a "file" parameter or printed to the CLI by setting "as-string".
		
		When using the "as-string" parameter executed script is blocked (not executed in the background).
			`{`
			`:local j [:execute {`
			`/interface print follow`
			`where [:log info ~Sname~]}];`
			`:delay 10s;`
			`:do { /system script job remove $j } on-error={}`
			`}`

find	`:find <arg> <arg> <start>`	return position of a substring or array element	`:put [:find "abc" "a" -1];`

jobname	:jobname	return current script name	**Limit script execution to single instance** `:if ([/system script job print count-only as-value where script=[:jobname] ] > 1) do={ :error "script instance already running" }`
len	`:len <expression>`	return string length or array element count	`:put [:len "length=8"];`
log	`:log <topic> <message>`	write a message to the system log. Available topics are `"debug, error, info and warning"`	`:log info "Hello from script";`
onerror	`:onerror <var_name> in={<command>} do={<expression>}`	The command used to catch errors and get error details. The **do={...}** block is executed, when **in={...}** block has an error, and error details are written in <var_name> variable. Parameter order is important. The "error" parameter must be set before "do" block, otherwise do block will not see the local variable. :onerror can return *false* (if there is no error) and *true* (if there is an error) values, so it can be used in **:if** condition statement scripts.	`:onerror errorName in={ :error "failure" } do={ :put "Critical $errorName" }`
parse	`:parse <expression>`	parse the string and return parsed console commands. Can be used as a function.	`:global myFunc [:parse ":put hello!"];` `$myFunc;`
pick	`:pick <var> <start> [<end>]`	return range of elements or substring. If the end is not specified, will return only one element from an array. • var - value to pick elements from • start - starting index to start picking from (the first element index is 0) • end - terminating index (element at this index is not included)	`[admin@MikroTik] > :put [: pick "abcde" 1 3]` `bc`
put	`:put <expression>`	put the supplied argument into the console	:put "Hello world"
resolve	`:resolve <arg>`	return the IP address of the given DNS name	`:put [:resolve "www.mikrotik.com"];`

retry	:retry command=<expr> delay=[num] max=[num] on-error=<expr>	Try to execute the given command "max" amount of times with a given "delay" between tries. On failure, execute the expression given in the "on-error" block	:retry command={abc} delay=1 max=2 on-error={:put "got error"}
			got error
			
			`:retry command={abc} delay=1 max=2 on-error={:put "got error"}`
			`got error`

serialize	:serialize [<value>] to=[arg]	Serialize specified value/array to JSON format. Specify the format using **to=json**.	`:put [:serialize value=a,b,c to=json]` `["a","b","c"]`
deserialize	:deserialize [<value>] from=[arg]	Deserialize specified value/array from JSON format. Specify the format using **from=json**.	`:put [:deserialize from=json value="[\"a\",\"b\",\"c\"]"]` `a;b;c`

typeof	`:typeof <var>`	the return data type of variable	`:put [:typeof 4];`
rndnum	`:rndnum from= [num] to=[num]`	random number generator	`:put [:rndnum from=1 to=99];`
rndstr	`:rndstr from= [str] length=[num]`	Random string generator. **from** specifies characters to construct the string from and defaults to all ASCII letters and numerals. **length** specifies the length of the string to create and defaults to 16.	`:put [:rndstr from="abcdef%^&" length=33];`
set	`:set <var> [<value>]`	assign value to a declared variable.	`:global a; :set a true;`
terminal	:terminal	terminal related commands	
time	`:time <expression>`	return interval of time needed to execute the command	`:put [:time {:for i from=1 to=10 do={ :delay 100ms }}];`

timestamp	`:timestamp`	returns the time since epoch, where epoch is January 1, 1970 (Thursday), not counting leap seconds	`[admin@MikroTik] > :put [: timestamp]`
			`2735w21:41:43.481891543`
			`or`
			`[admin@MikroTik] > :put [: timestamp]`
			`2735w1d21:41:43.481891543`
			`with the day offset`

toarray	`:toarray <var>`	convert a variable to the array	
tobool	`:tobool <var>`	convert a variable to boolean	
toid	`:toid <var>`	convert a variable to internal ID	
toip	`:toip <var>`	convert a variable to IP address	
toip6	`:toip6 <var>`	convert a variable to IPv6 address	
tonum	`:tonum <var>`	convert a variable to an integer	
tostr	`:tostr <var>`	convert a variable to a string	

totime	`:totime <var>`	convert a variable to time	`[admin@191] > :put [ totime "1970-01-11 12:33" ]`
			`1w3d12:33:00`
			`[admin@191] > :put [ totime "1970-02-11" ]`
			`5w6d00:00:00`
			`[admin@191] > :put [ totime 3 ]`
			`00:00:03`
			`[admin@191] > :put [ totime "1000d" ]`
			`142w6d00:00:00`

tocrlf	`:tocrlf [<value>]`	converts line endings to CRLF	
tolf	`:tolf [<value>]`	converts line endings to LF	

Menu specific commands

Common commands

The following commands are available from most sub-menus:

Command	Syntax	Description

add	`add <param>=<value>. .<param>=<value>`	add new item
remove	`remove <id>`	remove selected item
enable	`enable <id>`	enable selected item
disable	`disable <id>`	disable selected item

set	`set <id> <param>=<value>. .<param>=<value>`	change selected items parameter, more than one parameter can be specified at the time. The parameter can be unset by specifying '!' before the parameter.
		
		Example:
		`/ip firewall filter add chain=blah action=accept protocol=tcp port=123 nth=4,2 print`
		`set 0 !port chain=blah2 !nth protocol=udp`

get	`get <id> <param>=<value>`	get the selected item's parameter value
print	`print <param><param>=[<value>]`	print menu items. Output depends on the print parameters specified. The most common print parameters are described here
export	`export [file=<value>]`	export configuration from the current menu and its sub-menus (if present). If the file parameter is specified output will be written to the file with the extension '.rsc', otherwise the output will be printed to the console. Exported commands can be imported by import command
edit	`edit <id> <param>`	edit selected items property in the built-in text editor
find	`find <expression>`	Returns list of internal numbers for items that are matched by given expression. For example: `:put [ /interface find name~"ether"]`

import

The import command is available from the root menu and is used to import configuration from files created by an export command or written manually by hand.

print parameters

Several parameters are available for print command:

Parameter	Description	Example
append		
as-value	print output as an array of parameters and its values	`:put [/ip address print as-value]`
brief	print brief description	
detail	print detailed description, the output is not as readable as brief output but may be useful to view all parameters	
count-only	print only count of menu items	
file	print output to a file	
follow	print all current entries and track new entries until ctrl-c is pressed, very useful when viewing log entries	`/log print follow`
follow-only	print and track only new entries until ctrl-c is pressed, very useful when viewing log entries	`/log print follow-only`
from	print parameters only from specified item	`/user print from=admin`

interval	continuously print output in a selected time interval, useful to track down changes where `follow` is not acceptable	`/interface print interval=2`
terse	show details in a compact and machine-friendly format	
value-list	show values single per line (good for parsing purposes)	
without-paging	If the output does not fit in the console screen then do not stop, print all information in one piece	
where	expressions followed by where parameters can be used to filter outmatched entries	`/ip route print where interface="ether1"`

More than one parameter can be specified at a time, for example, `/ip route print count-only interval=1 where interface="ether1"`

Loops and conditional statements

Loops

Command	Syntax	Description
do..while	`:do { <commands> } while=( <conditions> ); :while ( <conditions> ) do={ <commands> };`	execute commands until a given condition is met.
for	`:for <var> from=<int> to=<int> step=<int> do={ <commands> }`	execute commands over a given number of iterations
foreach	`:foreach <var> in=<array> do={ <commands> };`	execute commands for each element in a list

Conditional statement

Command	Syntax	Description
if	`:if (<condition>) do={<commands>} else={<commands>} <expression>`	If a given condition is `true` then execute commands in the `do` block, otherwise execute commands in the `else` block if specified.

Example:

```
{
        :local myBool true;
        :if ($myBool = false) do={ :put "value is false" } else={ :put "value is true" }
}
```

Functions

Scripting language does not allow you to create functions directly, however, you could use :parse command as a workaround.

Starting from v6.2 new syntax is added to easier define such functions and even pass parameters. It is also possible to return function value with **:return** co mmand.

See examples below:

```
#define function and run it
:global myFunc do={:put "hello from function"}
$myFunc

output:
hello from function

#pass arguments to the function
:global myFunc do={:put "arg a=$a"; :put "arg '1'=$1"}
$myFunc a="this is arg a value" "this is arg1 value"

output:
arg a=this is arg a value
arg '1'=this is arg1 value
```

Notice that there are two ways how to pass arguments:

- pass arg with a specific name ("a" in our example)
- pass value without arg name, in such case arg "1", "2" .. "n" is used.

Return example

```
:global myFunc do={ :return ($a + $b)}
:put [$myFunc a=6 b=2]

output:
8
```

You can even clone an existing script from the script environment and use it as a function.

```
#add script
/system script add name=myScript source=":put \"Hello $myVar !\""

:global myFunc [:parse [/system script get myScript source]]
$myFunc myVar=world

output:
Hello world !
```

If the function contains a defined global variable that names match the name of the passed parameter, then the globally defined variable is ignored, for compatibility with scripts written for older versions. This feature can change in future versions. **Avoid using parameters with the same name as global variables.**

For example:

```
:global my2 "123"

:global myFunc do={ :global my2; :put $my2; :set my2 "lala"; :put $my2 }
$myFunc my2=1234
:put "global value $my2"
```

The output will be:

```
1234
lala
global value 123
```

Nested function example

Note: to call another function its name needs to be declared (the same as for variables)

```
:global funcA do={ :return 5 }
:global funcB do={
        :global funcA;
        :return ([$funcA] + 4)
}
:put [$funcB]

Output:
9
```

Catch run-time errors

Starting from v6.2 scripting has the ability to catch run-time errors.

For example, the [code]:reslove[/code] command if failed will throw an error and break the script.

```
   [admin@MikroTik] > { :put [:resolve www.example.com]; :put "lala";}
   failure: dns name does not exist
```

Now we want to catch this error and proceed with our script:

```
:do {
        :put [:resolve www.example.com];
} on-error={ :put "resolver failed"};
:put "lala"

output:

resolver failed
lala
```

Operations with Arrays

Warning: Key name in the array contains any character other than a lowercase character, it should be put in quotes

For example:

```
[admin@ce0] > {:local a { "aX"=1 ; ay=2 }; :put ($a->"aX")}
1
```

Loop through keys and values

"foreach" command can be used to loop through keys and elements:

```
[admin@ce0] > :foreach k,v in={2; "aX"=1 ; y=2; 5} do={:put ("$k=$v")}

0=2
1=5
aX=1
y=2
```

If the "foreach" command is used with one argument, then the element value will be returned:

```
[admin@ce0] > :foreach k in={2; "aX"=1 ; y=2; 5} do={:put ("$k")}

2
5
1
2
```

Note: If the array element has a key then these elements are sorted in alphabetical order, elements without keys are moved before elements with keys and their order is not changed (see example above).

Change the value of a single array element

```
[admin@MikroTik] > :global a {x=1; y=2}
[admin@MikroTik] > :set ($a->"x") 5
[admin@MikroTik] > :environment print
a={x=5; y=2}
```

Script repository

Sub-menu level: `/system script`

Contains all user-created scripts. Scripts can be executed in several different ways:

- **on event** - scripts are executed automatically on some facility events (scheduler, netwatch, VRRP)
- **by another script** - running script within the script is allowed
- **manually** - from console executing a run command or in winbox

Note: Only scripts (including schedulers, netwatch, etc) with equal or higher permission rights can execute other scripts.

Property	Description
comment (*string*; Default:)	Descriptive comment for the script
dont-require-permissions (*yes* | *no*; Default: *no*)	Bypass permissions check when the script is being executed, useful when scripts are being executed from services that have limited permissions, such as Netwatch
name (*string*; Default: *"Script[num]"*)	name of the script
policy (*string*; Default: ftp,reboot,read,write,policy, test,password,sniff,sensitive,romon)	list of applicable policies: - **ftp** - can log on remotely via FTP and send and retrieve files from the router - **password** - change passwords - **policy** - manage user policies, add and remove user - **read** - can retrieve the configuration - **reboot** - can reboot the router - **sensitive** - allows changing "hide sensitive" parameter - **sniff** - can run sniffer, torch, etc - **test** - can run ping, traceroute, bandwidth test - **write** - can change the configuration Read more detailed policy descriptions here
source (*string*;)	Script source code

Read-only status properties:

Property	Description
last-started (*date*)	Date and time when the script was last invoked.
owner (*string*)	The user who created the script
run-count (*integer*)	Counter that counts how many times the script has been executed

Menu specific commands

Command	Description
run (*run [id|name]*)	Execute the specified script by ID or name

Environment

Sub-menu level:

- `/system script environment`
- `/environment`

Contains all user-defined variables and their assigned values.

```
[admin@MikroTik] > :global example;
[admin@MikroTik] > :set example 123
[admin@MikroTik] > /environment print
"example"=123
```

Read-only status properties:

Property	Description
name (*string*)	Variable name
user (*string*)	The user who defined variable
value ()	The value assigned to a variable

Job

Sub-menu level: `/system script job`

Contains a list of all currently running scripts.
Read-only status properties:

Property	Description
owner (*string*)	The user who is running the script
policy (*array*)	List of all policies applied to the script
started (*date*)	Local date and time when the script was started

See also

- Scripting Examples
- Manual: Scripting Tips and Tricks

Scripting examples

Introduction

This section contains some useful scripts and shows all available scripting features. Script examples used in this section were tested with the latest 3.x version.

Create a file

it is not possible to create a file directly, however, there is a workaround:

```
/file print file=myFile
/file set myFile.txt contents=""
```

Check if IP on the interface has changed

Sometimes provider gives dynamic IP addresses. This script will compare if a dynamic IP address is changed.

```
:global currentIP;

:local newIP [/ip address get [find interface="ether1"] address];

:if ($newIP != $currentIP) do={
    :put "ip address $currentIP changed to $newIP";
    :set currentIP $newIP;
}
```

Strip netmask

This script is useful if you need an IP address without a netmask (for example to use it in a firewall), but "`/ip address get [id] address`" returns the IP address and netmask.

```
:global ipaddress 10.1.101.1/24

:for i from=( [:len $ipaddress] - 1) to=0 do={
        :if ( [:pick $ipaddress $i] = "/") do={
                :put [:pick $ipaddress 0 $i]
        }
}
```

Another much more simple way:

```
:global ipaddress 10.1.101.1/24
:put [:pick $ipaddress 0 [:find $ipaddress "/"]]
```

Resolve host-name

Many users are asking features to use DNS names instead of IP addresses for radius servers, firewall rules, etc.

So here is an example of how to resolve the RADIUS server's IP.

Let's say we have the radius server configured:

```
/radius
add address=3.4.5.6 comment=myRad
```

And here is a script that will resolve the IP address, compare resolved IP with configured one, and replace it if not equal:

```
/system script add name="resolver" source= {

:local resolvedIP [:resolve "server.example.com"];
:local radiusID [/radius find comment="myRad"];
:local currentIP [/radius get $radiusID address];

:if ($resolvedIP != $currentIP) do={
   /radius set $radiusID address=$resolvedIP;
   /log info "radius ip updated";
}

}
```

Add this script to the scheduler to run for example every 5 minutes

```
/system scheduler add name=resolveRadiusIP on-event="resolver" interval=5m
```

Write simple queue stats in multiple files

Let's consider queue namings are "some text.1" so we can search queues by the last number right after the dot.

```
:local entriesPerFile 10;
:local currentQueue 0;
:local queuesInFile 0;
:local fileContent "";
#determine needed file count
:local numQueues [/queue simple print count-only] ;
:local fileCount ($numQueues / $entriesPerFile);
:if ( ($fileCount * $entriesPerFile) != $numQueues) do={
   :set fileCount ($fileCount + 1);
}

#remove old files
/file remove [find name~"stats"];

:put "fileCount=$fileCount";

:for i from=1 to=$fileCount do={
#create file
   /file print file="stats$i.txt";
#clear content
   /file set [find name="stats$i.txt"] contents="";

   :while ($queuesInFile < $entriesPerFile) do={
     :if ($currentQueue < $numQueues) do={
         :set currentQueue ($currentQueue +1);
         :put $currentQueue ;
         /queue simple
         :local internalID [find name~"\\.$currentQueue\$"];
         :put "internalID=$internalID";
         :set fileContent ($fileContent . [get $internalID target-address] . \
           " " . [get $internalID total-bytes] . "\r\n");
     }
     :set queuesInFile ($queuesInFile +1);

   }
   /file set "stats$i.txt" contents=$fileContent;
   :set fileContent "";
   :set queuesInFile 0;

}
```

Generate backup and send it by e-mail

This script generates a backup file and sends it to a specified e-mail address. The mail subject contains the router's name, current date, and time.

Note that the SMTP server must be configured before this script can be used. See /tool e-mail for configuration options.

```
/system backup save name=email_backup
/tool e-mail send file=email_backup.backup to="me@test.com" body="See attached file" \
   subject="$[/system identity get name] $[/system clock get time] $[/system clock get date] Backup")
```

⚠ The backup file contains sensitive information like passwords. So to get access to generated backup files, the script or scheduler must have a 'sensitive' policy.

Use string as a function

```
:global printA [:parse ":local A; :put \$A;" ];
$printA
```

Check bandwidth and add limitations

This script checks if the download on an interface is more than 512kbps if true then the queue is added to limit the speed to 256kbps.

```
:foreach i in=[/interface find] do={
    /interface monitor-traffic $i once do={
        :if ($"received-bits-per-second" > 0 ) do={
            :local tmpIP [/ip address get [/ip address find interface=$i] address] ;
#            :log warning $tmpIP ;
            :for j from=( [:len $tmpIP] - 1) to=0 do={
                :if ( [:pick $tmpIP $j] = "/") do={
                    /queue simple add name=$i max-limit=256000/256000 dst-address=[:pick $tmpIP 0 $j] ;
                }
            }
        }
    }
}
```

Block access to specific websites

This script is useful if you want to block certain websites but you don't want to use a web proxy.

This example looks at entries "Rapidshare" and "youtube" in the DNS cache and adds IPs to the address list named "restricted". Before you begin, you must set up a router to catch all DNS requests:

```
/ip firewall nat
add action=redirect chain=dstnat comment=DNS dst-port=53 protocol=tcp to-ports=53
add action=redirect chain=dstnat dst-port=53 protocol=udp to-ports=53
```

and add firewall

```
/ip firewall filter
add chain=forward dst-address-list=restricted action=drop
```

Now we can write a script and schedule it to run, let's say, every 30 seconds.

Script Code:

```
:foreach i in=[/ip dns cache find] do={
    :local bNew "true";
    :local cacheName [/ip dns cache all get $i name] ;
#    :put $cacheName;

    :if (([:find $cacheName "rapidshare"] >= 0) || ([:find $cacheName "youtube"] >= 0)) do={

        :local tmpAddress [/ip dns cache get $i address] ;
#        :put $tmpAddress;

# if address list is empty do not check
        :if ( [/ip firewall address-list find list="restricted" ] = "") do={
            :log info ("added entry: $[/ip dns cache get $i name] IP $tmpAddress");
            /ip firewall address-list add address=$tmpAddress list=restricted comment=$cacheName;
        } else={
            :foreach j in=[/ip firewall address-list find list="restricted"] do={
                :if ( [/ip firewall address-list get $j address] = $tmpAddress ) do={
                    :set bNew "false";
                }
            }
            :if ( $bNew = "true" ) do={
                :log info ("added entry: $[/ip dns cache get $i name] IP $tmpAddress");
                /ip firewall address-list add address=$tmpAddress list=restricted comment=$cacheName;
            }
        }
    }
}
```

Parse file to add ppp secrets

This script requires that entries inside the file are in the following format:

username,password,local_address,remote_address,profile,service

For example:

```
janis,123,1.1.1.1,2.2.2.1,ppp_profile,myService
juris,456,1.1.1.1,2.2.2.2,ppp_profile,myService
aija,678,1.1.1.1,2.2.2.3,ppp_profile,myService
```

```
:global content [/file get [/file find name=test.txt] contents] ;
:global contentLen [ :len $content ] ;

:global lineEnd 0;
:global line "";
:global lastEnd 0;


:do {
       :set lineEnd [:find $content "\r\n" $lastEnd ] ;
       :set line [:pick $content $lastEnd $lineEnd] ;
       :set lastEnd ( $lineEnd + 2 ) ;

       :local tmpArray [:toarray $line] ;
        :if ( [:pick $tmpArray 0] != "" ) do={
        :put $tmpArray;
         /ppp secret add name=[:pick $tmpArray 0] password=[:pick $tmpArray 1] \
             local-address=[:pick $tmpArray 2] remote-address=[:pick $tmpArray 3] \
             profile=[:pick $tmpArray 4] service=[:pick $tmpArray 5];
}
} while ($lineEnd < $contentLen)
```

Detect new log entry

This script is checking if a new log entry is added to a particular buffer.

In this example we will use PPPoE logs:

```
/system logging action
add name="pppoe"
/system logging
add action=pppoe topics=pppoe,info,!ppp,!debug
```

Log buffer will look similar to this one:

```
[admin@mainGW] > /log print where buffer=pppoe
13:11:08 pppoe,info PPPoE connection established from 00:0C:42:04:4C:EE
```

Now we can write a script to detect if a new entry is added.

```
:global lastTime;

:global currentBuf [ :toarray [ /log find buffer=pppoe  ] ] ;
:global currentLineCount [ :len $currentBuf ] ;
:global currentTime [ :totime [/log get [ :pick $currentBuf ($currentLineCount -1) ] time   ] ];

:global message "";

:if ( $lastTime = "" ) do={
        :set lastTime $currentTime ;
        :set message [/log get [ :pick $currentBuf ($currentLineCount-1) ] message];

} else={
        :if ( $lastTime < $currentTime ) do={
                :set lastTime $currentTime ;
                :set message [/log get [ :pick $currentBuf ($currentLineCount-1) ] message];
        }
}
```

After a new entry is detected, it is saved in the "message" variable, which you can use later to parse log messages, for example, to get the PPPoE client's mac addresses.

Allow use of ntp.org pool service for NTP

This script resolves the hostnames of two NTP servers, compares the result with the current NTP settings, and changes the addresses if they're different. This script is required as RouterOS does not allow hostnames to be used in the NTP configuration. Two scripts are used. The first defines some system variables which are used in other scripts and the second does the grunt work:

```
# System configuration script - "GlobalVars"

:put "Setting system globals";

# System name
:global SYSname [/system identity get name];

# E-mail address to send notifications to
:global SYSsendemail "mail@my.address";

# E-mail address to send notifications from
:global SYSmyemail "routeros@my.address";

# Mail server to use
:global SYSemailserver "1.2.3.4";

# NTP pools to use (check www.pool.ntp.org)
:global SYSntpa "0.uk.pool.ntp.org";
:global SYSntpb "1.uk.pool.ntp.org";
```

```
# Check and set NTP servers - "setntppool"

# We need to use the following globals which must be defined here even
# though they are also defined in the script we call to set them.
:global SYSname;
:global SYSsendemail;
:global SYSmyemail;
:global SYSmyname;
:global SYSemailserver;
:global SYSntpa;
:global SYSntpb;

# Load the global variables with the system defaults
/system script run GlobalVars

# Resolve the two ntp pool hostnames
:local ntpipa [:resolve $SYSntpa];
:local ntpipb [:resolve $SYSntpb];

# Get the current settings
:local ntpcura [/system ntp client get primary-ntp];
:local ntpcurb [/system ntp client get secondary-ntp];

# Define a variable so we know if anything's changed.
:local changea 0;
:local changeb 0;

# Debug output
:put ("Old: " . $ntpcura . " New: " . $ntpipa);
:put ("Old: " . $ntpcurb . " New: " . $ntpipb);

# Change primary if required
:if ($ntpipa != $ntpcura) do={
    :put "Changing primary NTP";
    /system ntp client set primary-ntp="$ntpipa";
    :set changea 1;
    }

# Change secondary if required
:if ($ntpipb != $ntpcurb) do={
    :put "Changing secondary NTP";
    /system ntp client set secondary-ntp="$ntpipb";
    :set changeb 1;
    }

# If we've made a change, send an e-mail to say so.
:if (($changea = 1) || ($changeb = 1)) do={
    :put "Sending e-mail.";
    /tool e-mail send \
        to=$SYSsendemail \
        subject=($SYSname . " NTP change") \
        from=$SYSmyemail \
        server=$SYSemailserver \
        body=("Your NTP servers have just been changed:\n\nPrimary:\nOld: " . $ntpcura . "\nNew: " \
          . $ntpipa . "\n\nSecondary\nOld: " . $ntpcurb . "\nNew: " . $ntpipb);
    }
```

Scheduler entry:

```
/system scheduler add \
  comment="Check and set NTP servers" \
  disabled=no \
  interval=12h \
  name=CheckNTPServers \
  on-event=setntppool \
  policy=read,write,test \
  start-date=jan/01/1970 \
  start-time=16:00:00
```

Other scripts

- Dynamic_DNS_Update_Script_for_EveryDNS
- Dynamic_DNS_Update_Script_for_ChangeIP.com
- UPS Script

System Information and Utilities

In This Section:

Clock

Introduction

RouterOS uses data from the TZ database, Most of the time zones from this database are included, and have the same names. Because local time on the router is used mostly for timestamping and time-dependent configuration, and not for historical date calculations, time zone information about past years is not included. Currently, only information starting from 2005 is included.

Following settings are available in the **/system clock** console path and in the "Time" tab of the "System > Clock" WinBox window.

Startup date and time is **jan/02/1970 00:00:00** [+|-]gmt-offset.

Properties

Property	Description
time (*HH:MM:SS);*	where *HH* - hour 00..24, *MM* - minutes 00..59, *SS* - seconds 00..59).
date (*mmm/DD/YYYY);*	where *mmm* - month, one of *jan*, *feb*, *mar*, *apr*, *may*, *jun*, *jul*, *aug*, *sep*, *oct*, *nov*, *dec*, *DD* - date, 00..31, *YYYY* - year, 1970..2037): **date** and **time** show current local time on the router. These values can be adjusted using the **set** command. Local time cannot, however, be exported, and is not stored with the rest of the configuration.
time-zone-name (*manual*, or name of time zone; default value: *manual*);	Name of the time zone. As most of the text values in RouterOS, this value is case sensitive. Special value *manual* applies manually configured GMT offset, which by default is *00:00* with no daylight saving time.
time-zone-autodetect (*yes* or *no*; default: yes);	Feature available from v6.27. If enabled, the time zone will be set automatically.

⚠ Time-zone-autodetect by default is enabled on new RouterOS installation and after configuration reset. The time zone is detected depending on the router's public IP address and our Cloud servers database. Since RouterOS v6.43 your device will use cloud2.mikrotik.com to communicate with MikroTik's Cloud server. Older versions will use cloud.mikrotik.com to communicate with the MikroTik's Cloud server.

Configuration

Active time zone information

- **dst-active** (*yes* or *no*>; read-only property): This property has the value *yes* while daylight saving time of the current time zone is active.
- **gmt-offset** ([*+*|-]*HH:MM* - offset in hours and minutes; read-only property): This is the current value of GMT offset used by the system, after applying base time zone offset and active daylight saving time offset.

Manual time zone configuration

These settings are available in **/system clock manual** console path and in the "Manual Time Zone" tab of the "System > Clock" WinBox window. These settings have an effect only when **time-zone-name**=*manual*. It is only possible to manually configure single daylight saving time period.

- **time-zone**, **dst-delta** ([*+*|-]*HH:MM* - time offset in hours and minutes, leading plus sign is optional; default value: *+00:00*) : While DST is not active use GMT offset **time-zone**. While DST is active use GMT offset **time-zone** + **dst-delta**.
- **dst-start**, **dst-end** (*mmm/DD/YYYY HH:MM: SS* - date and time, either date or time can be omitted in the **set** command; default value: *jan/01/1970 00:00:00*): Local time when DST starts and ends.

Device-mode

The **device-mode** is a feature which sets specific limitations on a device, or limits access to specific configuration options.
There are two available modes: *enterprise* and *home*. By default, all devices use the mode *enterprise,* which allows all functionality except *container*. The h ome mode disables the following features: *scheduler, socks, fetch, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, email, zerotier, container.*

```
[admin@MikroTik] > system/device-mode/print
  mode: enterprise
```

The device mode can be changed by the user, but remote access to the device is not enough to change it. After changing the device-mode, you need to confirm it, by pressing a button on the device itself, or perform a "cold reboot" - that is, unplug the power.

```
[admin@MikroTik] > system/device-mode/update mode=home
  update: please activate by turning power off or pressing reset or mode button
          in 5m00s
-- [Q quit|D dump|C-z pause]
```

If no power off or button press is performed within the specified time, the mode change is canceled. If another update command is run in parallel, both will be canceled.

The following commands are available in the **system/device-mode/** menu:

Property	Description
get	Returns value that you can assign to variable or print on the screen.
print	Shows the active mode and its properties.
update	Applies changes to the specified properties, see below.

List of available properties

Property	Description
container, fetch, scheduler, traffic-gen, ipsec, pptp, smb, l2tp, proxy, sniffer, zerotier, bandwidth-test, email, hotspot, romon, socks. (*yes* / *no*; Default: **yes**, for enterprise mode)	The list of available features, which can be controlled with the **device-mode** option.
activation-timeout (default: **5m**);	The reset button or power off activation timeout can be set in range 00:00:10 .. 1d00:00:00. If the reset button is not pressed (or cold reboot is not performed) during this interval, the update will be canceled.
flagging-enabled (*yes* / *no*; Default: **yes**)	Enable or disable the *flagged* status. See below for a detailed description.
flagged (*yes* / *no*; Default: **no**)	RouterOS employs various mechanisms to detect tampering with it's system files. If the system has detected unauthorized access to RouterOS, the status "flagged" is set to yes. If "flagged" is set to yes, for your safety, certain limitations are put in place. See below chapter for more information.
mode: (home, enterprise; default: **enterprise**);	Allows choosing from available modes that will limit device functionality. In the future, various modes could be added. By default, **enterprise** mode allows all options except container. So to use the **container** feature, you will need to turn it on by performing a device-mode update. By default, **home** mode disables the following features: **scheduler, socks, fetch, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, email, zerotier, container.**

More specific control over the available features is possible. Each of the features controlled by device-mode can be specifically turned on or off, for example:

```
[admin@MikroTik] > system/device-mode/update mode=home email=yes
[admin@MikroTik] > system/device-mode/update mode=enterprise zerotier=no
```

If the update command specifies any of the mode parameters, this update replaces the entire device-mode configuration. In this case, all "per-feature" settings will be lost, except those specified with this command. For instance:

```
[admin@MikroTik] > system/device-mode/update mode=home email=yes fetch=yes
[admin@MikroTik] > system/device-mode/print
   mode: home
  fetch: yes
  email: yes
[admin@MikroTik] > system/device-mode/update mode=enterprise sniffer=no
-- reboot --
[admin@MikroTik] > system/device-mode/print
     mode: enterprise
  sniffer: no
```

We see that fetch = yes and email = yes is missing, as they were overriden with the mode change. However, specifying only "per-feature" settings will change only those:

```
[admin@MikroTik] > system/device-mode/update hotspot=no
-- reboot --
[admin@MikroTik] > system/device-mode/print
     mode: enterprise
  sniffer: no
  hotspot: no
```

If the feature is disabled, an error message is displayed for interactive commands:

```
[admin@MikroTik] > system/device-mode/print
     mode: enterprise
  sniffer: no
  hotspot: no
[admin@MikroTik] > tool/sniffer/quick
failure: not allowed by device-mode
```

However, it is possible to add the configuration to a disabled feature, but there will be a comment showing the disabled feature in the device-mode:

```
[admin@MikroTik] > ip hotspot/add interface=ether1
[admin@MikroTik] > ip hotspot/print
Flags: X, S - HTTPS
Columns: NAME, INTERFACE, PROFILE, IDLE-TIMEOUT
#   NAME      INTERFACE  PROFILE  IDLE-TIMEOUT
;;; inactivated, not allowed by device-mode
0 X hotspot1  ether1     default  5m
```

Flagged status

Along with the device-mode feature, RouterOS now can analyze the whole configuration at system startup, to determine if there are any signs of unauthorized access to your router. If suspicious configuration is detected, the suspicious configuration will be disabled and the **flagged** parameter will be set to "yes". The device has now a Flagged state and enforces certain limitations.

```
[admin@MikroTik] > system/device-mode/print
     mode: enterprise
  flagged: yes
  sniffer: no
  hotspot: no
```

If the system has this flagged status, the current configuration works, but it is not possible to perform the following actions:

bandwidth-test, traffic-generator, sniffer, as well as configuration actions that enable or create new configuration entries (it will still be possible to disable or delete them) for the following programs: *system scheduler, SOCKS proxy, pptp, l2tp, ipsec, proxy, smb*.

When performing the aforementioned actions while the router has the flagged state, you will receive an error message:

```
[admin@MikroTik] > /tool sniffer/quick
failure: configuration flagged, check all router configuration for unauthorized changes and update device-mode
[admin@MikroTik] > /int l2tp-client/add connect-to=1.1.1.1 user=user
failure: configuration flagged, check all router configuration for unauthorized changes and update device-mode
```

To exit the flagged state, you must perform the command "/system/device-mode/update flagged=no". The system will ask to either press a button, or issue a hard reboot (cut power physically or do a hard reboot of the virtual machine).

Important! Although the system has disabled any malicious looking rules, which triggered the flagged state, it is crucial to inspect all of your configuration for other unknown things, before exiting the flagged state. If your system has been flagged, assume that your system has been compromised and do a full audit of all settings before re-enabling the system for use. After completing the audit, change all the system passwords and upgrade to the latest RouterOS version.

E-mail

An E-mail tool is a utility that allows sending e-mails from the router. The tool can be used to send regular configuration backups and exports to a network administrator.

Email tool uses only plain authentication and TLS encryption. Other methods are not supported.

Properties

```
/tool e-mail
```

This submenu allows setting SMTP server that will be used.

Property	Description
address (*IP/IPv6 address*; Default: **0.0.0.0**)	SMTP server's IP address.
from (*string*; Default: **<>**)	Name or email address that will be shown as a receiver.
password (*string*; Default: **""**)	Password used for authenticating to an SMTP server.
port (*integer[0..65535]*; Default: **25**)	SMTP server's port.
tls (*no|yes|starttls*; Default: **no**)	Whether to use TLS encryption: • yes - sends STARTTLS and drops the session if TLS is not available on the server • no - do not send STARTTLS • starttls - sends STARTTLS and continue without TLS if a server responds that TLS is not available
user (*string*; Default: **""**)	The username used for authenticating to an SMTP server.
vrf (*VRF name*; default value: **main**)	Set VRF on which service is creating outgoing connections.

⚠ **Note:** All server's configurations (if specified) can be overridden by send command.

Sending Email

```
/tool e-mail send
```

Send command takes the following parameters:

Property	Description
body (*string*; Default:)	The actual body of the email message
cc (*string*; Default:)	Send a copy to listed recipients. Multiple addresses allowed, use "," to separate entries
file (*File[,File]*; Default:)	List of the file names that will be attached to the mail separated by a comma.

from (*string*; Default:)	Name or email address which will appear as the sender. If a not specified value from the server's configuration is used.
password (*string*; Default:)	Password used to authenticate to an SMTP server. If a not specified value from the server's configuration is used.
port (*integer[0..65535]*; Default:)	Port of SMTP server. If not specified, a value from the server's configuration is used.
server (*IP/IPv6 address*; Default:)	Ip or IPv6 address of SMTP server. If not specified, a value from the server's configuration is used.
tls (*yes|no|starttls*; Default: **no**)	Whether to use TLS encryption: • yes - sends STARTTLS and drops the session if TLS is not available on the server • no - do not send STARTTLS • starttls - sends STARTTLS and continue without TLS if a server responds that TLS is not available
subject (*string*; Default:)	The subject of the message.
to (*string*; Default:)	Destination email address. Single address allowed.
user (*string*; Default:)	The username used to authenticate to an SMTP server. If not specified, a value from the server's configuration is used.

Basic examples

This example will show how to send an email with configuration export every 24hours.

1. Configure SMTP server

```
[admin@MikroTik] /tool e-mail> set server=10.1.1.1 port=25 from="router@mydomain.com"
```

2. Add a new script named "export-send":

```
/export file=export
/tool e-mail send to="config@mydomain.com" subject="$[/system identity get name] export" \
body="$[/system clock get date] configuration file" file=export.rsc
```

3. Add scheduler to run our script:

```
/system scheduler add on-event="export-send" start-time=00:00:00 interval=24h
```

Send e-mail to a server using TLS/SSL encryption. For example, Google mail requires that.

After the Google mail added **a new security policy** that **does not allow 3d-party devices to authenticate using** your standard **Gmail password** → you need to generate a 16-digit passcode ("App" password) and use it instead of your Gmail password. To configure this, navigate to the "**Security>Signing in to Google**" section settings and:

- **Enable 2-Step Verification;**
- **Generate an App password**.

Use the newly generated App password in the "set password=**mypassword**" setting shown below.

1. configure a client to connect to the correct server:

```
/tool e-mail
set address=smtp.gmail.com
set port=465
set tls=yes
set from=myuser@gmail.com
set user=myuser
set password=mypassword
```

2. send e-mail using send command:

```
/tool e-mail send to=myuser@anotherdomain.com subject="email test" body="email test"
```

Fetch

Summary

Fetch is one of the console tools in MikroTik RouterOS. It is used to copy files to/from a network device via HTTP, HTTPS, FTP or SFTP. It can also be used to send POST/GET requests and send any kind of data to a remote server. In HTTPS mode by default, no certificate checks are made, setting *check-certificate* to *yes* enables trust chain validation from the local certificate store (can be used only in HTTPS mode).

Properties

Property	Description
address (*string*; Default:)	IP address of the device to copy file from.
as-value (*set* | *not-set*; Default: **not-set**)	Store the output in a variable, should be used with the output property.
ascii (*yes* | *no*; Default: **no**)	Can be used with FTP and TFTP
certificate (*string*; Default:)	Certificate that should be used for host verification. Can be used only in HTTPS mode.
check-certificate (*yes* | *yes-without-crl* | *no*; Default: **no**)	Enables trust chain validation from local certificate store. *yes-without-crl, validates a certificate, not performing CRL check (certificate revocation list).* Can be used only in HTTPS mode.
dst-path (*string*; Default:)	Destination filename and path.
duration (*time;* Default:)	Time how long fetch should run.
host (*string*; Default:)	A domain name or virtual domain name (if used on a website, from which you want to copy information). For example, `address=wiki.mikrotik.com host=forum.mikrotik.com` In this example the resolved ip address is the same (66.228.113.27), but hosts are different.
http-auth-scheme (*basic|digest*; Default: **basic**)	HTTP authentication scheme
http-method (*delete|get|head|post|put|patch*; Default: **get**)	HTTP method to use
http-data (*string*; Default:)	The data, that is going to be sent, when using PUT or POST methods. Data limit is 64Kb.
http-header-field (*string*; Default: ***empty***)	List of all header fields and their values, in the form of `http-header-field=h1:fff,h2:yyy`
http-content-encoding (*deflate|gzip*; Default: ***empty***)	Encodes the payload using **gzip** or **deflate** compression and adds a corresponding Content-Encoding header. Usable for HTTP POST and PUT only.
keep-result (*yes* | *no*; Default: **yes**)	If yes, creates an input file.
mode (*ftp|http|https|sftp|tftp*; Default: **http**)	Choose the protocol of connection - http, https , ftp, sftp or tftp.

output (*none*|*file*|*user*|*user-with-headers*; Default: **file**)	Sets where to store the downloaded data. • `none` - do not store downloaded data • `file` - store downloaded data in a file • `user` - store downloaded data in the data variable (variable limit is 64Kb) • user-with-headers - store downloaded data and headers in the data variable (variable limit is 64Kb (20Kb for downloaded data, 44Kb for headers))
password (*string*; Default: **anonymous**)	Password, which is needed for authentication to the remote device.
port (*integer*; Default:)	Connection port.
src-address (ip address; Default:)	Source address that is used to establish connection. Can be used only HTTP/S and SFTP modes.
src-path (*string*; Default:)	Title of the remote file you need to copy.
upload (*yes* | *no*; Default: **no**)	Only (S)FTP modes support upload. If enabled then fetch will be used to upload files to a remote server. Requires *src-path* and *dst-path* parameters to be set.
url (*string*; Default:)	URL pointing to file. Can be used instead of **address** and **src-path** parameters.
user (*string*; Default: **anonymous**)	Username, which is needed for authentication to the remote device.

Configuration Examples

The following example shows how to copy the file with filename "conf.rsc" from a device with ip address 192.168.88.2 by FTP protocol and save it as file with filename "123.rsc". User and password are needed to login into the device.

```
[admin@MikroTik] /tool> fetch address=192.168.88.2 src-path=conf.rsc \
user=admin mode=ftp password=123 dst-path=123.rsc port=21 \
host="" keep-result=yes
```

Example to upload file to another router:

```
[admin@MikroTik] /tool> fetch address=192.168.88.2 src-path=conf.rsc \
user=admin mode=ftp password=123 dst-path=123.rsc upload=yes
```

Another file download example that demonstrates the usage of url property.

```
[admin@MikroTik] /> /tool fetch url="https://www.mikrotik.com/img/netaddresses2.pdf" mode=http
  status: finished

[admin@test_host] /> /file print
 # NAME                    TYPE                SIZE                CREATION-TIME
 ...
 5 netaddresses2.pdf       .pdf file           11547               jun/01/2010 11:59:51
```

Sending information to a remote host

It is possible to use an HTTP POST request to send information to a remote server, that is prepared to accept it. In the following example, we send geographic coordinates to a PHP page:

```
/tool/fetch http-method=post http-header-field="Content-Type:application/json" http-data="{\"lat\":\"56.12\",\"
lon\":\"25.12\"}" url="https://testserver.lv/index.php"
```

In this example, the data is uploaded as a file. Important note, since variable data comes from a file, a file can only be in size up to 4KB. This is a limitation of RouterOS variables.

```
/export file=export.rsc

:global data [/file get [/file find name=export.rsc] contents];
:global $url "https://prod-51.westeurope.logic.azure.com:443/workflows/blabla/triggers/manual/paths/invoke....";

/tool fetch mode=https http-method=put http-data=$data url=$url
```

Return value to a variable

It is possible to save the result of the fetch command to a variable. For example, it is possible to trigger a certain action based on the result that an HTTP page returns. You can find a very simple example below that disables **ether2** whenever a PHP page returns "0":

```
{
    :local result [/tool fetch url=https://10.0.0.1/disable_ether2.php as-value output=user];
    :if ($result->"status" = "finished") do={
        :if ($result->"data" = "0") do={
            /interface ethernet set ether2 disabled=yes;
        } else={
            /interface ethernet set ether2 disabled=no;
        }
    }
}
```

Files & Backups

Files

File menu shows all userspace files on the router. It is possible to see and edit file content or delete file. File creation is possible starting from RouterOS 7.9 beta2. If RouterOS ".npk" package is uploaded, the file menu will also show package-specific information, for example, architecture, build date and time, etc.

```
[admin@MikroTik] > file print detail
 0 name="routeros-mipsbe-6.45.7.npk" type="package" size=11.5MiB creation-time=oct/29/2019 11:36:15
   package-name="routeros-mipsbe" package-version="6.45.7" package-build-time=oct/24/2019 08:44:35
   package-architecture="mips"

 1 name="flash" type="directory" creation-time=jan/01/1970 02:00:03

 2 name="flash/skins" type="directory" creation-time=jan/01/1970 02:00:04

 3 name="flash/rw" type="directory" creation-time=sep/06/2019 14:01:16

 4 name="flash/rw/pckg" type="directory" creation-time=sep/06/2019 14:01:16
```

To create new file (command added in RouterOS 7.9beta2):

```
[admin@MikroTik] > file add name=lala
```

If the device has a directory named **"flash"** in its file list, then files which you want to be kept after the system reboot/power cycle must be stored within it. As anything outside of it is kept within a RAM disk and will be lost upon reboot. This does not include .npk upgrade files as they will be applied by the upgrade process before the system discards the RAM drive content.

For multi core devices with a NAND flash memory (e.g. CCR series routers, RB4011iGS), RouterOS uses a write-back that will cache file changes into RAM memory instead of writing them straight away into flash media. The file changes will be stored on the flash when it is absolutely necessary, the writing can be delayed by up to 40 seconds. This helps to reduce CPU cycles which results in better performance. However, this can cause empty or zero-length files when a device experiences a sudden power loss, because files were not fully saved on a flash.

Properties

Property	Description
contents (*string*; Default:)	The actual content of the file
creation-time (*time*)	A time when the file was created
name (*string*)	Name of the file
package-architecture (*string*)	Architecture that package is built for. Applies only to RouterOS ".npk" files
package-built-time (*string*)	A time when the package was built. Applies only to RouterOS ".npk" files
package-name (*string*)	Name of the installable package that. Applies only to RouterOS ".npk" files
package-version (*string*)	A version of the installable package that. Applies only to RouterOS ".npk" files
size (*integer*)	File size in bytes
file type (*string*)	Type of the file. For folders, the file type is the *directory*

Backups

RouterOS backup feature allows you to save the current device's configuration, which then can be re-applied on the same or a different identical model. This is very useful since it allows you to effortlessly restore the device's configurations or to re-apply the same configuration on a backup device. The system's backup file also contains the device's MAC addresses, which are also restored when the backup file is loaded.

If The Dude and user-manager are installed on the router, then the system backup will not contain configuration from these services. Therefore additional care should be taken to save a configuration from these services, for example, configuration export.

System's backups contain sensitive information about your device and its configuration, always consider encrypting the backup file and keeping the backup file in a safe place.

To save a backup configure the following:

```
[admin@MikroTik] >  system backup save name=/flash/backup1 password=StrongPass encryption=aes-sha256
Saving system configuration
Configuration backup saved
```

Note that we use "/flash/ "before the actual backup name on the devices with flash memory. As stated above, backups saved outside the flash folder will be deleted after a reboot or power cycle:

```
[admin@MikroTik] > system backup save name=backup2 password=StrongPass encryption=aes-sha256
Saving system configuration
Configuration backup saved
[admin@MikroTik] > file print detail
 0 name="flash" type="directory" creation-time=jan/01/1970 02:00:03

 1 name="flash/skins" type="directory" creation-time=jan/01/1970 02:00:04

 2 name="flash/rw" type="directory" creation-time=sep/06/2019 14:01:16

 3 name="flash/rw/pckg" type="directory" creation-time=sep/06/2019 14:01:16

 4 name="backup2.backup" type="backup" size=22.4KiB creation-time=oct/29/2019 11:40:33

 5 name="flash/backup1.backup" type="backup" size=22.4KiB creation-time=oct/29/2019 11:40:11
```

To load a backup, simply configure the following:

```
[admin@MikroTik] > system backup load name=/flash/backup1 password=StrongPass
Restore and reboot? [y/N]:
Y
```

Identity

Overview

Setting the System's Identity provides a unique identifying name for when the system identifies itself to other routers in the network and when accessing services such as DHCP, Neighbour Discovery, and default wireless SSID. The default system Identity is set to 'MikroTik'.

System Identity has a 64 maximum character length

Configuration

To set system identity in RouterOS:

```
[admin@MikroTik] > /system identity set name=New_Identity
[admin@New_Identity] >
```

The current System Identity is always displayed after the logged-in account name and with the print command:

```
[admin@New_Identity] /system identity>print
name: New_Identity
[admin@New_Identity] /system identity>
```

SNMP

It is also possible to change the router system identity by SNMP set command:

```
snmpset -c public -v 1 192.168.0.0 1.3.6.1.2.1.1.5.0 s New_Identity
```

snmpset - Linux based SNMP application used for SNMP SET requests to set information on a network entity;

- *public* - router's community name;
- *192.168.0.0* - IP address of the router;
- *1.3.6.1.2.1.1.5.0* - SNMP value for router's identity;

Interface Lists

Summary

Allows defining a set of interfaces for easier interface management in the different interface-based configuration sections such as Neighbor Discovery, Firewall, Bridge, and Internet Detect.

Lists

Sub-menu: `/interface list`

This menu contains information about all interface lists available on the router. There are four predefined lists - `all` (contains all interfaces), `none` (contains no interfaces), `dynamic` (contains dynamic interfaces), and `static` (contains static interfaces). It is also possible to create additional interface lists.

> Dynamic interfaces are interfaces that have a "dynamic" flag. Any interface that doesn't have a dynamic flag will be part of the `static` interface list.

Property	Description
name (*string*)	Name of the interface list
include (*string*)	Defines interface list which members are included in the list. It is possible to add multiple lists separated by commas
exclude (*string*)	Defines interface list which members are excluded from the list. It is possible to add multiple lists separated by commas

Members are added to the interface list in the following order:

1. include members are added to the interface list
2. exclude members are removed from the list
3. Statically configured members are added to the list

Members

Sub-menu: `/interface list member`

This sub-menu contains information about statically configured interface members to each interface list. Note that dynamically added interfaces by include and exclude statements are not represented in this sub-menu.

Property	Description
interface (*string*)	Name of the interface
list (*string*)	Name of the interface list

> Care must be taken when working with bridges and lists. Adding a bridge as a member is not the same as adding all its ports! And adding all slave ports as members is not the same as adding the bridge itself. This can particularly impact functionality of neighbor discovery.

Neighbor discovery

Summary

Neighbor Discovery protocols allow us to find devices compatible with MNDP (MikroTik Neighbor Discovery Protocol), CDP (Cisco Discovery Protocol), or LLDP (Link Layer Discovery Protocol) in the Layer2 broadcast domain. It can be used to map out your network.

Neighbor list

The neighbor list shows all discovered neighbors in the Layer2 broadcast domain. It shows to which interface neighbor is connected, its IP/MAC addresses, and other related parameters. The list is read-only, an example of a neighbor list is provided below:

```
[admin@MikroTik] /ip neighbor print
 # INTERFACE ADDRESS         MAC-ADDRESS       IDENTITY   VERSION    BOARD
 0 ether13   192.168.33.2    00:0C:42:00:38:9F MikroTik   5.99       RB1100AHx2
 1 ether11   1.1.1.4         00:0C:42:40:94:25 test-host  5.8        RB1000
 2 Local     10.0.11.203     00:02:B9:3E:AD:E0 c2611-r1   Cisco I...
 3 Local     10.0.11.47      00:0C:42:84:25:BA 11.47-750  5.7        RB750
 4 Local     10.0.11.254     00:0C:42:70:04:83 tsys-sw1   5.8        RB750G
 5 Local     10.0.11.202     00:17:5A:90:66:08 c7200      Cisco I...
```

Sub-menu: `/ip neighbor`

Property	Description
address (*IP*)	The highest IP address configured on a discovered device
address6 (*IPv6*)	IPv6 address configured on a discovered device
age (*time*)	Time interval since last discovery packet
discovered-by (*cdp|lldp|mndp*)	Shows the list of protocols the neighbor has been discovered by. The property is available since RouterOS version 7.7.
board (*string*)	RouterBoard model. Displayed only to devices with installed RouterOS
identity (*string*)	Configured system identity
interface (*string*)	Interface name to which discovered device is connected
interface-name (*string*)	Interface name on the neighbor device connected to the L2 broadcast domain. Applies to CDP.
ipv6 (*yes | no*)	Shows whether the device has IPv6 enabled.
mac-address (*MAC*)	Mac address of the remote device. Can be used to connect with mac-telnet.
platform (*string*)	Name of the platform. For example "MikroTik", "cisco", etc.
software-id (*string*)	RouterOS software ID on a remote device. Applies only to devices installed with RouterOS.
system-caps (*string*)	System capabilities reported by the Link-Layer Discovery Protocol (LLDP).
system-caps-enabled (*string*)	Enabled system capabilities reported by the Link-Layer Discovery Protocol (LLDP).
unpack (*none|simple|uncompressed-headers|uncompressed-all*)	Shows the discovery packet compression type.

uptime (*time*)	Uptime of remote device. Shown only to devices installed with RouterOS.
version (*string*)	Version number of installed software on a remote device

⚠ Starting from RouterOS v6.45, the number of neighbor entries are limited to (total RAM in megabytes)*16 per interface to avoid memory exhaustion.

Discovery configuration

It is possible to change whether an interface participates in neighbor discovery or not using an Interface list. If the interface is included in the discovery interface list, it will send out basic information about the system and process received discovery packets broadcasted in the Layer2 network. Removing an interface from the interface list will disable both the discovery of neighbors on this interface and also the possibility of discovering this device itself on that interface.

```
/ip neighbor discovery-settings
```

Property	Description
discover-interface-list (*string*; Default: **static**)	Interface list on which members the discovery protocol will run on
lldp-mac-phy-config (*yes* | *no*; Default: **no**)	Whether to send MAC/PHY Configuration/Status TLV in LLDP, which indicates the interface capabilities, current setting of the duplex status, bit rate, and auto-negotiation. Only applies to the Ethernet interfaces. While TLV is optional in LLDP, it is mandatory when sending LLDP-MED, meaning this TLV will be included when necessary even though the property is configured as disabled.
lldp-max-frame-size (*yes* | *no*; Default: **no**)	Whether to send Maximum Frame Size TLV in LLDP, which indicates the maximum frame size capability of the interface in bytes (`l2mtu` + 18). Only applies to the Ethernet interfaces.
lldp-poe-power (*yes* | *no*; Default: **yes**)	Whether to send IEEE 802.3 Organizationally Specific Power Via MDI TLV in LLDP, which allows an exchange of information between a PSE and a PD. Only applies to the Ethernet interfaces that support PoE-Out. The sending of LLDP-MED Organizationally Specific Extended Power via MDI TLV is not configurable, and it is included in outgoing LLDP-MED packets when necessary. The setting is available since RouterOS version 7.15, and it replaces PoE-out port `poe-lldp-enabled` setting.
lldp-med-net-policy-vlan (*integer 0..4094*; Default: **disabled**)	Advertised VLAN ID for LLDP-MED Network Policy TLV. This allows assigning a VLAN ID for LLDP-MED capable devices, such as VoIP phones. The TLV will only be added to interfaces where LLDP-MED capable devices are discovered. Other TLV values are predefined and cannot be changed: • Application Type - Voice • VLAN Type - Tagged • L2 Priority - 0 • DSCP Priority - 0 When used together with the bridge interface, the (R/M)STP protocol should be enabled with `protocol-mode` setting. Additionally, other neighbor discovery protocols (e.g. CDP) should be excluded using `protocol` setting to avoid LLDP-MED misconfiguration.
mode (*rx-only* | *tx-only* | *tx-and-rx*; Default: **tx-and-rx**)	Selects the neighbor discovery packet sending and receiving mode. The setting is available since RouterOS version 7.7.
protocol (*cdp* | *lldp* | *mndp*; Default: **cdp,lldp,mndp**)	List of used discovery protocols

Since RouterOS v6.44, neighbor discovery is working on individual slave interfaces. Whenever a master interface (e.g. bonding or bridge) is included in the discovery interface list, all its slave interfaces will automatically participate in neighbor discovery. It is possible to allow neighbor discovery only to some slave interfaces. To do that, include the particular slave interface in the list and make sure that the master interface is not included.

```
/interface bonding
add name=bond1 slaves=ether5,ether6
/interface list
add name=only-ether5
/interface list member
add interface=ether5 list=only-ether5
/ip neighbor discovery-settings
set discover-interface-list=only-ether5
```

Now the neighbor list shows a master interface and actual slave interface on which a discovery message was received.

```
[admin@R2] > ip neighbor print
 # INTERFACE ADDRESS                                         MAC-ADDRESS       IDENTITY   VERSION
BOARD
 0 ether5    192.168.88.1                                    CC:2D:E0:11:22:33 R1         6.45.4 ... CCR1036-
8G-2S+
   bond1
```

LLDP

Depending on RouterOS configuration, different type-length-value (TLV) can be sent in the LLDP message, this includes:

- Chassis ID (MAC address)
- Port ID (interface name)
- Time To Live
- System Name (system identity)
- System Description (platform - MikroTik, software version - RouterOS version, hardware name - RouterBoard name)
- Management Address (all IP addresses configured on the port)
- System Capabilities (enabled system capabilities, e.g. bridge or router)
- Port Description (combined interface name like "bridge/ether1" if the sending interface is part of bridge or bond, or interface name same as Port ID)
- IEEE 802.3 MAC/PHY Configuration/Status
- IEEE 802.3 Power Via MDI
- IEEE 802.3 Maximum Frame Size
- LLDP-MED Media Capabilities (list of MED capabilities)
- LLDP-MED Network Policy (assigned VLAN ID for voice traffic)
- LLDP-MED Extended Power via MDI
- Port Extension (Port Extender and Controller Bridge advertisement)
- End of LLDPDU

Note

Summary

```
/system note
```

The system note feature allows you to assign arbitrary text notes or messages that will be displayed on each login right after the banner. For example, you may distribute warnings between system administrators this way, or describe what does that particular router actually do. To configure system note, you may upload a plain text file named **sys-note.txt** on the router's FTP server, or, additionally, edit the settings in this menu

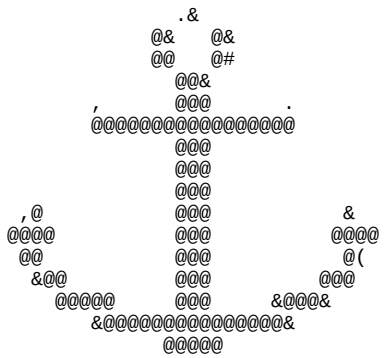
Properties

Property	Description
note (*string*; Default:)	Note that will be displayed.
show-at-login (*yes* | *no*; Default: **yes**)	whether to show system note on each login

Example

It is possible to add multi-line notes using an embedded text editor (*/system note edit note*), for example, add ASCII art to your home router:

```
system/note/set note=
```



NTP

RouterOS v6 implements the SNTP protocol defined in RFC4330, manycast mode is not supported. SNTP client is included in the *system* package. To use an NTP server, *ntp* package must be installed and enabled.

RouterOS v7 main package includes NTP client and server functionality, which is based on RFC5905.

The client configuration is located in the **/system ntp client** console path, and the ***"System > SNTP Client"*** (RouterOS version 6), ***"System > NTP Client"*** (RouterOS version 7) WinBox window. This configuration is shared by the SNTP client implementation in the *system* package and the NTP client implementation in the *ntp* package. When *ntp* package is installed and enabled, the SNTP client is disabled automatically.

RouterOS version 6

SNTP Client properties:

Property	Description
enabled (*yes, no default: no*)	Enable SNTP client for time synchronization
mode (*broadcast, unicast, filed is read-only*)	Mode that the SNTP client will operate in. If no NTP servers are configured *broadcast* mode will be used. If there is a dynamic or static NTP server IP address or FQDN used it will automatically switch to unicast mode.
primary-ntp (*IP address default: 0.0.0.0*)	IP address of the NTP server that has to be used for time synchronization. If both values are non-zero, then the SNTP client will alternate between the two server addresses, switching to the other when the request to the current server times out or when the "KoD" packet is received, indicating that the server is not willing to respond to requests from this client. The following formats are accepted: - *ipv4* - *ipv6*
secondary-ntp (*IP address default: 0.0.0.0*)	see **primary-ntp**
server-dns-names (*Comma separated domain name list default:*)	To set the NTP server using its domain name. The domain name will be resolved each time an NTP request is sent. Router has to have */ip dns* configured.

Status

- **active-server** (IP address; read-only property) : Currently selected NTP server address. This value is equal to **primary-ntp** or **secondary-ntp**.
- **poll-interval** (Time interval; read-only property) : Current interval between requests sent to the active server. The initial value is 16 seconds, and it is increased by doubling to 15 minutes.

Last received packet information

Values of the following properties are reset when the SNTP client is stopped or restarted, either because of a configuration change, or because of a network error.

- **last-update-from** (IP address; read-only property) : Source IP address of the last received NTP server packed that was successfully processed.

- **last-update-before** (Time interval; read-only property) : Time since the last successfully received server message.
- **last-adjustment** (Time interval; read-only property) : Amount of clock adjustment that was calculated from the last successfully received NTP server message.
- **last-bad-packet-from** (IP address; read-only property) : Source IP address of last received SNTP packed that was not successfully processed. Reason of the failure and time since this packet was received is available in the next two properties.
- **last-bad-packet-before** (Time interval; read-only property) : Time since the last receive failure.
- **last-bad-packet-reason** (Text; read-only property) : Text that describes the reason of the last receive failure. Possible values are:
  - *bad-packet-length* - Packet length is not in the acceptable range.
  - *server-not-synchronized* - Leap Indicator field is set to "alarm condition" value, which means that clock on the server has not been synchronized yet.
  - *zero-transmit-timestamp* - Transmit Timestamp field value is 0.
  - *bad-mode* - Value of the Mode field is neither 'server' nor 'broadcast'.
  - *kod-ABCD* - Received "KoD" (Kiss-o'-Death) response. *ABCD* is the short "kiss code" text from the Reference Identifier field.
  - *broadcast* - Received proadcast message, but **mode**=*unicast*.
  - *non-broadcast* - Received packed was server reply, but **mode**=*broadcast*.
  - *server-ip-mismatch* - Received response from address that is not **active-server**.
  - *originate-timestamp-mismatch* - Originate Timestamp field in the server response message is not the same as the one included in the last request.
  - *roundtrip-too-long* - request/response roundtrip exceeded 1 second.

Client settings example:

To check the status of the NTP client in CLI, use the "print" command

```
[admin@ntp-example_v6] > /system ntp client print
           enabled: no
      primary-ntp: 0.0.0.0
    secondary-ntp: 0.0.0.0
  server-dns-names:
              mode: unicast
```

To enable the NTP client and set IP addresses or FQDN of the NTP servers:

```
[admin@ntp-example_v6] > /system ntp client set enabled=yes
[admin@ntp-example_v6] > /system ntp client print
            enabled: yes
        primary-ntp: 0.0.0.0
      secondary-ntp: 0.0.0.0
   server-dns-names:
               mode: unicast
    dynamic-servers: x.x.x.x, x.x.x.x
      poll-interval: 15s
      active-server: x.x.x.x
   last-update-from: x.x.x.x
  last-update-before: 6s570ms
     last-adjustment: -1ms786us
[admin@ntp-example_v6] > /system ntp client set primary-ntp=162.159.200.123
[admin@ntp-example_v6] > /system ntp client print
           enabled: yes
      primary-ntp: 162.159.200.123
    secondary-ntp: 0.0.0.0
  server-dns-names:
              mode: unicast
   dynamic-servers: x.x.x.x, x.x.x.x
     poll-interval: 16s
     active-server: x.x.x.x
```

NTP Server settings:

Server configuration is located in **/system ntp server**

Property	Description

enabled (*yes* or *no*; default value: *no*)	Enable NTP server
broadcast (*yes* or *no*; default value: *no*)	Enable certain NTP server mode, for this mode to work you have to set up broadcast-addresses field
multicast (*yes* or *no*; default value: *no*)	Enable certain NTP server mode
manycast (*yes* or *no*; default value: *no*)	Enable certain NTP server mode
broadcast-addresses (*IP address*; default value:)	Set broadcast address to use for NTP server broadcast mode

Example:

Set up an NTP server for the local network that is 192.168.88.0/24

```
/system ntp server set broadcast=yes broadcast-addresses=192.168.88.255 enabled=yes manycast=no
```

RouterOS version 7

NTP Client properties:

Property	Description
enabled (*yes, no default: no*)	Enable NTP client for time synchronization
mode (*broadcast, manycast, multicast, unicast*)	Mode that the NTP client will operate in
NTP servers	The list of NTP servers. It is possible to add static entries. The following formats are accepted: *- FQDN ("Resolved Address" will appear in the "Servers"- window in an appropriate column if the address is resolved) or IP address can be used. If DHCP-Client property* ***use-peer-ntp=yes*** *- the dynamic entries advertised by DHCP* *- ipv4* *- ipv4@vrf* *- ipv6* *- ipv6@vrf* *- ipv6-linklocal%interface*
vrf (*default: main*)	Virtual Routing and Forwarding
Servers (*Button/Section*)	A detailed table of dynamically and statically added NTP servers (Address, Resolved address, Min Poll, Max Poll, iBurst, Auth. Key) To set the NTP server using its FQDN. The domain name will be resolved each time an NTP request is sent. Router has to have */ip/dns* configured.

Peers	Current parameter values
	
	`[admin@ntp-example_v7] > /system/ntp/monitor-peers`
	` type="ucast-client" address=x.x.x.x refid="y.y.y.y" stratum=3 hpoll=10 ppoll=10 root-delay=28.869 ms root-disp=50.994 ms`
	`   offset=-0.973 ms delay=0.522 ms disp=15.032 ms jitter=0.521 ms`
	`-- [Q quit|D dump|C-z pause]`

Keys	NTP symmetric keys, used for authentication between the NTP client and server. Key Identifier (Key ID) - an integer identifying the cryptographic key used to generate the message-authentication code.

Status

- **synchronized, stopped, waiting, using-local-clock** - Current status of the NTP client
- **Frequency drift** - The fractional frequency drift per unit time.
- **synced-server** - The IP address of the NTP Server.
- **synced-stratum** - The accuracy of each server is defined by a number called the stratum, with the topmost level (primary servers) assigned as one and each level downwards (secondary servers) in the hierarchy assigned as one greater than the preceding level.
- **system-offset** - This is a signed, fixed-point number indicating the offset of the NTP server's clock relative to the local clock, in seconds.

NTP Server settings:

Server configuration is located in **/system ntp server**

Property	Description
enabled (*yes* or *no*; default value: *no*)	Enable NTP server
broadcast (*yes* or *no*; default value: *no*)	Enable certain NTP server mode, for this mode to work you have to set up broadcast-addresses field
multicast (*yes* or *no*; default value: *no)*	Enable certain NTP server mode
manycast (*yes* or *no*; default value: *no)*	Enable certain NTP server mode
broadcast-addresses (*IP address*; default value:)	Set broadcast address to use for NTP server broadcast mode
vrf (*default: main*)	Virtual Routing and Forwarding
use-local-clock (*yes* or *no*; default value: *no*)	The server will supply its local system time as valid if others are not available.
local-clock-stratum	Manually set stratum if **use-local-clock=yes**
auth-key (default value: *none*)	NTP symmetric key, used for authentication between the NTP client and server. Key Identifier (Key ID) - an integer identifying the cryptographic key used to generate the message-authentication code.

Log messages

SNTP client can produce the following log messages. See the article "log" on how to set up logging and how to inspect logs.

- **ntp**,**debug** gradually adjust by *OFFS*
- **ntp**,**debug** instantly adjust by *OFFS*
- **ntp**,**debug** Wait for *N* seconds before sending the next message
- **ntp**,**debug** Wait for *N* seconds before restarting
- **ntp**,**debug**,**packet** packet receive an error, restarting
- **ntp**,**debug**,**packet** received *PKT*
- **ntp**,**debug**,**packet** ignoring received *PKT*
- **ntp**,**debug**,**packet** error sending to *IP*, restarting
- **ntp**,**debug**,**packet** sending to *IP PKT*

Explanation of log message fields

- *OFFS* - difference of two NTP timestamp values, in hexadecimal.
- *PKT* - dump of NTP packet. If the packet is shorter than the minimum 48 bytes, it is dumped as a hexadecimal string. Otherwise, the packet is dumped as a list of field names and values, one per log line. Names of fields follow RFC4330.
- *IP* - remote IP address.

NOTE: the above logging rules work only with the built-in SNTP client, the separate NTP package doesn't have any logging facilities.

Partitions

Summary

Partitioning is supported on ARM, ARM64, MIPS, TILE, and PowerPC RouterBOARD type devices.

It is possible to partition NAND flash, allowing to install own OS on each partition and specify primary and fallback partitions.

If a partition should fail for some reason (failed upgrade, problematic configuration introduced, software problem), the next partition will boot instead. This can be used as an interactive backup where you keep a verified working installation and upgrade only some secondary partition. If you upgrade your configuration, and it proves to be good, you can use the "save config" button to copy it over to other partitions.

⚠ Repartitioning of the NAND requires the latest bootloader version

Minimum partition sizes:

- 32MB on MIPS
- 40MB on PowerPC
- 48MB on TILE

The maximum number of allowed partitions is 8.

```
[admin@1009up] > /partitions/print
Flags: A - ACTIVE; R - RUNNING
Columns: NAME, FALLBACK-TO, VERSION, SIZE
# NAME FALL VERSION SIZE
0 AR part0 next RouterOS v7.1beta4 Dec/15/2020 15:55:11 128MiB
```

Commands

Property	Description
repartition (*integer*)	Will reboot the router and reformat the NAND, leaving only active partition.
copy-to (*<partition>*)	Clone **running** OS with the config to specified partition. Previously stored data on the partition will be erased.
save-config-to (*<partition>*)	Clone **running-config** on a specified partition. Everything else is untouched.
restore-config-from (*<partition>*)	Copy config from specified partition to **running** partition

Properties

Property	Description
name (*string*; Default:)	Name of the partition
fallback-to (*etherboot* | *next* | *<partition-name>*; Default: **next**)	What to do if an active partition fails to boot: - **etherboot** - switch to etherboot - **next'** - try next partition - fallback to the specified partition

Read-only

Property	Description

active (*yes* | *no*)	Partition is active
running (*yes* | *no*)	Currently running partition
size (*integer[MiB]*)	Partition size
version (*string*)	Current RouterOS version installed on the partition

Precision Time Protocol

Summary

Precision Time Protocol is used to synchronize clocks throughout the network. On a local area network, it achieves clock accuracy in the sub-microsecond range, making it suitable for measurement and control systems. RouterOS supports IEEE 1588-2008, PTPv2. Support is hardware dependant, please see the supported device list below.

Supported features:

- Boundary/Ordinary clock
- E2E delay mode
- PTP delay mode
- UDP over IPv4 multicast transport mode
- L2 transport mode
- priority1 can be configured to decide master/slave
- PTP clock IS NOT synced with the system clock

General properties

Sub-menu: `/system ptp`

Property	Description
port	Sub-menu used for adding, removing, or viewing assigned ports
status	Sub-menu that shows PTP ports, their state, and delay on slave ports
comment (*string*; Default:)	Short description of the PTP profile
name (*string*; Default:)	Name of the PTP profile
delay-mode (*auto* | *e2e* | *ptp*; Default: **auto**)	Configures delay mode for PTP profile ○ *auto* - selects delay mode automatically ○ *e2e* - use the delay request-response mechanism ○ *ptp* - use the peer delay mechanism
priority1 (*integer [0..255]*; auto; Default: **auto**)	the priority value for influencing grandmaster election
profile (*802.1as; default; g8275.1;* Default: **default**)	IEEE 1588-2008 includes a *profile* concept defining PTP operating parameters and options. IEEE 802.1AS is an adaptation of PTP for use with Audio Video Bridging and Time-Sensitive Networking. Uses delay-mode=p2p, transport-mode=l2; recommends using priority1=auto. g8275.1 profile is for frequency and phase synchronization in a fully PTP-aware network. Only allows priority1=auto (128), priority2=128, domain=24, delay-mode=e2e, transport=l2. default profile, PTPv2 default configuration, allows for more configuration options than other profiles, but default values with auto settings correspond to: priority1=128. priority2=128, domain=0,transport=ipv4, delay-mode=e2e
transport (*auto; ipv4; l2;* Default: **auto**)	transport protocol to be used: IPv4 or layer2

For more details regarding Precision Time Protocol please see the following standards IEEE 1588 and IEEE 802.1as.

We strongly recommend keeping default/auto values, as there are different requirements between profiles. And assigning them manually can result in misconfiguration.

Configuration

To configure the device to participate in PTP you first need to create a PTP profile:

```
/system ptp add name=ptp1
#to view the created profile use
/system ptp print
Flags: I - inactive, X - disabled
0 name="ptp1" priority1=auto delay-mode=auto transport=auto profile=default
```

Note

Only 1 PTP profile is supported per device

After creating a PTP profile, you need to assign ports to it:

```
/system ptp port add interface=ether1 ptp=ptp1
#to view assigned ports use
/system ptp port print
Flags: I - inactive
0 ptp=ptp1 interface=ether8

1 ptp=ptp1 interface=ether22
```

To monitor the PTP profile, use the monitor command:

```
#on grandmaster device
[admin@grandmaster] > system ptp monitor numbers=0
name: test
clock-id: 64:D1:54:FF:FE:EB:AE:C3
priority1: 30
priority2: 128
i-am-gm: yes

#on non-grandmaster device
[admin@328] /system ptp monitor 0
name: ptp1
clock-id: 64:D1:54:FF:FE:EB:AD:C7
priority1: 128
priority2: 128
i-am-gm: no
gm-clock-id: 64:D1:54:FF:FE:EB:AE:C3
gm-priority1: 30
gm-priority2: 128
master-clock-id: 64:D1:54:FF:FE:EB:AE:C3
slave-port: ether8
freq-drift: 2147483647 ppb
offset: 1396202830 ns
hw-offset: 1306201921 ns
slave-port-delay: 2075668440 ns
```

Monitor properties

Property	Description
clock-id:	local clock ID
priority1:	priority1 value, depending on the PTP profile selected, an adjustable value used to influence the grandmaster election.
priority2:	priority2 value, non-adjustable in RouterOS
i-am-gm: yes | no	shows if the device is a grandmaster clock
gm-clock-id:	grandmaster clock ID - Within a domain, a clock that is the ultimate source of time for clock synchronization using the protocol.
gm-priority1:	grandmaster priority1
gm-priority2:	grandmaster priority2
master-clock-id:	master clock ID - In the context of a single Precision Time Protocol (PTP) communication path, a clock that is the source of time to which all other clocks on that path synchronize.
slave-port:	shows which port is going towards the master or grandmaster clock
freq-drift:	frequency drift in PPB (parts per billion) - time that would be lost every second in relation to the master clock, IF there was no synchronization.
offset:	difference between clock values
hw-offset:	offset difference from the hardware clock
slave-port-delay:	the time it takes for a packet to be delivered to a directly connected device

Device support

Note: devices not included in the list below, does not support Precision Time Protocol

Supported on:

- CRS326-24G-2S+ supported only on Gigabit Ethernet ports
- CRS328-24P-4S+ supported only on Gigabit Ethernet ports
- CRS317-1G-16S+ supported on all ports
- CRS326-24S+2Q+ supported on SFP+ and QSFP+ interfaces
- CRS312-4C+8XG supported on all ports
- CRS318-16P-2S+ supported only on Gigabit Ethernet ports

Scheduler

Summary

The scheduler can trigger script execution at a particular time moment, after a specified time interval, or both.

Properties

- **interval** *(time; default: 0s)* - interval between two script executions, if time interval is set to zero, the script is only executed at its start time, otherwise it is executed repeatedly at the time interval is specified
- **name** *name)* - name of the task
- **on-event** *(name)* - name of the script to execute. It must be presented at /system script
- **run-count** *(read-only: integer)* - to monitor script usage, this counter is incremented each time the script is executed
- **start-date** *(date)* - date of the first script execution
- **start-time** *(time)* - time of the first script execution
- **startup** - execute the script 3 seconds after the system startup.

Notes

Rebooting the router will reset the run-count counter.

If more than one script has to be executed simultaneously, they are executed in the order they appear in the scheduler configuration. This can be important if one scheduled script is used to disable another one.

If a more complex execution pattern is needed, it can usually be done by scheduling several scripts, and making them enable and disable each other.

Note: if scheduler item has start-time set to startup, it behaves as if start-time and start-date were set to time 3 seconds after console starts up. It means that all scripts having `start-time is startup` and `interval is 0` will be executed once each time router boots. If the interval is set to value other than 0 scheduler will **not** run at startup

Examples

We will add a task that executes the script log-test every hour:

```
[admin@MikroTik] system script> add name=log-test source=":log info message=test"
[admin@MikroTik] system script> print
Flags: I - invalid
0 name="log-test" owner="admin" policy=ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon dont-
require-permissions=no run-count=0
source=:log info message=test
[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add name=run-1h interval=1h
on-event=log-test
[admin@MikroTik] system scheduler> print
Flags: X - disabled
# NAME ON-EVENT START-DATE START-TIME INTERVAL RUN-COUNT
0 run-1h log-test mar/30/2004 06:11:35 1h 0
[admin@MikroTik] system scheduler>
```

In another example, there will be two scripts added that will change the bandwidth setting of a queue rule "Cust0". Every day at 9AM the queue will be set to 64Kb/s and at 5PM the queue will be set to 128Kb/s. The queue rule, the scripts, and the scheduler tasks are below:

```
[admin@MikroTik] queue simple> add name=Cust0 interface=ether1 \
\... dst-address=192.168.0.0/24 limit-at=64000
 [admin@MikroTik] queue simple> print
 Flags: X - disabled, I - invalid 0 name="Cust0" target-address=0.0.0.0/0 dst-address=192.168.0.0/24
 interface=ether1 limit-at=64000 queue=default priority=8 bounded=yes
[admin@MikroTik] queue simple> /system script
[admin@MikroTik] system script> add name=start_limit source={/queue simple set \
 \... Cust0 limit-at=64000}
[admin@MikroTik] system script> add name=stop_limit source={/queue simple set \
\... Cust0 limit-at=128000}
[admin@MikroTik] system script> print
0 name="start_limit" source="/queue simple set Cust0 limit-at=64000"
owner=admin run-count=0
1 name="stop_limit" source="/queue simple set Cust0 limit-at=128000"
owner=admin run-count=0
[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add interval=24h name="set-64k" \
\... start-time=9:00:00 on-event=start_limit
[admin@MikroTik] system scheduler> add interval=24h name="set-128k" \
 \... start-time=17:00:00 on-event=stop_limit
[admin@MikroTik] system scheduler> print
 Flags: X - disabled
# NAME ON-EVENT START-DATE START-TIME INTERVAL RUN-COUNT
 0 set-64k start... oct/30/2008 09:00:00 1d 0
1 set-128k stop_... oct/30/2008 17:00:00 1d 0
[admin@MikroTik] system scheduler>
```

The following example schedules a script that sends each week a backup of router configuration by e-mail.

```
[admin@MikroTik] system script> add name=e-backup source={/system backup
{... save name=email; /tool e-mail send to="root@host.com" subject=([/system
{... identity get name] . " Backup") file=email.backup}
[admin@MikroTik] system script> print
0 name="e-backup" source="/system backup save name=ema... owner=admin run-count=0

[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add interval=7d name="email-backup" \
 \... on-event=e-backup
[admin@MikroTik] system scheduler> print
 Flags: X - disabled
 # NAME ON-EVENT START-DATE START-TIME INTERVAL RUN-COUNT
0 email-... e-backup oct/30/2008 15:19:28 7d 1
[admin@MikroTik] system scheduler>
```

Do not forget to set the e-mail settings, i.e., the SMTP server and From: address under /tool e-mail. For example:

```
[admin@MikroTik] tool e-mail> set server=159.148.147.198 from=SysAdmin@host.com
[admin@MikroTik] tool e-mail> print
 server: 159.148.147.198
from: SysAdmin@host.com
[admin@MikroTik] tool e-mail>
```

Example below will put 'x' in logs each hour from midnight till noon:

```
[admin@MikroTik] system script> add name=enable-x source={/system scheduler
{... enable x}
[admin@MikroTik] system script> add name=disable-x source={/system scheduler
{... disable x}
[admin@MikroTik] system script> add name=log-x source={:log info message=x}
[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add name=x-up start-time=00:00:00 \
\... interval=24h on-event=enable-x
[admin@MikroTik] system scheduler> add name=x-down start-time=12:00:00
 \... interval=24h on-event=disable-x
[admin@MikroTik] system scheduler> add name=x start-time=00:00:00 interval=1h \
\... on-event=log-x
[admin@MikroTik] system scheduler> print
Flags: X - disabled
 # NAME ON-EVENT START-DATE START-TIME INTERVAL RUN-COUNT
0 x-up enable-x oct/30/2008 00:00:00 1d 0
1 x-down disab... oct/30/2008 12:00:00 1d 0
2 x log-x oct/30/2008 00:00:00 1h 0
[admin@MikroTik] system scheduler>
```

Services

Summary

This page lists protocols and ports used by various MikroTik RouterOS services. It helps you to determine why your MikroTik router listens to certain ports, and what you need to block/allow in case you want to prevent or grant access to certain services. Please see the relevant sections of the Manual for more explanations.

The default services are:

Property	Description
telnet	Telnet service
ftp	FTP service
www	Webfig HTTP service
ssh	SSH service
www-ssl	Webfig HTTPS service
api	API service
winbox	Responsible for Winbox tool access, as well as Tik-App smartphone app and Dude probe
api-ssl	API over SSL service

Properties

Note that it is not possible to add new services, only existing service modifications are allowed.

Sub-menu: `/ip service`

Property	Description
address (*IP address/netmask | IPv6/0..128* ; Default:)	List of IP/IPv6 prefixes from which the service is accessible
certificate (*name*; Default: **none**)	The name of the certificate used by a particular service. Applicable only for services that depend on certificates (*www-ssl, api-ssl*)
name (*name*; Default: **none**)	Service name
port (*integer: 1..65535*; Default:)	The port particular service listens on
tls-version (*any | only-1.2*; Default: **any**)	Specifies which TLS versions to allow by a particular service
vrf (*name*; Default: **main**)	Specify which VRF instance to use by a particular service

Example

For example, allow API only from a specific IP/IPv6 address range

```
[admin@dzeltenais_burkaans] /ip service> set api address=10.5.101.0/24,2001:db8:fade::/64
[admin@dzeltenais_burkaans] /ip service> print
Flags: X - disabled, I - invalid
 #   NAME     PORT  ADDRESS                                       CERTIFICATE
 0   telnet   23
 1   ftp      21
 2   www      80
 3   ssh      22
 4 X www-ssl  443                                                 none
 5   api      8728  10.5.101.0/24
                    2001:db8:fade::/64
 6   winbox   8291
```

Service Ports

Hosts behind a NAT-enabled router do not have true end-to-end connectivity. Therefore some Internet protocols might not work in scenarios with NAT.

To overcome these limitations RouterOS includes a number of NAT helpers, that enable NAT traversal for various protocols.

If connection tracking is not enabled then firewall service ports will be shown as inactive

Sub-menu: `/ip firewall service-port`

Helper	Description
FTP	FTP service helper
H323	H323 service helper
IRC	IRC service helper
PPTP	PPTP tunneling helper
UDPLITE	UDP-Lite service helper
DCCP	DCCP service helper
SCTP	SCTP service helper
SIP	SIP helper. Additional options: • **sip-direct-media** allows redirecting the RTP media stream to go directly from the caller to the callee. The default value is *yes*. • **sip-timeout** allows adjusting TTL of SIP UDP connections. Default: 1 hour. In some setups, you have to reduce that.
TFTP	TFTP service helper
RSTP	RTSP service helper

udplite, **dccp**, and **sctp** are built-in services of the connection tracking. Since these are not separately loaded modules, they cannot be disabled separately, they got disabled together with the connection tracking.

Protocols and ports

The table below shows the list of protocols and ports used by RouterOS.

Proto/Port	Description

20/tcp	FTP data connection
21/tcp	FTP control connection
22/tcp	Secure Shell (SSH) remote login protocol
23/tcp	Telnet protocol
53/tcp **53/udp**	DNS
67/udp	Bootstrap protocol or DHCP Server
68/udp	Bootstrap protocol or DHCP Client
80/tcp	World Wide Web HTTP
123/udp	Network Time Protocol (NTP)
161/udp	Simple Network Management Protocol (SNMP)
179/tcp	Border Gateway Protocol (BGP)
443/tcp	Secure Socket Layer (SSL) encrypted HTTP
500/udp	Internet Key Exchange (IKE) protocol
520/udp **521/udp**	RIP routing protocol
546/udp	DHCPv6 Client message
547/udp	DHCPv6 Server message
646/tcp	LDP transport session
646/udp	LDP hello protocol
1080/tcp	SOCKS proxy protocol
1698/udp 1699/udp	RSVP TE Tunnels
1701/udp	Layer 2 Tunnel Protocol (L2TP)
1723/tcp	Point-To-Point Tunneling Protocol (PPTP)
1900/udp **2828/tcp**	Universal Plug and Play (uPnP)
1966/udp	MME originator message traffic
1966/tcp	MME gateway protocol
2000/tcp	Bandwidth test server
5246,5247/udp	CAPsMAN
5350/udp	NAT-PMP client
5351/udp	NAT-PMP server
5678/udp	Mikrotik Neighbor Discovery Protocol
6343/tcp	Default OpenFlow port
8080/tcp	HTTP Web Proxy
8291/tcp	Winbox
8728/tcp	API
8729/tcp	API-SSL

20561/udp	MAC winbox
/1	ICMP
/2	Multicast | IGMP
/4	IPIP encapsulation
/41	IPv6 (encapsulation)
/46	RSVP TE tunnels
/47	General Routing Encapsulation (GRE) - used for PPTP and EoIP tunnels
/50	Encapsulating Security Payload for IPv4 (ESP)
/51	Authentication Header for IPv4 (AH)
/89	OSPF routing protocol
/103	Multicast | PIM
/112	VRRP

TFTP

Introduction

Trivial File Transfer Protocol or simply TFTP is a very simple protocol used to transfer files. Each nonterminal packet is acknowledged separately.

```
ip/tftp/
```

This menu contains all TFTP access rules. If in this menu are no rules, the TFTP server is not started when RouterOS boots. This menu only shows 1 additional attribute compared to what you can set when creating a rule.

Parameters

Property	Description
ip-address *(required)*	Range of IP addresses accepted as clients if empty *0.0.0.0/0* will be used
allow-rollover *(Default: No)*	If set to *yes* TFTP server will allow the sequence number to roll over when the maximum value is reached. This is used to enable large downloads using the TFTP server.
req-filename	Requested filename as a *regular expression (regex)* if a field is left empty it defaults to *.**
real-filename	If **req-filename** and **real-filename** values are set and valid, the requested filename will be replaced with matched file. This field has to be set. If multiple *regex* is specified in *req-filename*, with this field you can set which ones should match, so this rule is validated. The *real-filename* format for using multiple *regex* is **filename\0\5\6**
allow (*default: yes*)	To allow connection if the above fields are set. if *no*, a connection will be interrupted
read-only (*default: no*)	Sets if a file can be written to, if set to "yes" write attempt will fail with error
hits *(read-only)*	How many times this access rule entry has been used (read-only)

Settings

```
/ip/tftp/settings
```

This menu contains all TFTP settings.

Property	Description
max-block-size (*default: 4096*)	Maximum accepted block size value. During the transfer negotiation phase, the RouterOS device will not negotiate a larger value than this.

Regexp

Req-filename field allowed regexp, allowed regexp in this field are:

brackets () - marking subsection:

```
example 1 a(sd|fg) will match asd or afg
```

asterisk "*" - match zero or more times preceding symbol:

```
example 1 a* will match any length name consisting purely of symbols a or no symbols at all
example 2 .* will match any length name, also, empty field
example 3 as*df will match adf, asdf, assdf, asssdf etc.
```

plus "+" will match one or more times the preceding symbol:

```
example: as+df will match asdf, assdf etc.
```

dot "." - matches any symbol:

```
example as.f will match asdf, asbf ashf etc.
```

square brackets [] - variation between:

```
example as[df] will match asd and asf
```

question mark "?" will match one or no symbols:

```
example asd?f will match asdf and asf
```

caret "^" - used at the beginning of the line means that the line starts with;

dollar "$" - means at the end of the line.

Examples

If a file is requested return the file from the store called sata1:

```
/ip tftp add req-filename=file.txt real-filename=/sata1/file.txt allow=yes read-only=yes
```

If we want to give out one specific *file* no matter what the user is requesting:

```
/ip tftp add req-filename=.* real-filename=/sata1/file.txt allow=yes read-only=yes
```

If the user requests *aaa.bin* or *bbb.bin* then give them *ccc.bin*:

```
/ip tftp add req-filename="(aaa.bin)|(bbb.bin)" real-filename="/sata1/ccc.bin\\0" allow=yes read-only=yes
```

RouterOS receives TFTP requests, but the client gets a transfer timeout?

Some embedded clients request large block sizes and yet do not handle fragmented packets correctly. For these clients, it is recommended to set "max-block-size" on the RouterOS side or "blksize" on Client-side to the value of the smallest MTU on your network minus 32 bytes (20 bytes for IP, 8 for UDP, and 4 for TFTP) and more if you use IP options on your network.

Virtual Private Networks

In This Section:

6to4

Summary

Sub-menu: `/interface 6to4`

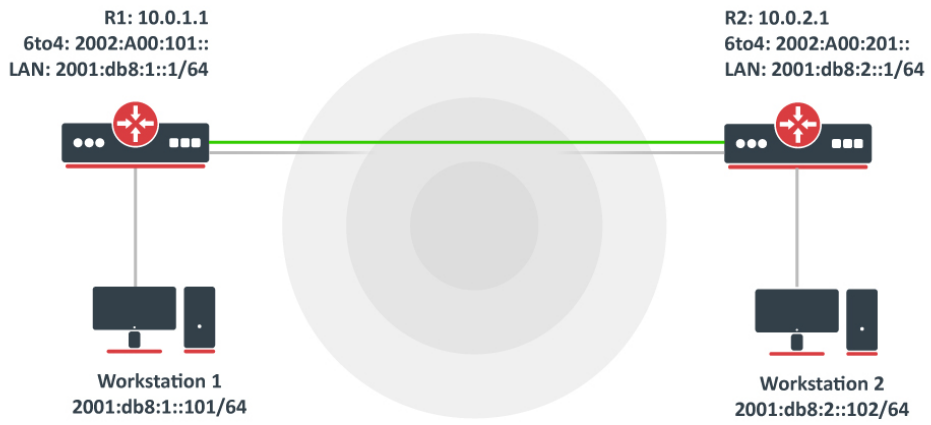
6to4 is a special mechanism that allows IPv6 packets to be transmitted over IPv4 networks without the need of explicitly configured tunnel interfaces. It is especially useful for connecting two or more IPv6 networks over a network that does not have IPv6 support. There are two different ways of 6to4 mechanism. If *remote-address* is not configured, the router will encapsulate and send an IPv6 packet directly over IPv4 if the first 16 bits are *2002*, using the next 32 bits as the destination (IPv4 address converted to hex). In other case, the IPv6 packet will be sent directly to the IPv4 *remote-address*.

Property Description

Property	Description
clamp-tcp-mss (*yes* | *no*; Default: **yes**)	Controls whether to change MSS size for received TCP SYN packets. When enabled, a router will change the MSS size for received TCP SYN packets if the current MSS size exceeds the tunnel interface MTU (taking into account the TCP/IP overhead). The received encapsulated packet will still contain the original MSS, and only after decapsulation the MSS is changed.
comment (*string* ; Default:)	Short description of the interface.
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled.
dont-fragment (*inherit* | *no*; Default: **no**)	Whether to include DF bit in related packets: *no* - fragment if needed, *inherit* - use Dont Fragment flag of original packet. (Without Dont Fragment: inherit - packet may be fragmented).
dscp (*integer: 0-63*; Default: **inherited**)	DSCP value of packet. Inherited option means that DSCP value will be inherited from packet which is going to be encapsulated.
ipsec-secret (*string*; Default:)	When secret is specified, router adds dynamic IPsec peer to remote-address with pre-shared key and policy (by default phase2 uses sha1/aes128cbc).
keepalive (*integer[/time],integer 0..4294967295*; Default: **0,0**)	Tunnel keepalive parameter sets the time interval in which the tunnel running flag will remain even if the remote end of tunnel goes down. If configured time,retries fail, interface running flag is removed. Parameters are written in following format: `KeepaliveInterval,KeepaliveRetries` where `KeepaliveInterval` is time interval and `KeepaliveRetries` - number of retry attempts. By default keepalive is set to 10 seconds and 10 retries.
local-address (*IP* ; Default:)	Source address of the packets, local on the router.
mtu (*integer*; Default: **auto**)	Layer3 maximum transmission unit.
name (*string*; Default:)	Interface name.
remote-address (*IP*; Default:)	IP address of remote end of 6to4 tunnel. If left unspecified, IPv4 address from 2002::/16 gateway address will be derived.

Configuration Examples

Simple 6to4 tunnel encapsulation (Currently not working)



It is possible to simply route IPv6 packets over IPv4 network by utilizing the 2002::/16 allocated address space. All 6to4 nodes has to have reachable IPv4 addresses - if you are running this setup over the Internet, all IPv4's must be public addresses.

R1 configuration:

Create the 6to4 tunnel interface:

```
/interface 6to4
add name=6to4-tunnel1
```

Assign an IPv6 address with '2002' as the first 16 bits and IPv4 in hex format as the next 32 bits. For example, if the router's IP address is 10.0.1.1, the IPv6 address is 2002:A00:101::

```
/ipv6 address
add address=2002:a00:101::/128 advertise=no interface=6to4-tunnel1
```

Add a route to specially allocated 6to4 tunnel range over the 6to4-tunnel interface.

```
/ipv6 route
add dst-address=2002::/16 gateway=6to4-tunnel1
```

R2 configuration:

Create the 6to4 tunnel interface:

```
/interface 6to4
add name=6to4-tunnel1
```

Assign an IPv6 address that is generated by the same principles as R1. In this case, 10.0.2.1 translates to 2002:A00:201::

```
/ipv6 address
add address=2002:a00:201::/128 advertise=no interface=6to4-tunnel1
```

The 6to4 route is necessary on this side as well.

```
/ipv6 route
add dst-address=2002::/16 gateway=6to4-tunnel1
```

Testing:

After configuring both devices, it should be possible to ping the IPv6 addresses if they were generated correctly.

From R1:

```
/ping 2002:a00:201::
```

Hurricane Electric Tunnel Broker Example

Following example will show how to get IPv6 connectivity on a RouterOS device through IPv4 network using 6to4 tunnel.

To be able to create the tunnel, you have to have a public IPv4 address and enable ping from Tunnel Broker IPv4 server.

When you create a tunnel using Hurricane Electric Tunnel Broker, you will be given a routed /64 IPv6 prefix and additional information necessary for setting up the tunnel.

IPv6 Tunnel Endpoints

Server IPv4 address:	216.66.80.90
Server IPv6 address:	2001:470:**27**:37e::1/64
Client IPv4 address:	**194.105.56.170**
Client IPv6 address:	2001:470:**27**:37e::2/64

Available DNS Resolvers

Anycasted IPv6 Caching Nameserver:	2001:470:20::2
Anycasted IPv4 Caching Nameserver:	74.82.42.42

Routed IPv6 Prefixes and rDNS Delegations

Routed /64:	2001:470:**28**:37e::/64

This example presumes that your public IPv4 address is 194.105.56.170

Hurricane Electric provides ready to use commands for RouterOS in the 'Example Configurations' section:

```
/interface 6to4
add comment="Hurricane Electric IPv6 Tunnel Broker" disabled=no local-address=194.105.56.170 mtu=1280 name=sit1
remote-address=216.66.80.90
/ipv6 route
add comment="" disabled=no distance=1 dst-address=2000::/3 gateway=2001:470:27:37e::1 scope=30 target-scope=10
/ipv6 address
add address=2001:470:27:37e::2/64 advertise=no disabled=no eui-64=no interface=sit1
```

These commands will setup the tunnel itself - the router will be able to connect to IPv6 hosts, but end-user devices (computers, tablets, phones) will not yet have IPv6 connectivity.

To be able to assign IPv6 addresses to your clients you have to add the Routed IPv6 Prefix to your internal interface (by default bridge-local).

```
/ipv6 address add address=2001:470:28:37e:: interface=bridge-local advertise=yes
```

Enable DNS server advertising through network discovery

```
/ipv6 nd set [ find default=yes ] advertise-dns=yes
```

And finally add IPv6 DNS servers (these are Google public DNS servers, you can also use the one which is provided by Hurricane Electric - 2001:470:20::2).

```
/ip dns set allow-remote-requests=yes servers=2001:4860:4860::8888,2001:4860:4860::8844
```

Afterwards enable IPv6 on your device and you should have IPv6 connectivity. http://ipv6-test.com can be used to test IPv6 connectivity.

EoIP

Introduction

Sub-menu: `/interface eoip`

Ethernet over IP (EoIP) Tunneling is a MikroTik RouterOS protocol based on **GRE RFC 1701** that creates an Ethernet tunnel between two routers on top of an IP connection. The EoIP tunnel may run over IPIP tunnel, PPTP tunnel, or any other connection capable of transporting IP.
When the bridging function of the router is enabled, all Ethernet traffic (all Ethernet protocols) will be bridged just as if there where a physical Ethernet interface and cable between the two routers (with bridging enabled). This protocol makes multiple network schemes possible.

Network setups with EoIP interfaces:

- Possibility to bridge LANs over the Internet
- Possibility to bridge LANs over encrypted tunnels
- Possibility to bridge LANs over 802.11b 'ad-hoc' wireless networks

The EoIP protocol encapsulates Ethernet frames in GRE (IP protocol number 47) packets (just like PPTP) and sends them to the remote side of the EoIP tunnel.

Property Description

Property	Description
allow-fast-path (*yes* | *no*; Default: **yes**)	Whether to allow FastPath processing. Must be disabled if IPsec tunneling is used.
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled)**	Address Resolution Protocol mode. - disabled - the interface will not use ARP - enabled - the interface will use ARP - proxy-arp - the interface will use the ARP proxy feature - reply-only - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the "/ip arp" table. No dynamic entries will be automatically stored in the "/ip arp" table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*integer[/time]*; Default: **auto**)	Time interval in which ARP entries should time out.
clamp-tcp-mss (*yes* | *no*; Default: **yes**)	Controls whether to change MSS size for received TCP SYN packets. When enabled, a router will change the MSS size for received TCP SYN packets if the current MSS size exceeds the tunnel interface MTU (taking into account the TCP/IP overhead).The received encapsulated packet will still contain the original MSS, and only after decapsulation the MSS is changed.
comment (*string*; Default:)	Short description of the interface.
disabled (*yes* | *no* ; Default: **no**)	Whether an item is disabled.
dont-fragment (*inherit* | *no*; Default: **no**)	Whether to include DF bit in related packets: *no* - fragment if needed, *inherit* - use Dont Fragment flag of original packet. (Without Dont Fragment: inherit - packet may be fragmented).

dscp (*integer: 0-63*; Default: **inherited**)	DSCP value of packet. Inherited option means that dscp value will be inherited from packet which is going to be encapsulated.
ipsec-secret (*string*; Default:)	When secret is specified, router adds dynamic IPsec peer to remote-address with pre-shared key and policy (by default phase2 uses sha1/aes128cbc).
keepalive (*integer[/time],integer 0..4294967295*; Default: **10s,10**)	Tunnel keepalive parameter sets the time interval in which the tunnel running flag will remain even if the remote end of tunnel goes down. If configured time,retries fail, interface running flag is removed. Parameters are written in following format: `KeepaliveInterval,KeepaliveRetries` where `KeepaliveInterval` is time interval and `KeepaliveRetries` - number of retry attempts. By default keepalive is set to 10 seconds and 10 retries.
l2mtu (*integer; read-only*)	Layer2 Maximum transmission unit. Not configurable for EoIP. MTU in RouterOS
local-address (*IP*; Default:)	Source address of the tunnel packets, local on the router.
loop-protect	
loop-protect-disable-time	
loop-protect-send-interval	
mac-address (*MAC*; Default:)	Media Access Control number of an interface. The address numeration authority IANA allows the use of MAC addresses in the range from **00:00:5E:80:00:00 - 00:00:5E:FF:FF:FF** freely
mtu (*integer*; Default: **auto**)	Layer3 Maximum transmission unit
name (*string*; Default:)	Interface name
remote-address (*IP*; Default:)	IP address of remote end of EoIP tunnel
tunnel-id (*integer: 65536*; Default:)	Unique tunnel identifier, which must match other side of the tunnel

Configuration Examples

Parameter tunnel-id is a method of identifying a tunnel. It must be unique for each EoIP tunnel.

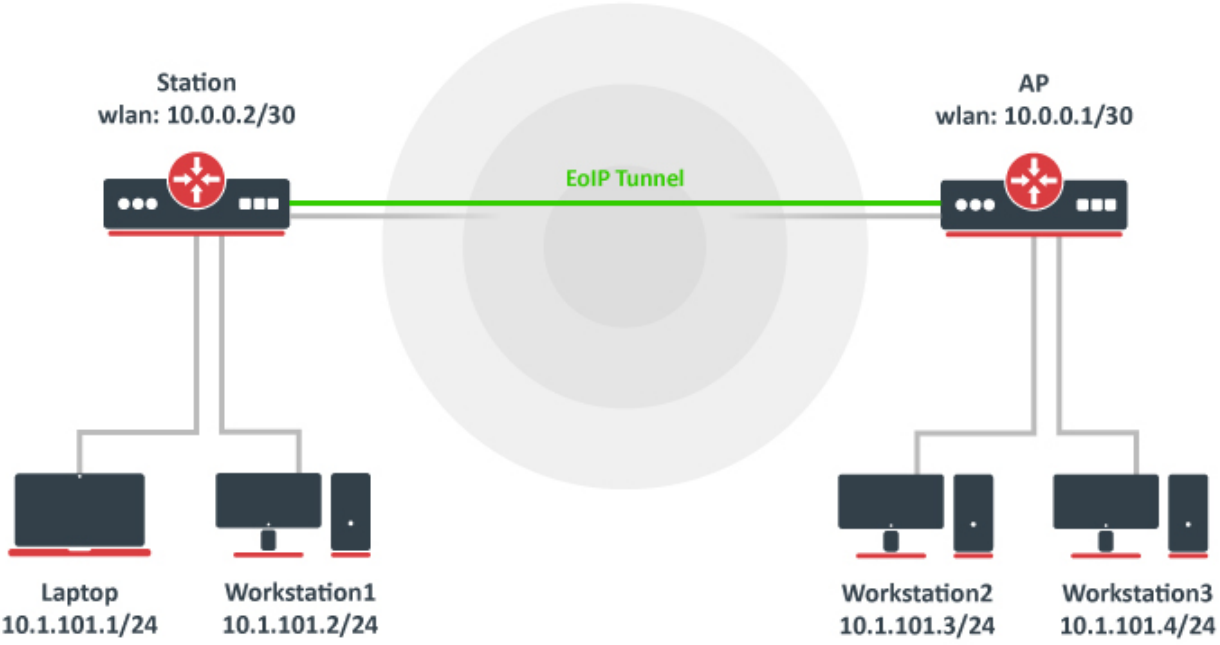
EoIP tunnel adds at least 42 byte overhead (8byte GRE + 14 byte Ethernet + 20 byte IP). MTU should be set to 1500 to eliminate packet fragmentation inside the tunnel (that allows transparent bridging of Ethernet-like networks so that it would be possible to transport full-sized Ethernet frame over the tunnel).

When bridging EoIP tunnels, it is highly recommended to set unique MAC addresses for each tunnel for the bridge algorithms to work correctly. For EoIP interfaces you can use MAC addresses that are in the range from **00:00:5E:80:00:00 - 00:00:5E:FF:FF:FF** , which IANA has reserved for such cases. Alternatively, you can set the second bit of the first byte to modify the auto-assigned address into a 'locally administered address', assigned by the network administrator, and thus use any MAC address, you just need to ensure they are unique between the hosts connected to one bridge.

Example

Let us assume we want to bridge two networks: 'Station' and 'AP'. By using EoIP setup can be made so that Station and AP LANs are in the same Layer2 broadcast domain.

Consider the following setup:



As you know wireless stations cannot be bridged, to overcome this limitation (not involving WDS) we will create an EoIP tunnel over the wireless link and bridge it with interfaces connected to local networks.

We will not cover wireless configuration in this example, let's assume that the wireless link is already established.

At first, we create an EoIP tunnel on our AP:

```
/interface eoip add name="eoip-remote" tunnel-id=0 remote-address=10.0.0.2 disabled=no
```

Verify the interface is created:

```
[admin@AP] > /interface eoip print
Flags: X - disabled; R - running
 0  R name="eoip-remote" mtu=auto actual-mtu=1458 l2mtu=65535 mac-address=FE:A5:6C:3F:26:C5 arp=enabled
      arp-timeout=auto loop-protect=default loop-protect-status=off loop-protect-send-interval=5s
      loop-protect-disable-time=5m local-address=0.0.0.0 remote-address=10.0.0.2 tunnel-id=0
      keepalive=10s,10 dscp=inherit clamp-tcp-mss=yes dont-fragment=no allow-fast-path=yes
```

Station router:

```
/interface eoip add name="eoip-main" tunnel-id=0 remote-address=10.0.0.1 disabled=no
```

Verify the interface is created:

```
[admin@Station] >  /interface eoip print
Flags: X - disabled; R - running
 0  R name="eoip-main" mtu=auto actual-mtu=1458 l2mtu=65535 mac-address=FE:4B:71:05:EA:8B arp=enabled
      arp-timeout=auto loop-protect=default loop-protect-status=off loop-protect-send-interval=5s
      loop-protect-disable-time=5m local-address=0.0.0.0 remote-address=10.0.0.1 tunnel-id=0
      keepalive=10s,10 dscp=inherit clamp-tcp-mss=yes dont-fragment=no allow-fast-path=yes
```

Next, we will bridge local interfaces with EoIP tunnel on our AP. If you already have a local bridge interface, simply add EoIP interface to it:

```
/interface bridge port add bridge=bridge1 interface=eoip-remote
```

The bridge port list should list all local LAN interfaces and the EoIP interface:

```
[admin@AP] > /interface bridge port print
Flags: I - INACTIVE; H - HW-OFFLOAD
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, HORIZON
#    INTERFACE       BRIDGE   HW   PVID  PRIORITY  PATH-COST  INTERNAL-PATH-COST  HORIZON
0  H ether2          bridge1  yes     1  0x80             10                  10  none
1  H ether3          bridge1  yes     1  0x80             10                  10  none
2    eoip-remote     bridge1  yes     1  0x80             10                  10  none
```

On Station router, if you do not have a local bridge interface, create a new bridge and add both EoIP and local LAN interfaces to it:

```
/interface bridge add name=bridge1
/interface bridge port add bridge=bridge1 interface=ether2
/interface bridge port add bridge=bridge1 interface=eoip-main
```

Verify the bridge port section:

```
[admin@Station] > /interface bridge port print
Flags: I - INACTIVE; H - HW-OFFLOAD
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, HORIZON
#    INTERFACE     BRIDGE   HW   PVID  PRIORITY  PATH-COST  INTERNAL-PATH-COST  HORIZON
0  H ether2        bridge1  yes     1  0x80             10                  10  none
2    eoip-main     bridge1  yes     1  0x80             10                  10  none
```

Now both sites are in the same Layer2 broadcast domain. You can set up IP addresses from the same network on both sites.

GRE

Introduction

Sub-menu: `/interface gre`
Standards: RFC1701

GRE (Generic Routing Encapsulation) is a tunneling protocol that was originally developed by Cisco. It can encapsulate a wide variety of protocols creating a virtual point-to-point link.

GRE is the same as IPIP and EoIP which were originally developed as stateless tunnels. This means that if the remote end of the tunnel goes down, all traffic that was routed over the tunnels will get blackholed. To solve this problem, RouterOS has added a 'keepalive' feature for GRE tunnels.

GRE tunnel adds a 24 byte overhead (4-byte gre header + 20-byte IP header). GRE tunnel can forward only IP and IPv6 packets (ethernet type 800 and 86dd). Do not use the "Check gateway" option "arp" when a GRE tunnel is used as a route gateway.

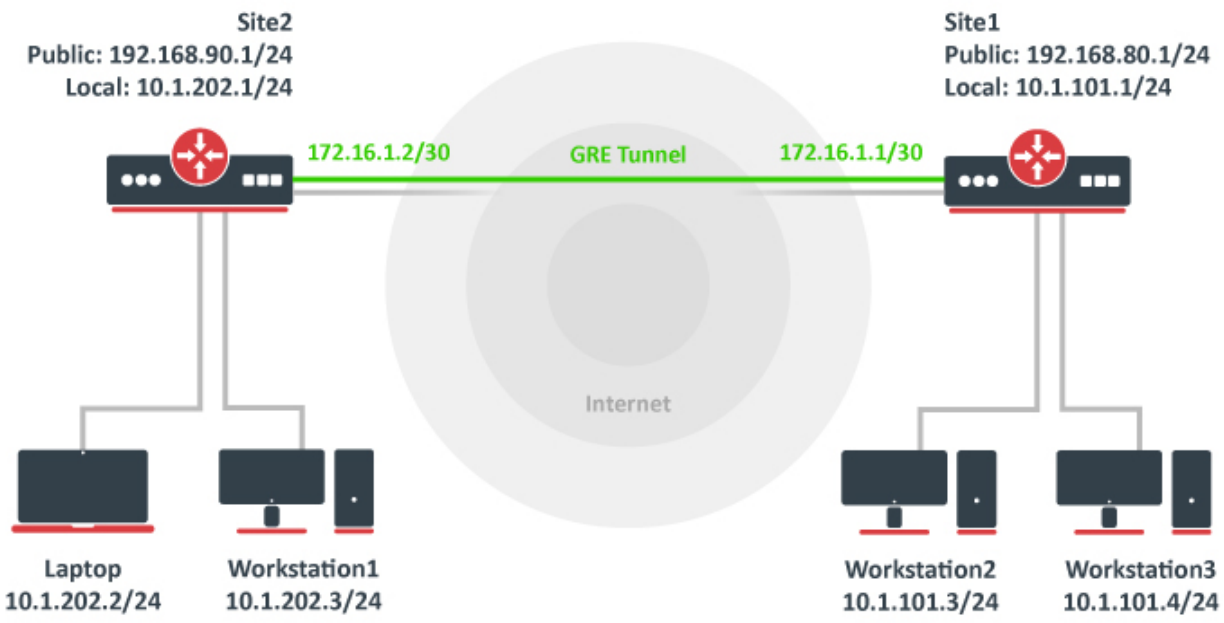
Properties

Property	Description
allow-fast-path (*yes* | *no*; Default: **yes**)	Whether to allow FastPath processing. Must be disabled if IPsec tunneling is used.
clamp-tcp-mss (*yes* | *no*; Default: **yes**)	Controls whether to change MSS size for received TCP SYN packets. When enabled, a router will change the MSS size for received TCP SYN packets if the current MSS size exceeds the tunnel interface MTU (taking into account the TCP/IP overhead). The received encapsulated packet will still contain the original MSS, and only after decapsulation the MSS is changed.
comment (*string*; Default:)	Short description of the tunnel.
disabled (*yes* | *no*; Default: **no**)	Enables/disables tunnel.
dont-fragment (*inherit* | *no*; Default: **no**)	Whether to include DF bit in related packets: *no* - fragment if needed, *inherit* - use Dont Fragment flag of original packet. (Without Dont Fragment: inherit - packet may be fragmented).
dscp (*inherit* | *integer [0-63]*; Default:)	Set dscp value in Gre header to a fixed value or inherit from dscp value taken from tunnelled traffic
ipsec-secret (*string*; Default:)	When secret is specified, router adds dynamic IPsec peer to remote-address with pre-shared key and policy (by default phase2 uses sha1/aes128cbc).
keepalive (*integer [/time],integer 0..4294967295*; Default: **10s,10**)	Tunnel keepalive parameter sets the time interval in which the tunnel running flag will remain even if the remote end of tunnel goes down. If configured time,retries fail, interface running flag is removed. Parameters are written in following format: `KeepaliveInterval,KeepaliveRetries` where KeepaliveInterval is time interval and KeepaliveRetries - number of retry attempts. By default keepalive is set to 10 seconds and 10 retries.
l2mtu (*integer [0..65536]*; Default: **65535**)	Layer2 Maximum transmission unit.

local-address (*IP*; Default: **0.0.0.0**)	IP address that will be used for local tunnel end. If set to 0.0.0.0 then IP address of outgoing interface will be used.
mtu (*integer [0..65536]*; Default: **1476**)	Layer3 Maximum transmission unit.
name (*string*; Default:)	Name of the tunnel.
remote-address (*IP*; Default:)	IP address of remote tunnel end.

Setup example

The goal of this example is to get Layer 3 connectivity between two remote sites over the internet



We have two sites, **Site1** with local network range 10.1.101.0/24 and **Site2** with local network range 10.1.202.0/24.

The first step is to create GRE tunnels. A router on site 1:

```
/interface gre add name=myGre remote-address=192.168.90.1 local-address=192.168.80.1
```

A router on site 2:

```
/interface gre add name=myGre remote-address=192.168.80.1 local-address=192.168.90.1
```

As you can see tunnel configuration is quite simple.

> ⚠ In this example, a keepalive is not configured, so tunnel interface will have a **running** flag even if remote tunnel end is not reachable

Now we just need to set up tunnel addresses and proper routing. A router on site 1:

```
/ip address add address=172.16.1.1/30 interface=myGre
/ip route add dst-address=10.1.202.0/24 gateway=172.16.1.2
```

A router on site 2:

```
/ip address add address=172.16.1.2/30 interface=myGre
/ip route add dst-address=10.1.101.0/24 gateway=172.16.1.1
```

At this point, both sites have Layer 3 connectivity over the GRE tunnel.

IPIP

Summary

Sub-menu: `/interface ipip`
Standards: RFC2003

The IPIP tunneling implementation on the MikroTik RouterOS is RFC 2003 compliant. IPIP tunnel is a simple protocol that encapsulates IP packets in IP to make a tunnel between two routers. The IPIP tunnel interface appears as an interface under the interface list. Many routers, including Cisco and Linux, support this protocol. This protocol makes multiple network schemes possible.

IP tunneling protocol adds the following possibilities to a network setup:

- to tunnel Intranets over the Internet
- to use it instead of source routing

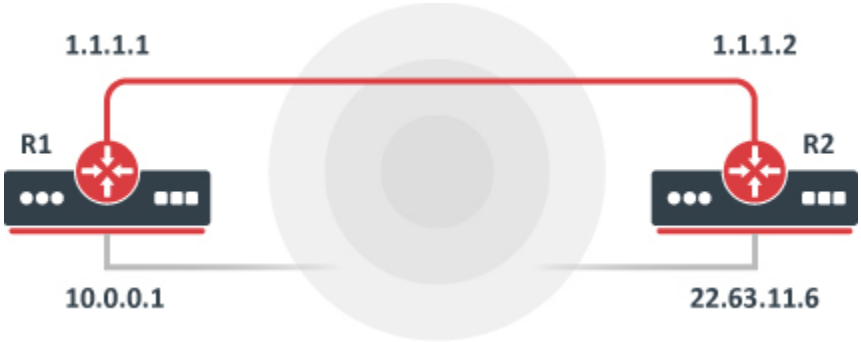
Properties

Property	Description
clamp-tcp-mss (*yes* | *no*; Default: **yes**)	Controls whether to change MSS size for received TCP SYN packets. When enabled, a router will change the MSS size for received TCP SYN packets if the current MSS size exceeds the tunnel interface MTU (taking into account the TCP/IP overhead). The received encapsulated packet will still contain the original MSS, and only after decapsulation the MSS is changed.
dont-fragment (*inherit* | *no*; Default: **no**)	Whether to include DF bit in related packets: *no* - fragment if needed, *inherit* - use Dont Fragment flag of original packet. (Without Dont Fragment: inherit - packet may be fragmented).
dscp (*inherit* | *integer [0-63]*; Default:)	Set dscp value in IPIP header to a fixed value or inherit from dscp value taken from tunnelled traffic
ipsec-secret (*string*; Default:)	When secret is specified, router adds dynamic ipsec peer to remote-address with pre-shared key and policy with default values (by default phase2 uses sha1/aes128cbc).
local-address (*IP*; Default:)	IP address on a router that will be used by IPIP tunnel
mtu (*integer*; Default: **1500**)	Layer3 Maximum transmission unit
keepalive (*integer [/time],integer 0..4294967295*; Default: **10s,10**)	Tunnel keepalive parameter sets the time interval in which the tunnel running flag will remain even if the remote end of tunnel goes down. If configured time,retries fail, interface running flag is removed. Parameters are written in following format: `KeepaliveInterval,KeepaliveRetries` where KeepaliveInterval is time interval and KeepaliveRetries - number of retry attempts. By default keepalive is set to 10 seconds and 10 retries.
name (*string*; Default:)	Interface name
remote-address (*IP*; Default:)	IP address of remote end of IPIP tunnel

⚠ There is no authentication or 'state' for this interface. The bandwidth usage of the interface may be monitored with the monitor feature from the interface menu.

Example

Suppose we want to add an IPIP tunnel between routers R1 and R2:



At first, we need to configure IPIP interfaces and then add IP addresses to them.

The configuration for router **R1** is as follows:

```
[admin@MikroTik] interface ipip> add
local-address: 10.0.0.1
remote-address: 22.63.11.6
[admin@MikroTik] interface ipip> print
Flags: X - disabled, R - running
# NAME MTU LOCAL-ADDRESS REMOTE-ADDRESS
0 X ipip1 1480 10.0.0.1 22.63.11.6

[admin@MikroTik] interface ipip> en 0
[admin@MikroTik] interface ipip> /ip address add address=1.1.1.1/24 interface=ipip1
```

The configuration of the **R2** is shown below:

```
[admin@MikroTik] interface ipip> add local-address=22.63.11.6 remote-address=10.
0.0.1
[admin@MikroTik] interface ipip> print
Flags: X - disabled, R - running
# NAME MTU LOCAL-ADDRESS REMOTE-ADDRESS
0 X ipip1 1480 22.63.11.6 10.0.0.1

[admin@MikroTik] interface ipip> enable 0
[admin@MikroTik] interface ipip> /ip address add address=1.1.1.2/24 interface=ipip1
```

Now both routers can ping each other:

```
[admin@MikroTik] interface ipip> /ping 1.1.1.2
1.1.1.2 64 byte ping: ttl=64 time=24 ms
1.1.1.2 64 byte ping: ttl=64 time=19 ms
1.1.1.2 64 byte ping: ttl=64 time=20 ms
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 19/21.0/24 ms
[admin@MikroTik] interface ipip>
```

IPsec

Introduction

Internet Protocol Security (IPsec) is a set of protocols defined by the Internet Engineering Task Force (IETF) to secure packet exchange over unprotected IP/IPv6 networks such as the Internet.

IPsec protocol suite can be divided into the following groups:

- **Internet Key Exchange (IKE)** protocols. Dynamically generates and distributes cryptographic keys for AH and ESP.
- **Authentication Header (AH)** RFC 4302
- **Encapsulating Security Payload (ESP)** RFC 4303

Internet Key Exchange Protocol (IKE)

The Internet Key Exchange (IKE) is a protocol that provides authenticated keying material for the Internet Security Association and Key Management Protocol (ISAKMP) framework. There are other key exchange schemes that work with ISAKMP, but IKE is the most widely used one. Together they provide means for authentication of hosts and automatic management of security associations (SA).

Most of the time IKE daemon is doing nothing. There are two possible situations when it is activated:

There is some traffic caught by a policy rule which needs to become encrypted or authenticated, but the policy doesn't have any SAs. The policy notifies the IKE daemon about that, and the IKE daemon initiates a connection to a remote host. IKE daemon responds to remote connection. In both cases, peers establish a connection and execute 2 phases:

- **Phase 1** - The peers agree upon algorithms they will use in the following IKE messages and authenticate. The keying material used to derive keys for all SAs and to protect following ISAKMP exchanges between hosts is generated also. This phase should match the following settings:
    - authentication method
    - DH group
    - encryption algorithm
    - exchange mode
    - hash algorithm
    - NAT-T
    - DPD and lifetime (optional)
- **Phase 2** - The peers establish one or more SAs that will be used by IPsec to encrypt data. All SAs established by the IKE daemon will have lifetime values (either limiting time, after which SA will become invalid, or the amount of data that can be encrypted by this SA, or both). This phase should match the following settings:
    - IPsec protocol
    - mode (tunnel or transport)
    - authentication method
    - PFS (DH) group
    - lifetime

There are two lifetime values - soft and hard. When SA reaches its soft lifetime threshold, the IKE daemon receives a notice and starts another phase 2 exchange to replace this SA with a fresh one. If SA reaches a hard lifetime, it is discarded.

Phase 1 is not re-keyed if DPD is disabled when the lifetime expires, only phase 2 is re-keyed. To force phase 1 re-key, enable DPD.

PSK authentication was known to be vulnerable against Offline attacks in "aggressive" mode, however recent discoveries indicate that offline attack is possible also in the case of "main" and "ike2" exchange modes. A general recommendation is to avoid using the PSK authentication method.

IKE can optionally provide a Perfect Forward Secrecy (PFS), which is a property of key exchanges, that, in turn, means for IKE that compromising the long term phase 1 key will not allow to easily gain access to all IPsec data that is protected by SAs established through this phase 1. It means an additional keying material is generated for each phase 2.

The generation of keying material is computationally very expensive. Exempli Gratia, the use of the modp8192 group can take several seconds even on a very fast computer. It usually takes place once per phase 1 exchange, which happens only once between any host pair and then is kept for a long time. PFS adds this expensive operation also to each phase 2 exchange.

Diffie-Hellman Groups

Diffie-Hellman (DH) key exchange protocol allows two parties without any initial shared secret to create one securely. The following Modular Exponential (MODP) and ECP Diffie-Hellman (also known as "Oakley") Groups are supported:

Diffie-Hellman Group	Name	Reference
Group 1	768 bits MODP group	RFC 2409
Group 2	1024 bits MODP group	RFC 2409
Group 5	1536 bits MODP group	RFC 3526
Group 14	2048 bits MODP group	RFC 3526
Group 15	3072 bits MODP group	RFC 3526
Group 16	4096 bits MODP group	RFC 3526
Group 17	6144 bits MODP group	RFC 3526
Group 18	8192 bits MODP group	RFC 3526
Group 19	256 bits random ECP group	RFC 5903
Group 20	384 bits random ECP group	RFC 5903
Group 21	521 bits random ECP group	RFC 5903

More on standards can be found here.

Larger DH groups offer better security but require more CPU power. Here are a few commonly used DH groups with varying levels of security and CPU impact:

DH Group 14 (2048-bit) - Provides a reasonable balance between security and CPU usage. It offers 2048-bit key exchange, which is considered secure for most applications today and is widely supported.

DH Group 5 (1536-bit) - Offers a slightly lower level of security compared to DH Group 14 but has a lower CPU impact due to the smaller key size. It is still considered secure for many scenarios.

DH Group 2 (1024-bit) - Should be used with caution because it provides the least security among commonly used groups. It has a lower CPU impact but is susceptible to attacks, especially as computational power increases. It's generally not recommended for new deployments.

For optimal security, it's advisable to use **DH Group 19**. It's considered fast and secure. However, DH Group 14 might give large load for your router, DH Group 5 can be a reasonable compromise between security and performance. DH Group 2 should generally be avoided unless you have legacy devices that require it.

The correct way of calculating security for your network infrastructure would be to choose how many bits of security you want and you could see how long it would require to decrypt your data, then you have to choose algorithms. Please see information for reference https://www.keylength.com/en/4/

IKE Traffic

To avoid problems with IKE packets hit some SPD rule and require to encrypt it with not yet established SA (that this packet perhaps is trying to establish), locally originated packets with UDP source port 500 are not processed with SPD. The same way packets with UDP destination port 500 that are to be delivered locally are not processed in incoming policy checks.

Setup Procedure

To get IPsec to work with automatic keying using IKE-ISAKMP you will have to configure policy, peer, and proposal (optional) entries.

> IPsec is very sensitive to time changes. If both ends of the IPsec tunnel are not synchronizing time equally(for example, different NTP servers not updating time with the same timestamp), tunnels will break and will have to be established again.

EAP Authentication methods

Outer Auth	Inner Auth
EAP-GTC	
EAP-MD5	
EAP-MSCHAPv2	
EAP-PEAPv0	EAP-MSCHAPv2 EAP-GPSK EAP-GTC EAP-MD5 EAP-TLS
EAP-SIM	
EAP-TLS	
EAP-TTLS	PAP CHAP MS-CHAP MS-CHAPv2 EAP-MSCHAPv2 EAP-GTC EAP-MD5 EAP-TLS

EAP-TLS on Windows is called "Smart Card or other certificates".

> Using ed25519 - authentication is not yet supported.

Authentication Header (AH)

AH is a protocol that provides authentication of either all or part of the contents of a datagram through the addition of a header that is calculated based on the values in the datagram. What parts of the datagram are used for the calculation, and the placement of the header depends on whether tunnel or transport mode is used.

The presence of the AH header allows to verify the integrity of the message but doesn't encrypt it. Thus, AH provides authentication but not privacy. Another protocol (ESP) is considered superior, it provides data privacy and also its own authentication method.

RouterOS supports the following authentication algorithms for AH:

- SHA2 (256, 512)
- SHA1
- MD5

Transport mode

In transport mode, the AH header is inserted after the IP header. IP data and header is used to calculate authentication value. IP fields that might change during transit, like TTL and hop count, are set to zero values before authentication.

Tunnel mode

In tunnel mode, the original IP packet is encapsulated within a new IP packet. All of the original IP packets are authenticated.

Encapsulating Security Payload (ESP)

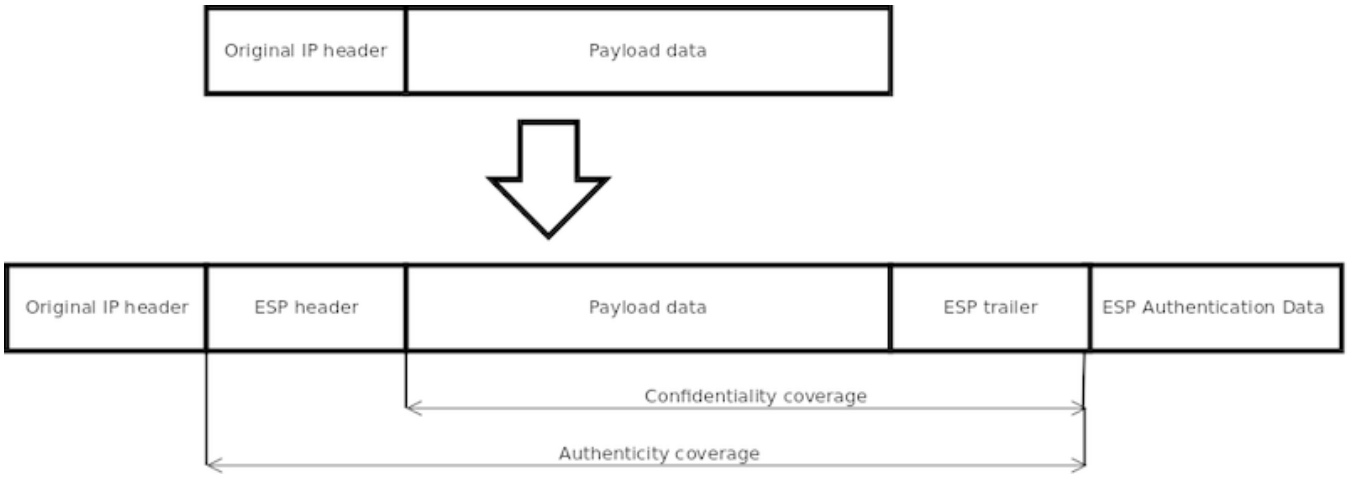
Encapsulating Security Payload (ESP) uses shared key encryption to provide data privacy. ESP also supports its own authentication scheme like that used in AH.

ESP packages its fields in a very different way than AH. Instead of having just a header, it divides its fields into three components:

- **ESP Header** - Comes before the encrypted data and its placement depends on whether ESP is used in transport mode or tunnel mode.
- **ESP Trailer** - This section is placed after the encrypted data. It contains padding that is used to align the encrypted data.
- **ESP Authentication Data** - This field contains an Integrity Check Value (ICV), computed in a manner similar to how the AH protocol works, for when ESP's optional authentication feature is used.

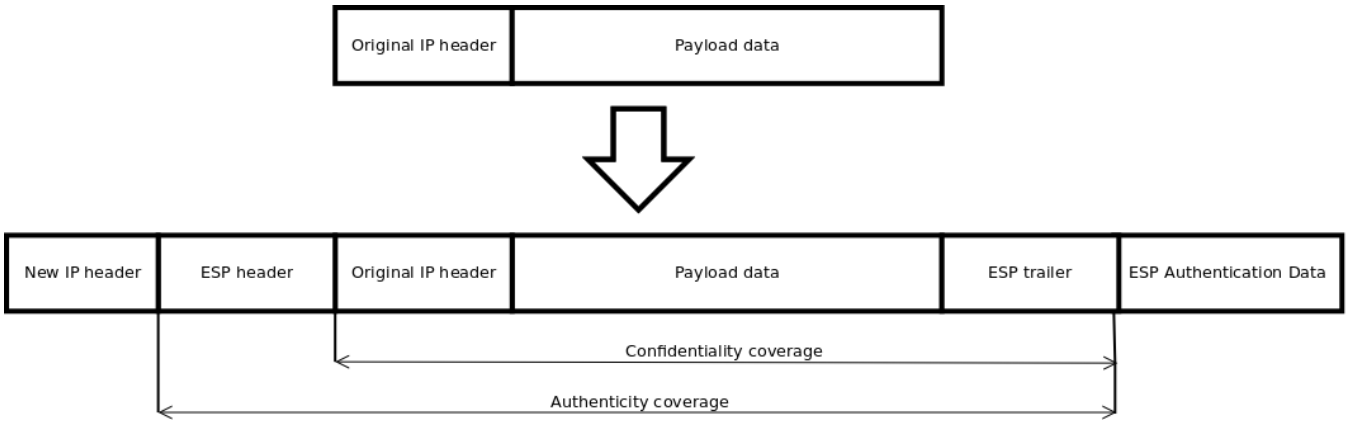
Transport mode

In transport mode, the ESP header is inserted after the original IP header. ESP trailer and authentication value are added to the end of the packet. In this mode only the IP payload is encrypted and authenticated, the IP header is not secured.



Tunnel mode

In tunnel mode, an original IP packet is encapsulated within a new IP packet thus securing IP payload and IP header.



Encryption algorithms

RouterOS ESP supports various encryption and authentication algorithms.

Authentication:

- **MD5**
- **SHA1**
- **SHA2 (256-bit, 512-bit)**

Encryption:

- **AES** - 128-bit, 192-bit, and 256-bit key AES-CBC, AES-CTR, and AES-GCM algorithms;
- **Blowfish** - added since v4.5
- **Twofish** - added since v4.5
- **Camellia** - 128-bit, 192-bit, and 256-bit key Camellia encryption algorithm added since v4.5
- **DES** - 56-bit DES-CBC encryption algorithm;
- **3DES** - 168-bit DES encryption algorithm;

Hardware acceleration

Hardware acceleration allows doing a faster encryption process by using a built-in encryption engine inside the CPU.

List of devices with hardware acceleration is available here

CPU	DES and 3DES				AES-CBC				AES-CTR				AES-GCM			
	MD5	SHA1	SHA256	SHA512	MD5	SHA1	SHA256	SHA512	MD5	SHA1	SHA256	SHA512	MD5	SHA1	SHA256	SHA
88F7040	no	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes
AL21400	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
AL32400	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
AL52400	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
AL73400	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IPQ-4018 / IPQ-4019	no	yes	yes	no	no	yes*	yes*	no	no	yes*	yes*	no	no	no	no	no
IPQ-5018	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes	no	no	no	no	no
IPQ-6010	no	no	no	no	no	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes
IPQ-8064	no	yes	yes	no	no	yes*	yes*	no	no	yes*	yes*	no	no	no	no	no
MT7621A	yes****	yes****	yes****	no	yes	yes	yes	no	no	no	no	no	no	no	no	no
P1023N SN5CFB	no	no	no	no	yes**	yes**	yes**	yes**	no	no	no	no	no	no	no	no
P202AS SE2KFB	yes	yes	yes	no	yes	yes	yes	yes	no	no	no	no	no	no	no	no

PPC460 GT	no	no	no	no	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***	no	no	no	no
TLR4 (TILE)	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes	no	no	no	no	no
x86 (AES-NI)	no	no	no	no	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***	yes***

* supported only 128 bit and 256 bit key sizes

** only manufactured since 2016, serial numbers that begin with number 5 and 7

*** AES-CBC and AES-CTR only encryption is accelerated, hashing done in software

**** DES is not supported, only 3DES and AES-CBC

IPsec throughput results of various encryption and hash algorithm combinations are published on the MikroTik products page.

Policies

The policy table is used to determine whether security settings should be applied to a packet.

Properties

Property	Description
action (*discard* / *encrypt* / *none*; Default: **encrypt**)	Specifies what to do with the packet matched by the policy. • none - pass the packet unchanged. • discard - drop the packet. • encrypt - apply transformations specified in this policy and it's SA.
comment (*string*; Default:)	Short description of the policy.
disabled (*yes* / *no*; Default: **no**)	Whether a policy is used to match packets.
dst-address (*IP/IPv6 prefix*; Default: **0.0.0.0/32**)	Destination address to be matched in packets. Applicable when tunnel mode (tunnel=yes) or template (template=yes) is used.
dst-port (*integer:0..65535* / *any*; Default: **any**)	Destination port to be matched in packets. If set to any all ports will be matched.
group (*string*; Default: **default**)	Name of the policy group to which this template is assigned.
ipsec-protocols (*ah* / *esp*; Default: **esp**)	Specifies what combination of Authentication Header and Encapsulating Security Payload protocols you want to apply to matched traffic.
level (*require* / *unique* / *use*; Default: **require**)	Specifies what to do if some of the SAs for this policy cannot be found: • use - skip this transform, do not drop the packet, and do not acquire SA from IKE daemon; • require - drop the packet and acquire SA; • unique - drop the packet and acquire a unique SA that is only used with this particular policy. It is used in setups where multiple clients can sit behind one public IP address (clients behind NAT).
peer (*string*; Default:)	Name of the peer on which the policy applies.
proposal (*string*; Default: **default**)	Name of the proposal template that will be sent by IKE daemon to establish SAs for this policy.
protocol (*all* / *egp* / *ggp*/ *icmp* / *igmp* / ...; Default: **all**)	IP packet protocol to match.
src-address (*ip/ipv6 prefix*; Default: **0.0.0.0/32**)	Source address to be matched in packets. Applicable when tunnel mode (tunnel=yes) or template (template=yes) is used.
src-port (*any* / *integer:0..65535*; Default: **any**)	Source port to be matched in packets. If set to any all ports will be matched.

template (*yes* | *no*; Default: **no**)	Creates a template and assigns it to a specified policy group. Following parameters are used by template: • group - name of the policy group to which this template is assigned; • src-address, dst-address - Requested subnet must match in both directions(for example 0.0.0.0/0 to allow all); • protocol - protocol to match, if set to all, then any protocol is accepted; • proposal - SA parameters used for this template; • level - useful when unique is required in setups with multiple clients behind NAT.
tunnel (*yes* | *no*; Default: **no**)	Specifies whether to use tunnel mode.

Read-only properties

Property	Description
active (*yes* | *no*)	Whether this policy is currently in use.
default (*yes* | *no*)	Whether this is a default system entry.
dynamic (*yes* | *no*)	Whether this is a dynamically added or generated entry.
invalid (*yes* | *no*)	Whether this policy is invalid - the possible cause is a duplicate policy with the same src-address and dst-address.
ph2-count (*integer*)	A number of active phase 2 sessions associated with the policy.
ph2-state (*expired* | *no-phase2* | *established*)	Indication of the progress of key establishing.
sa-dst-address (*ip/ipv6 address*; Default: **::**)	SA destination IP/IPv6 address (remote peer).
sa-src-address (*ip/ipv6 address*; Default: **::**)	SA source IP/IPv6 address (local peer).

Policy order is important starting from v6.40. Now it works similarly to firewall filters where policies are executed from top to bottom (priority parameter is removed).

All packets are IPIP encapsulated in tunnel mode, and their new IP header's src-address and dst-address are set to sa-src-address and sa-dst-address values of this policy. If you do not use tunnel mode (id est you use transport mode), then only packets whose source and destination addresses are the same as sa-src-address and sa-dst-address can be processed by this policy. Transport mode can only work with packets that originate at and are destined for IPsec peers (hosts that established security associations). To encrypt traffic between networks (or a network and a host) you have to use tunnel mode.

Statistics

This menu shows various IPsec statistics and errors.

Read-only properties

Property	Description
in-errors (*integer*)	All inbound errors that are not matched by other counters.
in-buffer-errors (*integer*)	No free buffer.
in-header-errors (*integer*)	Header error.
in-no-states (*integer*)	No state is found i.e. either inbound SPI, address, or IPsec protocol at SA is wrong.
in-state-protocol-errors (*integer*)	Transformation protocol-specific error, for example, SA key is wrong or hardware accelerator is unable to handle the number of packets.
in-state-mode-errors (*integer*)	Transformation mode-specific error.

in-state-sequence-errors (*integer*)	A sequence number is out of a window.
in-state-expired (*integer*)	The state is expired.
in-state-mismatches (*integer*)	The state has a mismatched option, for example, the UDP encapsulation type is mismatched.
in-state-invalid (*integer*)	The state is invalid.
in-template-mismatches (*integer*)	No matching template for states, e.g. inbound SAs are correct but the SP rule is wrong. A possible cause is a mismatched sa-source or sa-destination address.
in-no-policies (*integer*)	No policy is found for states, e.g. inbound SAs are correct but no SP is found.
in-policy-blocked (*integer*)	Policy discards.
in-policy-errors (*integer*)	Policy errors.
out-errors (*integer*)	All outbound errors that are not matched by other counters.
out-bundle-errors (*integer*)	Bundle generation error.
out-bundle-check-errors (*integer*)	Bundle check error.
out-no-states (*integer*)	No state is found.
out-state-protocol-errors (*integer*)	Transformation protocol specific error.
out-state-mode-errors (*integer*)	Transformation mode-specific error.
out-state-sequence-errors (*integer*)	Sequence errors, for example, sequence number overflow.
out-state-expired (*integer*)	The state is expired.
out-policy-blocked (*integer*)	Policy discards.
out-policy-dead (*integer*)	The policy is dead.
out-policy-errors (*integer*)	Policy error.

Proposals

Proposal information that will be sent by IKE daemons to establish SAs for certain policies.

Properties

Property	Description
auth-algorithms (*md5|null|sha1|sha256|sha512*; Default: **sha1**)	Allowed algorithms for authorization. SHA (Secure Hash Algorithm) is stronger but slower. MD5 uses a 128-bit key, sha1-160bit key.
comment (*string*; Default:)	
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled.
enc-algorithms (*null|des|3des|aes-128-cbc|aes-128-cbc|aes-128gcm|aes-192-cbc|aes-192-ctr|aes-192-gcm|aes-256-cbc|aes-256-ctr|aes-256-gcm|blowfish|camellia-128|camellia-192|camellia-256|twofish*; Default: **aes-256-cbc,aes-192-cbc,aes-128-cbc**)	Allowed algorithms and key lengths to use for SAs.
lifetime (*time*; Default: **30m**)	How long to use SA before throwing it out.

name (*string*; Default:)	
pfs-group (*ecp256* | *ecp384* | *ecp521* | *modp768* | *modp1024* | *modp1536* | *modp2048* | *modp3072* | *modp4096* | *modp6144* | *modp8192* | *none*; Default: **modp1024**)	The diffie-Helman group used for Perfect Forward Secrecy.

Read-only properties

Property	Description
default (*yes* | *no*)	Whether this is a default system entry.

Groups

In this menu, it is possible to create additional policy groups used by policy templates.

Properties

Property	Description
name (*string*; Default:)	
comment (*string*; Default:)	

Peers

Peer configuration settings are used to establish connections between IKE daemons. This connection then will be used to negotiate keys and algorithms for SAs. Exchange mode is the only unique identifier between the peers, meaning that there can be multiple peer configurations with the same remote-address as long as a different exchange-mode is used.

Properties

Property	Description
address (*IP/IPv6 Prefix*; Default: **0.0.0.0/0**)	If the remote peer's address matches this prefix, then the peer configuration is used in authentication and establishment of **Phase 1**. If several peer's addresses match several configuration entries, the most specific one (i.e. the one with the largest netmask) will be used.
comment (*string*; Default:)	Short description of the peer.
disabled (*yes* | *no*; Default: **no**)	Whether peer is used to matching remote peer's prefix.
exchange-mode (*aggressive* | *base* | *main* | *ike2*; Default: **main**)	Different ISAKMP phase 1 exchange modes according to RFC 2408. the **main** mode relaxes rfc2409 section 5.4, to allow pre-shared-key authentication in the main mode. ike2 mode enables Ikev2 RFC 7296. Parameters that are ignored by IKEv2 proposal-check, compatibility-options, lifebytes, dpd-maximum-failures, nat-traversal.
local-address (*IP/IPv6 Address*; Default:)	Routers local address on which Phase 1 should be bounded to.
name (*string*; Default:)	
passive (*yes* | *no*; Default: **no**)	When a passive mode is enabled will wait for a remote peer to initiate an IKE connection. The enabled passive mode also indicates that the peer is xauth responder, and disabled passive mode - xauth initiator. When a passive mode is a disabled peer will try to establish not only phase1 but also phase2 automatically, if policies are configured or created during the phase1.

port (*integer:0..65535*; Default: **500**)	Communication port used (when a router is an initiator) to connect to remote peer in cases if remote peer uses the non-default port.
profile (*string*; Default: **default**)	Name of the profile template that will be used during IKE negotiation.
send-initial-contact (*yes* | *no*; Default: **yes**)	Specifies whether to send "initial contact" IKE packet or wait for remote side, this packet should trigger the removal of old peer SAs for current source address. Usually, in road warrior setups clients are initiators and this parameter should be set to no. Initial contact is not sent if modecfg or xauth is enabled for ikev1.

Read-only properties

Property	Description
dynamic (*yes* | *no*)	Whether this is a dynamically added entry by a different service (e.g L2TP).
responder (*yes* | *no*)	Whether this peer will act as a responder only (listen to incoming requests) and not initiate a connection.

Profiles

Profiles define a set of parameters that will be used for IKE negotiation during Phase 1. These parameters may be common with other peer configurations.

Properties

Property	Description
dh-group (*modp768* | *modp1024* | *modp1536* | *modp2048* | *modp3072* | *modp4096* | *modp6144* | *modp8192* | *ecp256* | *ecp384* | *ecp521*; Default: **modp1024,modp2048**)	Diffie-Hellman group (cipher strength).
dpd-interval (*time* | *disable-dpd*; Default: **2m**)	Dead peer detection interval. If set to disable-dpd, dead peer detection will not be used.
dpd-maximum-failures (*integer: 1..100*; Default: **5**)	Maximum count of failures until peer is considered to be dead. Applicable if DPD is enabled.
enc-algorithm (*3des* | *aes-128* | *aes-192* | *aes-256* | *blowfish* | *camellia-128* | *camellia-192* | *camellia-256* | *des*; Default: **aes-128**)	List of encryption algorithms that will be used by the peer.
hash-algorithm (*md5* | *sha1* | *sha256* | *sha512*; Default: **sha1**)	Hashing algorithm. SHA (Secure Hash Algorithm) is stronger, but slower. MD5 uses 128-bit key, sha1-160bit key.
lifebytes (*Integer: 0..4294967295*; Default: **0**)	Phase 1 lifebytes is used only as administrative value which is added to proposal. Used in cases if remote peer requires specific lifebytes value to establish phase 1.
lifetime (*time*; Default: **1d**)	Phase 1 lifetime: specifies how long the SA will be valid.
name (*string*; Default:)	
nat-traversal (*yes* | *no*; Default: **yes**)	Use Linux NAT-T mechanism to solve IPsec incompatibility with NAT routers between IPsec peers. This can only be used with ESP protocol (AH is not supported by design, as it signs the complete packet, including the IP header, which is changed by NAT, rendering AH signature invalid). The method encapsulates IPsec ESP traffic into UDP streams in order to overcome some minor issues that made ESP incompatible with NAT.

proposal-check (*claim* | *exact* | *obey* | *strict*; Default: **obey**)	Phase 2 lifetime check logic: • claim - take shortest of proposed and configured lifetimes and notify initiator about it • exact - require lifetimes to be the same • obey - accept whatever is sent by an initiator • strict - if the proposed lifetime is longer than the default then reject the proposal otherwise accept a proposed lifetime

Identities

Identities are configuration parameters that are specific to the remote peer. The main purpose of identity is to handle authentication and verify the peer's integrity.

Properties

Property	Description
auth-method (*digital-signature* | *eap* | *eap-radius* | *pre-shared-key* | *pre-shared-key-xauth* | *rsa-key* | *rsa-signature-hybrid*; Default: **pre-shared-key**)	Authentication method: • digital-signature - authenticate using a pair of RSA certificates; • eap - IKEv2 EAP authentication for initiator (peer with a netmask of /32). Must be used together with eap-methods; • eap-radius - IKEv2 EAP RADIUS passthrough authentication for the responder (RFC 3579). A server certificate in this case is required. If a server certificate is not specified then only clients supporting EAP-only (RFC 5998) will be able to connect. Note that the EAP method should be compatible with EAP-only; • pre-shared-key - authenticate by a password (pre-shared secret) string shared between the peers (not recommended since an offline attack on the pre-shared key is possible); • rsa-key - authenticate using an RSA key imported in keys menu. Only supported in IKEv1; • pre-shared-key-xauth - authenticate by a password (pre-shared secret) string shared between the peers + XAuth username and password. Only supported in IKEv1; • rsa-signature-hybrid - responder certificate authentication with initiator XAuth. Only supported in IKEv1.
certificate (*string*; Default:)	Name of a certificate listed in System/Certificates (signing packets; the certificate must have the private key). Applicable if digital signature authentication method (auth-method=digital-signature) or EAP (auth-method=eap) is used.
comment (*string*; Default:)	Short description of the identity.
disabled (*yes* | *no*; Default: **no**)	Whether identity is used to match remote peers.
eap-methods (*eap-mschapv2* | *eap-peap* | *eap-tls* | *eap-ttls*; Default: **eap-tls**)	All EAP methods requires whole certificate chain including intermediate and root CA certificates to be present in System/Certificates menu. Also, the username and password (if required by the authentication server) must be specified. Multiple EAP methods may be specified and will be used in a specified order. Currently supported EAP methods: • eap-mschapv2; • eap-peap - also known as PEAPv0/EAP-MSCHAPv2; • eap-tls - requires additional client certificate specified under certificate parameter; • eap-ttls.
generate-policy (*no* | *port-override* | *port-strict*; Default: **no**)	Allow this peer to establish SA for non-existing policies. Such policies are created dynamically for the lifetime of SA. Automatic policies allows, for example, to create IPsec secured L2TP tunnels, or any other setup where remote peer's IP address is not known at the configuration time. • no - do not generate policies; • port-override - generate policies and force policy to use **any** port (old behavior); • port-strict - use ports from peer's proposal, which should match peer's policy.
key (*string*; Default:)	Name of the private key from keys menu. Applicable if RSA key authentication method (auth-method=rsa-key) is used.

match-by (*remote-id* | *certificate*; Default: **remote-id**)	Defines the logic used for peer's identity validation. • remote-id - will verify the peer's ID according to remote-id setting. • certificate will verify the peer's certificate with what is specified under remote-certificate setting.
mode-config (*none* | **request-only* | *string*; Default: **none**)	Name of the configuration parameters from mode-config menu. When parameter is set mode-config is enabled.
my-id (*auto* | *address* | *fqdn* | *user-fqdn* | *key-id*; Default: **auto**)	On initiator, this controls what ID_i is sent to the responder. On responder, this controls what ID_r is sent to the initiator. In IKEv2, responder also expects this ID in received ID_r from initiator. • auto - tries to use correct ID automatically: IP for pre-shared key, SAN (DN if not present) for certificate based connections; • address - IP address is used as ID; • dn - the binary Distinguished Encoding Rules (DER) encoding of an ASN.1 X.500 Distinguished Name; • fqdn - fully qualified domain name; • key-id - use the specified key ID for the identity; • user fqdn - specifies a fully-qualified username string, for example, "user@domain.com".
notrack-chain (*string*; Default:)	Adds IP/Firewall/Raw rules matching IPsec policy to a specified chain. Use together with generate-policy.
password (*string*; Default:)	XAuth or EAP password. Applicable if pre-shared key with XAuth authentication method (auth-method=pre-shared-key-xauth) or EAP (auth-method=eap) is used.
peer (*string*; Default:)	Name of the peer on which the identity applies.
policy-template-group (*none* | *string*; Default: **default**)	If generate-policy is enabled, traffic selectors are checked against templates from the same group. If none of the templates match, Phase 2 SA will not be established.
remote-certificate (*string*; Default:)	Name of a certificate (listed in System/Certificates) for authenticating the remote side (validating packets; no private key required). If a remote-certificate is not specified then the received certificate from a remote peer is used and checked against CA in the certificate menu. Proper CA must be imported in a certificate store. If remote-certificate and match-by=certificate is specified, only the specific client certificate will be matched. Applicable if digital signature authentication method (auth-method=digital-signature) is used.
remote-id (*auto* | *fqdn* | *user-fqdn* | *key-id* | *ignore*; Default: **auto**)	This parameter controls what ID value to expect from the remote peer. Note that all types except for ignoring will verify remote peer's ID with a received certificate. In case when the peer sends the certificate name as its ID, it is checked against the certificate, else the ID is checked against Subject Alt. Name. • auto - accept all ID's; • address - IP address is used as ID; • dn - the binary Distinguished Encoding Rules (DER) encoding of an ASN.1 X.500 Distinguished Name; • fqdn - fully qualified domain name. Only supported in IKEv2; • user fqdn - a fully-qualified username string, for example, "user@domain.com". Only supported in IKEv2; • key-id - specific key ID for the identity. Only supported in IKEv2; • ignore - do not verify received ID with certificate (dangerous). * Wildcard key ID matching **is not supported**, for example remote-id="key-id:CN=*.domain.com"
remote-key (*string*; Default:)	Name of the public key from keys menu. Applicable if RSA key authentication method (auth-method=rsa-key) is used.
secret (*string*; Default:)	Secret string. If it starts with '0x', it is parsed as a hexadecimal value. Applicable if pre-shared key authentication method (auth-method=pre-shared-key and auth-method=pre-shared-key-xauth) is used.
username (*string*; Default:)	XAuth or EAP username. Applicable if pre-shared key with XAuth authentication method (auth-method=pre-shared-key-xauth) or EAP (auth-method=eap) is used.

Read only properties

Property	Description
dynamic (*yes* | *no*)	Whether this is a dynamically added entry by a different service (e.g L2TP).

Active Peers

This menu provides various statistics about remote peers that currently have established phase 1 connection.

Read only properties

Property	Description
dynamic-address (*ip/ipv6 address*)	Dynamically assigned an IP address by mode config
last-seen (*time*)	Duration since the last message received by this peer.
local-address (*ip/ipv6 address*)	Local address on the router used by this peer.
natt-peer (*yes* | *no*)	Whether NAT-T is used for this peer.
ph2-total (*integer*)	The total amount of active IPsec security associations.
remote-address (*ip/ipv6 address*)	The remote peer's ip/ipv6 address.
responder (*yes* | *no*)	Whether the connection is initiated by a remote peer.
rx-bytes (*integer*)	The total amount of bytes received from this peer.
rx-packets (*integer*)	The total amount of packets received from this peer.
side (*initiator* | *responder*)	Shows which side initiated the Phase1 negotiation.
state (*string*)	State of phase 1 negotiation with the peer. For example, when phase1 and phase 2 are negotiated it will show state "established".
tx-bytes (*integer*)	The total amount of bytes transmitted to this peer.
tx-packets (*integer*)	The total amount of packets transmitted to this peer.
uptime (*time*)	How long peers are in an established state.

Commands

Property	Description
kill-connections ()	Manually disconnects all remote peers.

Mode configs

ISAKMP and IKEv2 configuration attributes are configured in this menu.

Properties

Property	Description
address (*none* | *string*; Default:)	Single IP address for the initiator instead of specifying a whole address pool.
address-pool (*none* | *string*; Default:)	Name of the address pool from which the responder will try to assign address if mode-config is enabled.
address-prefix-length (*integer [1..32]*; Default:)	Prefix length (netmask) of the assigned address from the pool.
comment (*string*; Default:)	

name (*string*; Default:)	
responder (*yes* | *no*; Default: **no**)	Specifies whether the configuration will work as an initiator (client) or responder (server). The initiator will request for mode-config parameters from the responder.
split-dns	List of DNS names that will be resolved using a system-dns=yes or static-dns= setting.
split-include (*list of IP prefix*; Default:)	List of subnets in CIDR format, which to tunnel. Subnets will be sent to the peer using the CISCO UNITY extension, a remote peer will create specific dynamic policies.
src-address-list (*address list*; Default:)	Specifying an address list will generate dynamic source NAT rules. This parameter is only available with responder=no. A roadWarrior client with NAT
static-dns (*list of IP*; Default:)	Manually specified DNS server's IP address to be sent to the client.
system-dns (*yes* | *no*; Default:)	When this option is enabled DNS addresses will be taken from `/ip dns`.

Read-only properties

Property	Description
default (*yes* | *no*)	Whether this is a default system entry.

⚠ Not all IKE implementations support multiple split networks provided by the split-include option.

⚠ If RouterOS client is initiator, it will always send CISCO UNITY extension, and RouterOS supports only split-include from this extension.

Both attributes Cisco Unity Split DNS (attribute type 28675) and RFC8598 (attribute type 25) are supported, ROS will answer to these attributes but only as responder.

⚠ It is not possible to use system-dns and static-dns at the same time, ROS can use only one DNS.

Installed SAs

This menu provides information about installed security associations including the keys.

Read-only properties

Property	Description
AH (*yes* | *no*)	Whether AH protocol is used by this SA.
ESP (*yes* | *no*)	Whether ESP protocol is used by this SA.
add-lifetime (*time/time*)	Added lifetime for the SA in format soft/hard: • soft - time period after which IKE will try to establish new SA; • hard - time period after which SA is deleted.
addtime (*time*)	Date and time when this SA was added.
auth-algorithm (*md5* | *null* | *sha1* | ...)	Currently used authentication algorithm.
auth-key (*string*)	Used authentication key.
current-bytes (*64-bit integer*)	A number of bytes seen by this SA.
dst-address (*IP*)	The destination address of this SA.

enc-algorithm (*des | 3des | aes-cbc | ...*)	Currently used encryption algorithm.
enc-key (*string*)	Used encryption key.
enc-key-size (*number*)	Used encryption key length.
expires-in (*yes | no*)	Time left until rekeying.
hw-aead (*yes | no*)	Whether this SA is hardware accelerated.
replay (*integer*)	Size of replay window in bytes.
spi (*string*)	Security Parameter Index identification tag
src-address (*IP*)	The source address of this SA.
state (*string*)	Shows the current state of the SA ("mature", "dying" etc)

Commands

Property	Description
flush ()	Manually removes all installed security associations.

Keys

This menu lists all imported public and private keys, that can be used for peer authentication. Menu has several commands to work with keys.

Properties

Property	Description
name (*string*; Default:)	

Read-only properties

Property	Description
key-size (*1024 | 2048 | 4096*)	Size of this key.
private-key (*yes | no*)	Whether this is a private key.
rsa (*yes | no*)	Whether this is an RSA key.

Commands

Property	Description
export-pub-key (*file-name; key*)	Export public key to file from one of existing private keys.
generate-key (*key-size; name*)	Generate a private key. Takes two parameters, name of the newly generated key and key size 1024,2048 and 4096.
import (*file-name; name*)	Import key from file.

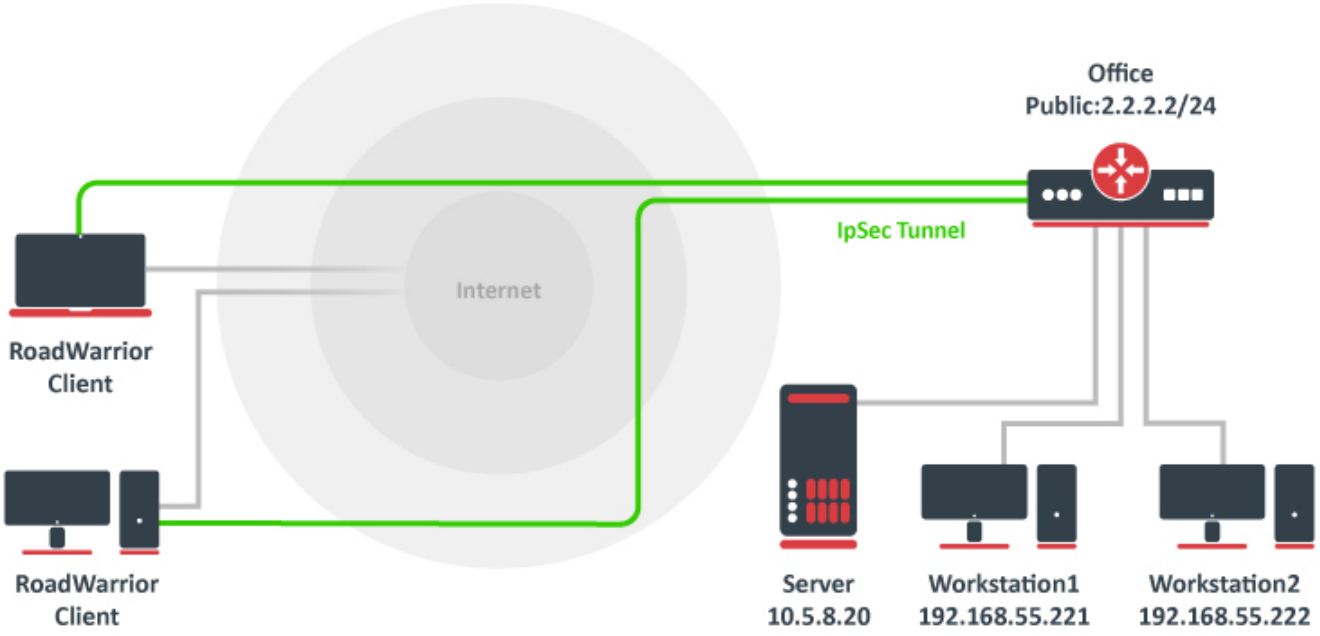
Settings

Property	Description
accounting (*yes* | *no*; Default:)	Whether to send RADIUS accounting requests to a RADIUS server. Applicable if EAP Radius (auth-method=eap-radius) or pre-shared key with XAuth authentication method (auth-method=pre-shared-key-xauth) is used.
interim-update (*time*; Default:)	The interval between each consecutive RADIUS accounting Interim update. Accounting must be enabled.
xauth-use-radius (*yes* | *no*; Default:)	Whether to use Radius client for XAuth users or not. Property is only applicable to peers using the **IKEv1** exchange mode.

Application Guides

RoadWarrior client with NAT

Consider setup as illustrated below. RouterOS acts as a RoadWarrior client connected to Office allowing access to its internal resources.



A tunnel is established, a local mode-config IP address is received and a set of dynamic policies are generated.

```
[admin@mikrotik] > ip ipsec policy print
Flags: T - template, X - disabled, D - dynamic, I - invalid, A - active, * - default
0 T * group=default src-address=::/0 dst-address=::/0 protocol=all proposal=default template=yes

1 DA src-address=192.168.77.254/32 src-port=any dst-address=10.5.8.0/24 dst-port=any protocol=all
action=encrypt level=unique ipsec-protocols=esp tunnel=yes sa-src-address=10.155.107.8
sa-dst-address=10.155.107.9 proposal=default ph2-count=1

2 DA src-address=192.168.77.254/32 src-port=any dst-address=192.168.55.0/24 dst-port=any protocol=all
action=encrypt level=unique ipsec-protocols=esp tunnel=yes sa-src-address=10.155.107.8
sa-dst-address=10.155.107.9 proposal=default ph2-count=1
```

Currently, only packets with a source address of 192.168.77.254/32 will match the IPsec policies. For a local network to be able to reach remote subnets, it is necessary to change the source address of local hosts to the dynamically assigned mode config IP address. It is possible to generate source NAT rules dynamically. This can be done by creating a new address list that contains all local networks that the NAT rule should be applied. In our case, it is 192.168.88.0/24.

```
/ip firewall address-list add address=192.168.88.0/24 list=local-RW
```

By specifying the address list under the mode-config initiator configuration, a set of source NAT rules will be dynamically generated.

```
/ip ipsec mode-config set [ find name="request-only" ] src-address-list=local-RW
```

When the IPsec tunnel is established, we can see the dynamically created source NAT rules for each network. Now every host in 192.168.88.0/24 is able to access Office's internal resources.

```
[admin@mikrotik] > ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic
0 D ;;; ipsec mode-config
chain=srcnat action=src-nat to-addresses=192.168.77.254 dst-address=192.168.55.0/24 src-address-list=local-RW

1 D ;;; ipsec mode-config
chain=srcnat action=src-nat to-addresses=192.168.77.254 dst-address=10.5.8.0/24 src-address-list=local-RW
```

Allow only IPsec encapsulated traffic

There are some scenarios where for security reasons you would like to drop access from/to specific networks if incoming/outgoing packets are not encrypted. For example, if we have L2TP/IPsec setup we would want to drop nonencrypted L2TP connection attempts.

There are several ways how to achieve this:

- Using IPsec policy matcher in firewall;
- Using generic IPsec policy with action set to **drop** and lower priority (can be used in Road Warrior setups where dynamic policies are generated);
- By setting DSCP or priority in mangle and matching the same values in firewall after decapsulation.

IPsec policy matcher

Let's set up an IPsec policy matcher to accept all packets that matched any of the IPsec policies and drop the rest:

```
add chain=input comment="ipsec policy matcher" in-interface=WAN ipsec-policy=in,ipsec
add action=drop chain=input comment="drop all" in-interface=WAN log=yes
```

IPsec policy matcher takes two parameters **direction, policy**. We used incoming direction and IPsec policy. IPsec policy option allows us to inspect packets after decapsulation, so for example, if we want to allow only GRE encapsulated packet from a specific source address and drop the rest we could set up the following rules:

```
add chain=input comment="ipsec policy matcher" in-interface=WAN ipsec-policy=in,ipsec protocol=gre
src=address=192.168.33.1
add action=drop chain=input comment="drop all" in-interface=WAN log=yes
```

For L2TP rule set would be:

```
add chain=input comment="ipsec policy matcher" in-interface=WAN ipsec-policy=in,ipsec protocol=udp dst-port=1701
add action=drop chain=input protocol=udp dst-port=1701 comment="drop l2tp" in-interface=WAN log=yes
```

Using generic IPsec policy

The trick of this method is to add a default policy with an action drop. Let's assume we are running an L2TP/IPsec server on a public 1.1.1.1 address and we want to drop all nonencrypted L2TP:

```
/ip ipsec policy
add src-address=1.1.1.1 dst-address=0.0.0.0/0 sa-src-address=1.1.1.1 protocol=udp src-port=1701 tunnel=yes
action=discard
```

Now router will drop any L2TP unencrypted incoming traffic, but after a successful L2TP/IPsec connection dynamic policy is created with higher priority than it is on default static rule, and packets matching that dynamic rule can be forwarded.

> ⚠ Policy order is important! For this to work, make sure the static drop policy is below the dynamic policies. Move it below the policy template if necessary.

```
[admin@rack2_10g1] /ip ipsec policy> print
Flags: T - template, X - disabled, D - dynamic, I - inactive, * - default
0 T * group=default src-address=::/0 dst-address=::/0 protocol=all
proposal=default template=yes

1 D src-address=1.1.1.1/32 src-port=1701 dst-address=10.5.130.71/32
dst-port=any protocol=udp action=encrypt level=require
ipsec-protocols=esp tunnel=no sa-src-address=1.1.1.1
sa-dst-address=10.5.130.71

2 src-address=1.1.1.1/32 src-port=1701 dst-address=0.0.0.0/0
dst-port=any protocol=udp action=discard level=unique
ipsec-protocols=esp tunnel=yes sa-src-address=1.1.1.1
sa-dst-address=0.0.0.0 proposal=default manual-sa=none
```

Manually specifying local-address parameter under Peer configuration

Using different routing table

IPsec, as any other service in RouterOS, uses the main routing table regardless of what local-address parameter is used for Peer configuration. It is necessary to apply routing marks to both IKE and IPSec traffic.

Consider the following example. There are two default routes - one in the main routing table and another in the routing table "backup". It is necessary to use the backup link for the IPsec site to site tunnel.

```
[admin@pair_r1] > /ip route print detail
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme, B -
blackhole, U - unreachable, P - prohibit
0 A S dst-address=0.0.0.0/0 gateway=10.155.107.1 gateway-status=10.155.107.1 reachable via ether1 distance=1
scope=30 target-scope=10 routing-mark=backup

1 A S dst-address=0.0.0.0/0 gateway=172.22.2.115 gateway-status=172.22.2.115 reachable via ether2 distance=1
scope=30 target-scope=10

2 ADC dst-address=10.155.107.0/25 pref-src=10.155.107.8 gateway=ether1 gateway-status=ether1 reachable
distance=0 scope=10

3 ADC dst-address=172.22.2.0/24 pref-src=172.22.2.114 gateway=ether2 gateway-status=ether2 reachable distance=0
scope=10

4 ADC dst-address=192.168.1.0/24 pref-src=192.168.1.1 gateway=bridge-local gateway-status=ether2 reachable
distance=0 scope=10

[admin@pair_r1] > /ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic
0 chain=srcnat action=masquerade out-interface=ether1 log=no log-prefix=""

1 chain=srcnat action=masquerade out-interface=ether2 log=no log-prefix=""
```

IPsec peer and policy configurations are created using the backup link's source address, as well as the NAT bypass rule for IPsec tunnel traffic.

```
/ip ipsec peer
add address=10.155.130.136/32 local-address=10.155.107.8 secret=test
/ip ipsec policy
add sa-src-address=10.155.107.8 src-address=192.168.1.0/24 dst-address=172.16.0.0/24 sa-dst-address=10.
155.130.136 tunnel=yes
/ip firewall nat
add action=accept chain=srcnat src-address=192.168.1.0/24 dst-address=172.16.0.0/24 place-before=0
```

Currently, we see "phase1 negotiation failed due to time up" errors in the log. It is because IPsec tries to reach the remote peer using the main routing table with an incorrect source address. It is necessary to mark UDP/500, UDP/4500, and ipsec-esp packets using Mangle:

```
/ip firewall mangle
add action=mark-connection chain=output connection-mark=no-mark dst-address=10.155.130.136 dst-port=500,4500
new-connection-mark=ipsec passthrough=yes protocol=udp
add action=mark-connection chain=output connection-mark=no-mark dst-address=10.155.130.136 new-connection-
mark=ipsec passthrough=yes protocol=ipsec-esp
add action=mark-routing chain=output connection-mark=ipsec new-routing-mark=backup passthrough=no
```

Using the same routing table with multiple IP addresses

Consider the following example. There are multiple IP addresses from the same subnet on the public interface. Masquerade rule is configured on out-interface. It is necessary to use one of the IP addresses explicitly.

```
[admin@pair_r1] > /ip address print
Flags: X - disabled, I - invalid, D - dynamic
# ADDRESS NETWORK INTERFACE
0 192.168.1.1/24 192.168.1.0 bridge-local
1 172.22.2.1/24 172.22.2.0 ether1
2 172.22.2.2/24 172.22.2.0 ether1
3 172.22.2.3/24 172.22.2.0 ether1

[admin@pair_r1] > /ip route print
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme, B -
blackhole, U - unreachable, P - prohibit
# DST-ADDRESS PREF-SRC GATEWAY DISTANCE
1 A S 0.0.0.0/0 172.22.2.115 1
3 ADC 172.22.2.0/24 172.22.2.1 ether1 0
4 ADC 192.168.1.0/24 192.168.1.1 bridge-local 0

[admin@pair_r1] /ip firewall nat> print
Flags: X - disabled, I - invalid, D - dynamic
0 chain=srcnat action=masquerade out-interface=ether1 log=no log-prefix=""
```

IPsec peer and policy configuration is created using one of the public IP addresses.

```
/ip ipsec peer
add address=10.155.130.136/32 local-address=172.22.2.3 secret=test
/ip ipsec policy
add sa-src-address=172.22.2.3 src-address=192.168.1.0/24 dst-address=172.16.0.0/24 sa-dst-address=10.
155.130.136 tunnel=yes
/ip firewall nat
add action=accept chain=srcnat src-address=192.168.1.0/24 dst-address=172.16.0.0/24 place-before=0
```

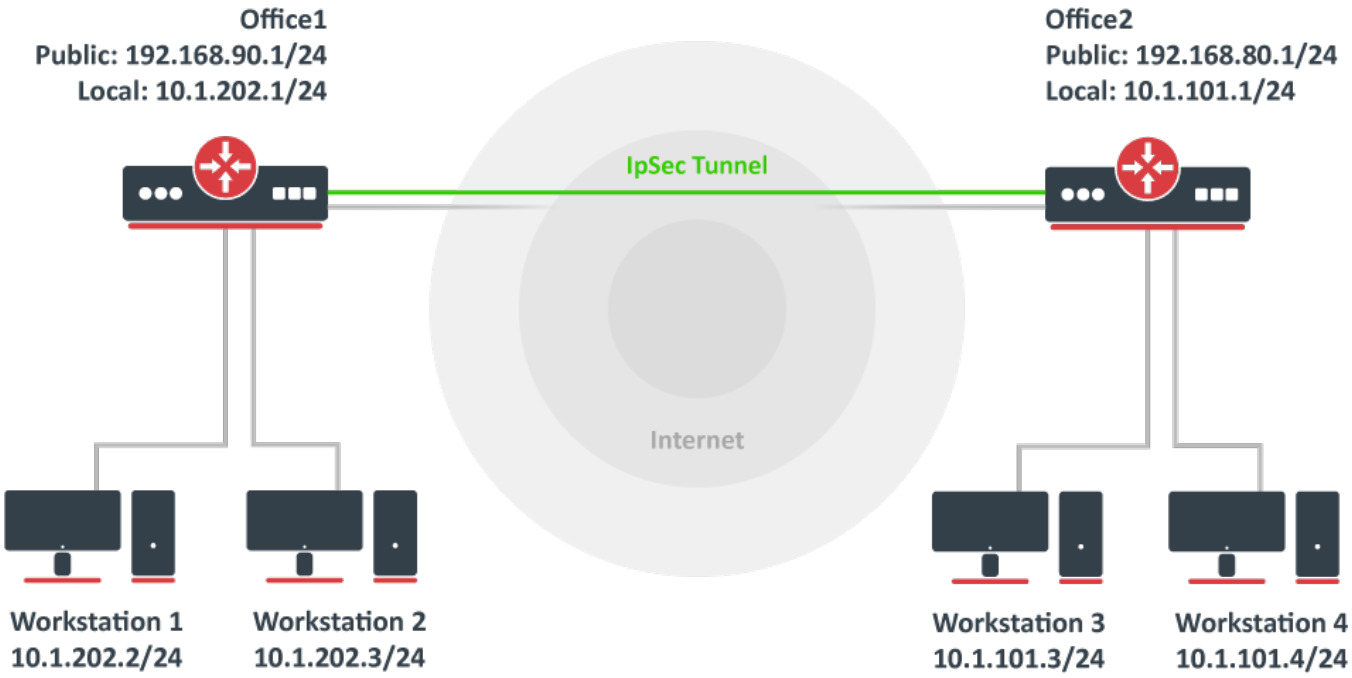
Currently, the phase 1 connection uses a different source address than we specified, and "phase1 negotiation failed due to time up" errors are shown in the logs. This is because masquerade is changing the source address of the connection to match the pref-src address of the connected route. The solution is to exclude connections from the public IP address from being masqueraded.

```
/ip firewall nat
add action=accept chain=srcnat protocol=udp src-port=500,4500 place-before=0
```

Application examples

Site to Site IPsec (IKEv1) tunnel

Consider setup as illustrated below. Two remote office routers are connected to the internet and office workstations are behind NAT. Each office has its own local subnet, 10.1.202.0/24 for Office1 and 10.1.101.0/24 for Office2. Both remote offices need secure tunnels to local networks behind routers.



Site 1 configuration

Start off by creating a new Phase 1 profile and Phase 2 proposal entries using stronger or weaker encryption parameters that suit your needs. It is advised to create separate entries for each menu so that they are unique for each peer in case it is necessary to adjust any of the settings in the future. These parameters must match between the sites or else the connection will not establish.

```
/ip ipsec profile
add dh-group=modp2048 enc-algorithm=aes-128 name=ike1-site2
/ip ipsec proposal
add enc-algorithms=aes-128-cbc name=ike1-site2 pfs-group=modp2048
```

Continue by configuring a peer. Specify the address of the remote router. This address should be reachable through UDP/500 and UDP/4500 ports, so make sure appropriate actions are taken regarding the router's firewall. Specify the name for this peer as well as the newly created profile.

```
/ip ipsec peer
add address=192.168.80.1/32 name=ike1-site2 profile=ike1-site2
```

The next step is to create an identity. For a basic pre-shared key secured tunnel, there is nothing much to set except for a **strong** secret and the peer to which this identity applies.

```
/ip ipsec identity
add peer=ike1-site2 secret=thisisnotasecurepsk
```

If security matters, consider using IKEv2 and a different auth-method.

Lastly, create a policy that controls the networks/hosts between whom traffic should be encrypted.

```
/ip ipsec policy
add src-address=10.1.202.0/24 src-port=any dst-address=10.1.101.0/24 dst-port=any tunnel=yes action=encrypt
proposal=ike1-site2 peer=ike1-site2
```

Site 2 configuration

Office 2 configuration is almost identical to Office 1 with proper IP address configuration. Start off by creating a new Phase 1 profile and Phase 2 proposal e ntries:

```
/ip ipsec profile
add dh-group=modp2048 enc-algorithm=aes-128 name=ike1-site1
/ip ipsec proposal
add enc-algorithms=aes-128-cbc name=ike1-site1 pfs-group=modp2048
```

Next is the peer and identity:

```
/ip ipsec peer
add address=192.168.90.1/32 name=ike1-site1 profile=ike1-site1
/ip ipsec identity
add peer=ike1-site1 secret=thisisnotasecurepsk
```

When it is done, create a policy:

```
/ip ipsec policy
add src-address=10.1.101.0/24 src-port=any dst-address=10.1.202.0/24 dst-port=any tunnel=yes action=encrypt
proposal=ike1-site1 peer=ike1-site1
```

At this point, the tunnel should be established and two IPsec Security Associations should be created on both routers:

```
/ip ipsec
active-peers print
installed-sa print
```

NAT and Fasttrack Bypass

At this point if you try to send traffic over the IPsec tunnel, it will not work, packets will be lost. This is because both routers have NAT rules (masquerade) that are changing source addresses before a packet is encrypted. A router is unable to encrypt the packet because the source address does not match the address specified in the policy configuration. For more information see the IPsec packet flow example.

To fix this we need to set up IP/Firewall/NAT bypass rule.

Office 1 router:

```
/ip firewall nat
add chain=srcnat action=accept place-before=0 src-address=10.1.202.0/24 dst-address=10.1.101.0/24
```

Office 2 router:

```
/ip firewall nat
add chain=srcnat action=accept place-before=0 src-address=10.1.101.0/24 dst-address=10.1.202.0/24
```

> ⚠ If you previously tried to establish an IP connection before the NAT bypass rule was added, you have to clear the connection table from the existing connection or restart both routers.

It is very important that the bypass rule is placed at the top of all other NAT rules.

Another issue is if you have IP/Fasttrack enabled, the packet bypasses IPsec policies. So we need to add accept rule before FastTrack.

```
/ip firewall filter
add chain=forward action=accept place-before=1
src-address=10.1.101.0/24 dst-address=10.1.202.0/24 connection-state=established,related
add chain=forward action=accept place-before=1
src-address=10.1.202.0/24 dst-address=10.1.101.0/24 connection-state=established,related
```

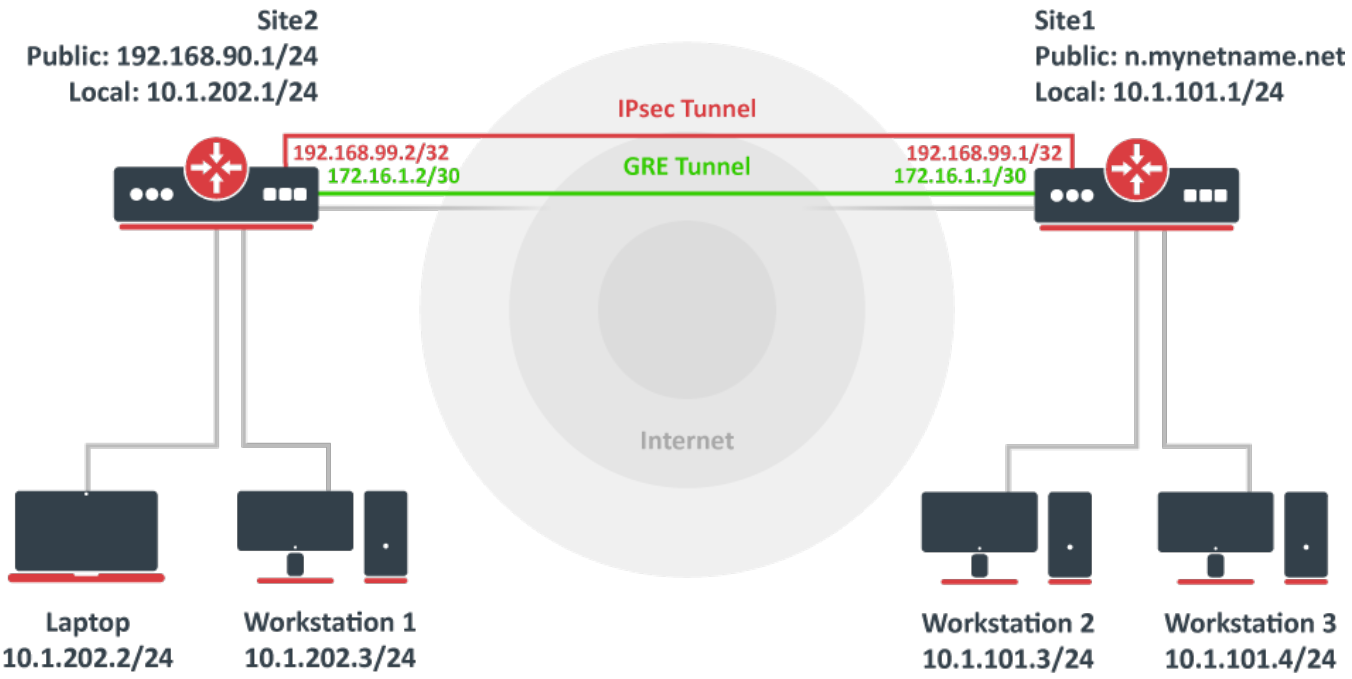
However, this can add a significant load to the router's CPU if there is a fair amount of tunnels and significant traffic on each tunnel.

The solution is to use IP/Firewall/Raw to bypass connection tracking, that way eliminating the need for filter rules listed above and reducing the load on CPU by approximately 30%.

```
/ip firewall raw
add action=notrack chain=prerouting src-address=10.1.101.0/24 dst-address=10.1.202.0/24
add action=notrack chain=prerouting src-address=10.1.202.0/24 dst-address=10.1.101.0/24
```

Site to Site GRE tunnel over IPsec (IKEv2) using DNS

This example explains how it is possible to establish a secure and encrypted GRE tunnel between two RouterOS devices when one or both sites do not have a static IP address. Before making this configuration possible, it is necessary to have a DNS name assigned to one of the devices which will act as a responder (server). For simplicity, we will use RouterOS built-in DDNS service IP/Cloud.



Site 1 (server) configuration

This is the side that will listen to incoming connections and act as a responder. We will use mode config to provide an IP address for the second site, but first, create a loopback (blank) bridge and assign an IP address to it that will be used later for GRE tunnel establishment.

```
/interface bridge
add name=loopback
/ip address
add address=192.168.99.1 interface=loopback
```

Continuing with the IPsec configuration, start off by creating a new Phase 1 profile and Phase 2 proposal entries using stronger or weaker encryption parameters that suit your needs. Note that this configuration example will listen to all incoming IKEv2 requests, meaning the profile configuration will be shared between all other configurations (e.g. RoadWarrior).

```
/ip ipsec profile
add dh-group=ecp256,modp2048,modp1024 enc-algorithm=aes-256,aes-192,aes-128 name=ike2
/ip ipsec proposal
add auth-algorithms=null enc-algorithms=aes-128-gcm name=ike2-gre pfs-group=none
```

Next, create a new mode config entry with responder=yes. This will provide an IP configuration for the other site as well as the host (loopback address) for policy generation.

```
/ip ipsec mode-config
add address=192.168.99.2 address-prefix-length=32 name=ike2-gre split-include=192.168.99.1/32 system-dns=no
```

It is advised to create a new policy group to separate this configuration from any existing or future IPsec configuration.

```
/ip ipsec policy group
add name=ike2-gre
```

Now it is time to set up a new policy template that will match the remote peers new dynamic address and the loopback address.

```
/ip ipsec policy
add dst-address=192.168.99.2/32 group=ike2-gre proposal=ike2-gre src-address=192.168.99.1/32 template=yes
```

The next step is to create a peer configuration that will listen to all IKEv2 requests. If you already have such an entry, you can skip this step.

```
/ip ipsec peer
add exchange-mode=ike2 name=ike2 passive=yes profile=ike2
```

Lastly, set up an identity that will match our remote peer by pre-shared-key authentication with a specific secret.

```
/ip ipsec identity
add generate-policy=port-strict mode-config=ike2-gre peer=ike2 policy-template-group=ike2-gre secret=test
```

The server side is now configured and listening to all IKEv2 requests. Please make sure the firewall is not blocking UDP/4500 port.

The last step is to create the GRE interface itself. This can also be done later when an IPsec connection is established from the client-side.

```
/interface gre
add local-address=192.168.99.1 name=gre-tunnel1 remote-address=192.168.99.2
```

Configure IP address and route to remote network through GRE interface.

```
/ip address
add address=172.16.1.1/30 interface=gre-tunnel1
/ip route
add dst-network=10.1.202.0/24 gateway=172.16.1.2
```

Site 2 (client) configuration

Similarly to server configuration, start off by creating a new Phase 1 profile and Phase 2 proposal configurations. Since this site will be the initiator, we can use a more specific profile configuration to control which exact encryption parameters are used, just make sure they overlap with what is configured on the server-side.

```
/ip ipsec profile
add dh-group=ecp256 enc-algorithm=aes-256 name=ike2-gre
/ip ipsec proposal
add auth-algorithms=null enc-algorithms=aes-128-gcm name=ike2-gre pfs-group=none
```

Next, create a new mode config entry with responder=no. This will make sure the peer requests IP and split-network configuration from the server.

```
/ip ipsec mode-config
add name=ike2-gre responder=no
```

It is also advised to create a new policy group to separate this configuration from any existing or future IPsec configuration.

```
/ip ipsec policy group
add name=ike2-gre
```

Create a new policy template on the client-side as well.

```
/ip ipsec policy
add dst-address=192.168.99.1/32 group=ike2-gre proposal=ike2-gre src-address=192.168.99.2/32 template=yes
```

Move on to peer configuration. Now we can specify the DNS name for the server under the address parameter. Obviously, you can use an IP address as well.

```
/ip ipsec peer
add address=n.mynetname.net exchange-mode=ike2 name=p1.ez profile=ike2-gre
```

Lastly, create an identity for our newly created peers.

```
/ip ipsec identity
add generate-policy=port-strict mode-config=ike2-gre peer=p1.ez policy-template-group=ike2-gre secret=test
```

If everything was done properly, there should be a new dynamic policy present.

```
/ip ipsec policy print
Flags: T - template, X - disabled, D - dynamic, I - invalid, A - active, * - default
0 T * group=default src-address=::/0 dst-address=::/0 protocol=all proposal=default template=yes

1 T group=ike2-gre src-address=192.168.99.2/32 dst-address=192.168.99.1/32 protocol=all proposal=ike2-gre
template=yes

2 DA src-address=192.168.99.2/32 src-port=any dst-address=192.168.99.1/32 dst-port=any protocol=all
action=encrypt level=unique ipsec-protocols=esp
tunnel=yes sa-src-address=192.168.90.1 sa-dst-address=(current IP of n.mynetname.net) proposal=ike2-gre ph2-
count=1
```

A secure tunnel is now established between both sites which will encrypt all traffic between 192.168.99.2 <=> 192.168.99.1 addresses. We can use these addresses to create a GRE tunnel.

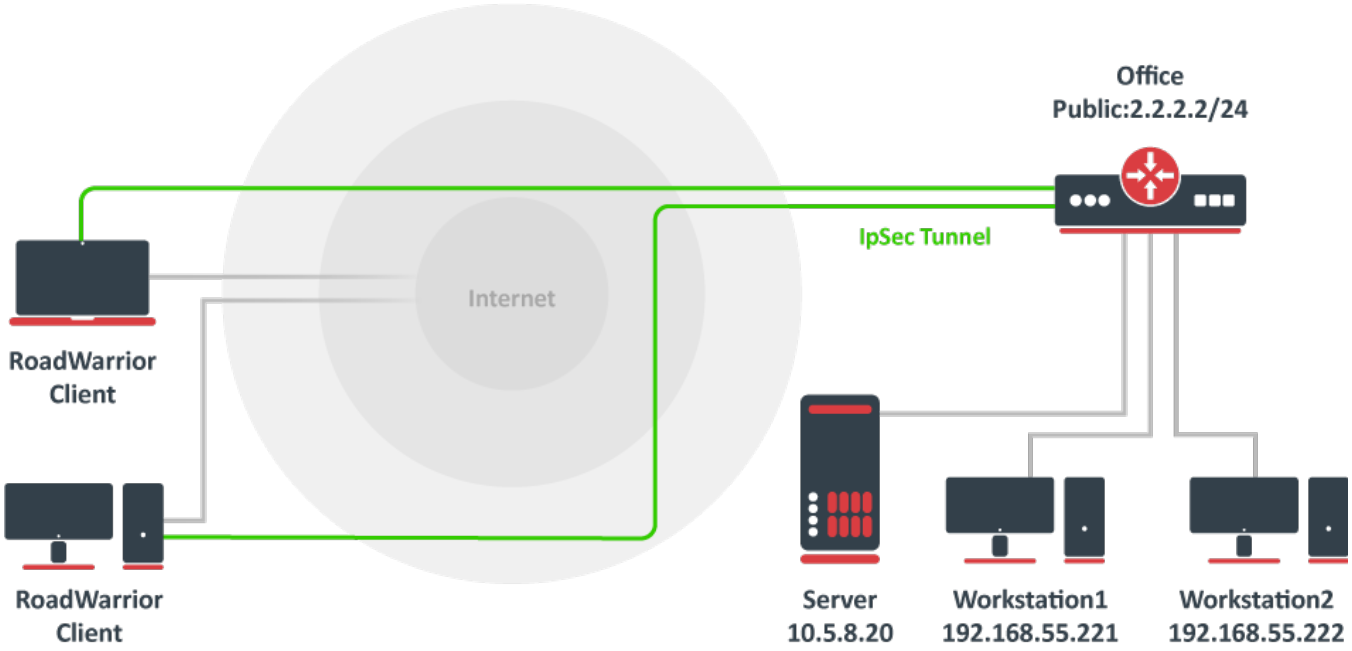
```
/interface gre
add local-address=192.168.99.2 name=gre-tunnel1 remote-address=192.168.99.1
```

Configure IP address and route to remote network through GRE interface.

```
/ip address
add address=172.16.1.2/30 interface=gre-tunnel1
/ip route
add dst-network=10.1.101.0/24 gateway=172.16.1.1
```

Road Warrior setup using IKEv2 with RSA authentication

This example explains how to establish a secure IPsec connection between a device connected to the Internet (road warrior client) and a device running RouterOS acting as a server.



RouterOS server configuration

Before configuring IPsec, it is required to set up certificates. It is possible to use a separate Certificate Authority for certificate management, however in this example, self-signed certificates are generated in RouterOS System/Certificates menu. Some certificate requirements should be met to connect various devices to the server:

- Common name should contain IP or DNS name of the server;
- SAN (subject alternative name) should have IP or DNS of the server;
- EKU (extended key usage) tls-server and tls-client are required.

Considering all requirements above, generate CA and server certificates:

```
/certificate
add common-name=ca name=ca
sign ca ca-crl-host=2.2.2.2
add common-name=2.2.2.2 subject-alt-name=IP:2.2.2.2 key-usage=tls-server name=server1
sign server1 ca=ca
```

Now that valid certificates are created on the router, add a new Phase 1 profile and Phase 2 proposal entries with pfs-group=none:

```
/ip ipsec profile
add name=ike2
/ip ipsec proposal
add name=ike2 pfs-group=none
```

Mode config is used for address distribution from IP/Pools:

```
/ip pool
add name=ike2-pool ranges=192.168.77.2-192.168.77.254
/ip ipsec mode-config
add address-pool=ike2-pool address-prefix-length=32 name=ike2-conf
```

Since that the policy template must be adjusted to allow only specific network policies, it is advised to create a separate policy group and template.

```
/ip ipsec policy group
add name=ike2-policies
/ip ipsec policy
add dst-address=192.168.77.0/24 group=ike2-policies proposal=ike2 src-address=0.0.0.0/0 template=yes
```

Create a new IPsec peer entry that will listen to all incoming IKEv2 requests.

```
/ip ipsec peer
add exchange-mode=ike2 name=ike2 passive=yes profile=ike2
```

Identity configuration

The identity menu allows to match specific remote peers and assign different configurations for each one of them. First, create a default identity, that will accept all peers, but will verify the peer's identity with its certificate.

```
/ip ipsec identity
add auth-method=digital-signature certificate=server1 generate-policy=port-strict mode-config=ike2-conf
peer=ike2 policy-template-group=ike2-policies
```

⚠ If the peer's ID (ID_i) is not matching with the certificate it sends, the identity lookup will fail. See remote-id in the identities section.

For example, we want to assign a different mode config for user "A", who uses certificate "rw-client1" to authenticate itself to the server. First of all, make sure a new mode config is created and ready to be applied for the specific user.

```
/ip ipsec mode-config
add address=192.168.66.2 address-prefix-length=32 name=usr_A split-include=192.168.55.0/24 system-dns=no
```

It is possible to apply this configuration for user "A" by using the match-by=certificate parameter and specifying his certificate with remote-certificate.

```
/ip ipsec identity
add auth-method=digital-signature certificate=server1 generate-policy=port-strict match-by=certificate mode-
config=usr_A peer=ike2 policy-template-group=ike2-policies remote-certificate=rw-client1
```

(Optional) Split tunnel configuration

Split tunneling is a method that allows road warrior clients to only access a specific secured network and at the same time send the rest of the traffic based on their internal routing table (as opposed to sending all traffic over the tunnel). To configure split tunneling, changes to mode config parameters are needed.

For example, we will allow our road warrior clients to only access the 10.5.8.0/24 network.

```
/ip ipsec mode-conf
set [find name="rw-conf"] split-include=10.5.8.0/24
```

It is also possible to send a specific DNS server for the client to use. By default, system-dns=yes is used, which sends DNS servers that are configured on the router itself in IP/DNS. We can force the client to use a different DNS server by using the static-dns parameter.

```
/ip ipsec mode-conf
set [find name="rw-conf"] system-dns=no static-dns=10.5.8.1
```

While it is possible to adjust the IPsec policy template to only allow road warrior clients to generate policies to network configured by split-include parameter , this can cause compatibility issues with different vendor implementations (see known limitations). Instead of adjusting the policy template, allow access to a secured network in IP/Firewall/Filter and drop everything else.

```
/ip firewall filter
add action=drop chain=forward src-address=192.168.77.0/24 dst-address=!10.5.8.0/24
```

⚠ Split networking is not a security measure. The client (initiator) can still request a different Phase 2 traffic selector.

Generating client certificates

To generate a new certificate for the client and sign it with a previously created CA.

```
/certificate
add common-name=rw-client1 name=rw-client1 key-usage=tls-client
sign rw-client1 ca=ca
```

PKCS12 format is accepted by most client implementations, so when exporting the certificate, make sure PKCS12 is specified.

```
/certificate
export-certificate rw-client1 export-passphrase=1234567890 type=pkcs12
```

A file named *cert_export_rw-client1.p12* is now located in the routers System/File section. This file should be securely transported to the client's device.

Typically PKCS12 bundle contains also a CA certificate, but some vendors may not install this CA, so a self-signed CA certificate must be exported separately using PEM format.

```
/certificate
export-certificate ca type=pem
```

A file named *cert_export_ca.crt* is now located in the routers System/File section. This file should also be securely transported to the client's device.

PEM is another certificate format for use in client software that does not support PKCS12. The principle is pretty much the same.

```
/certificate
export-certificate ca
export-certificate rw-client1 export-passphrase=1234567890
```

Three files are now located in the routers Files section: *cert_export_ca.crt*, *cert_export_rw-client1.crt* and *cert_export_rw-client1.key* which should be securely transported to the client device.

Known limitations

Here is a list of known limitations by popular client software IKEv2 implementations.

- Windows will always ignore networks received by split-include and request policy with destination 0.0.0.0/0 (TSr). When IPsec-SA is generated, Windows requests DHCP option 249 to which RouterOS will respond with configured split-include networks automatically.
- Both Apple macOS and iOS will only accept the first split-include network.
- Both Apple macOS and iOS will use the DNS servers from system-dns and static-dns parameters only when 0.0.0.0/0 split-include is used.
- While some implementations can make use of different PFS group for phase 2, it is advised to use pfs-group=none under proposals to avoid any compatibility issues.

RouterOS client configuration

Import a PKCS12 format certificate in RouterOS.

```
/certificate import file-name=cert_export_RouterOS_client.p12 passphrase=1234567890
```

There should now be the self-signed CA certificate and the client certificate in the Certificate menu. Find out the name of the client certificate.

```
/certificate print
```

cert_export_RouterOS_client.p12_0 is the client certificate.

It is advised to create a separate Phase 1 profile and Phase 2 proposal configurations to not interfere with any existing IPsec configuration.

```
/ip ipsec profile
add name=ike2-rw
/ip ipsec proposal
add name=ike2-rw pfs-group=none
```

While it is possible to use the default policy template for policy generation, it is better to create a new policy group and template to separate this configuration from any other IPsec configuration.

```
/ip ipsec policy group
add name=ike2-rw
/ip ipsec policy
add group=ike2-rw proposal=ike2-rw template=yes
```

Create a new mode config entry with responder=no that will request configuration parameters from the server.

```
/ip ipsec mode-config
add name=ike2-rw responder=no
```

Lastly, create peer and identity configurations.

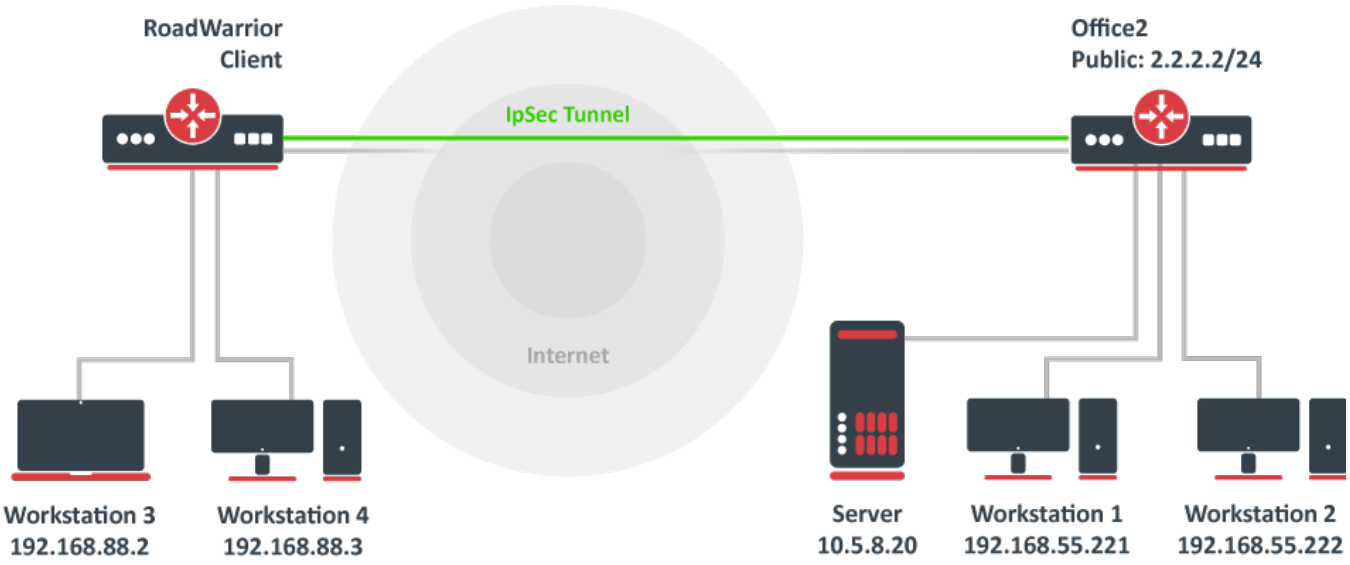
```
/ip ipsec peer
add address=2.2.2.2/32 exchange-mode=ike2 name=ike2-rw-client
/ip ipsec identity
add auth-method=digital-signature certificate=cert_export_RouterOS_client.p12_0 generate-policy=port-strict
mode-config=ike2-rw peer=ike2-rw-client policy-template-group=ike2-rw
```

Verify that the connection is successfully established.

```
/ip ipsec
active-peers print
installed-sa print
```

Enabling dynamic source NAT rule generation

If we look at the generated dynamic policies, we see that only traffic with a specific (received by mode config) source address will be sent through the tunnel. But a router in most cases will need to route a specific device or network through the tunnel. In such case, we can use source NAT to change the source address of packets to match the mode config address. Since the mode config address is dynamic, it is impossible to create a static source NAT rule. In RouterOS, it is possible to generate dynamic source NAT rules for mode config clients.



For example, we have a local network 192.168.88.0/24 behind the router and we want all traffic from this network to be sent over the tunnel. First of all, we have to make a new IP/Firewall/Address list which consists of our local network

```
/ip firewall address-list
add address=192.168.88.0/24 list=local
```

When it is done, we can assign the newly created IP/Firewall/Address list to the mode config configuration.

```
/ip ipsec mode-config
set [ find name=ike2-rw ] src-address-list=local
```

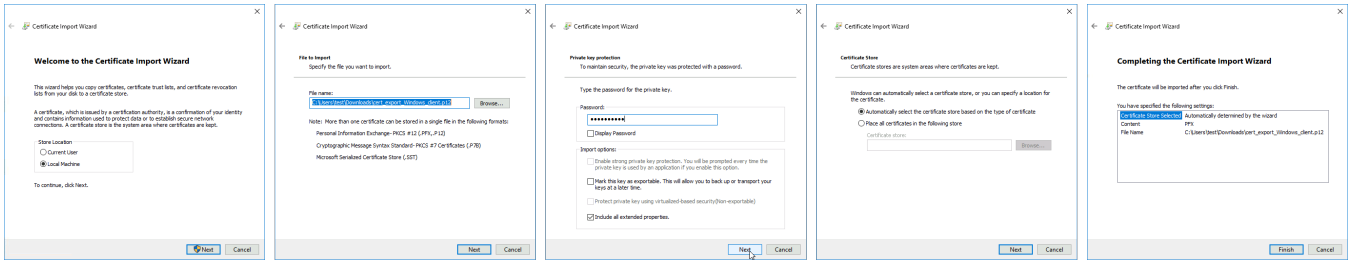
Verify correct source NAT rule is dynamically generated when the tunnel is established.

```
[admin@MikroTik] > /ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic
0 D ;;; ipsec mode-config
chain=srcnat action=src-nat to-addresses=192.168.77.254 src-address-list=local dst-address-list=!local
```

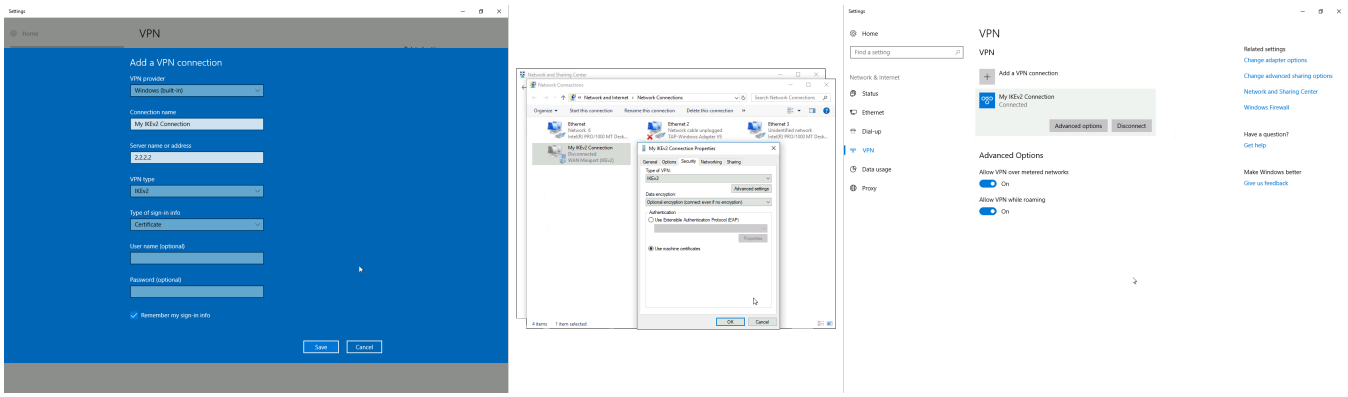
Make sure the dynamic mode config address is not a part of a local network.

Windows client configuration

Open PKCS12 format certificate file on the Windows computer. Install the certificate by following the instructions. Make sure you select the Local Machine store location.



You can now proceed to Network and Internet settings -> VPN and add a new configuration. Fill in the Connection name, Server name, or address parameters. Select IKEv2 under VPN type. When it is done, it is necessary to select "Use machine certificates". This can be done in Network and Sharing Center by clicking the Properties menu for the VPN connection. The setting is located under the Security tab.



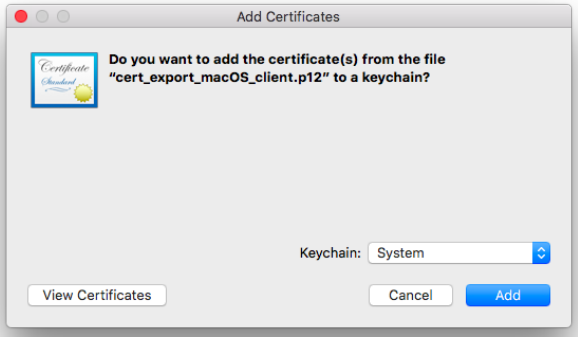
Currently, Windows 10 is compatible with the following Phase 1 (profiles) and Phase 2 (proposals) proposal sets:

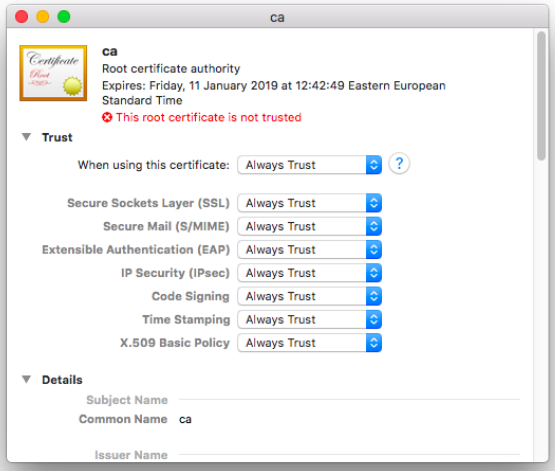
Phase 1		
Hash Algorithm	**Encryption Algorithm**	**DH Group**
SHA1	3DES	modp1024
SHA256	3DES	modp1024
SHA1	AES-128-CBC	modp1024
SHA256	AES-128-CBC	modp1024
SHA1	AES-192-CBC	modp1024
SHA256	AES-192-CBC	modp1024
SHA1	AES-256-CBC	modp1024
SHA256	AES-256-CBC	modp1024
SHA1	AES-128-GCM	modp1024
SHA256	AES-128-GCM	modp1024
SHA1	AES-256-GCM	modp1024
SHA256	AES-256-GCM	modp1024

Phase 2		
Hash Algorithm	**Encryption Algorithm**	**PFS Group**
SHA1	AES-256-CBC	none
SHA1	AES-128-CBC	none
SHA1	3DES	none
SHA1	DES	none
SHA1	none	none

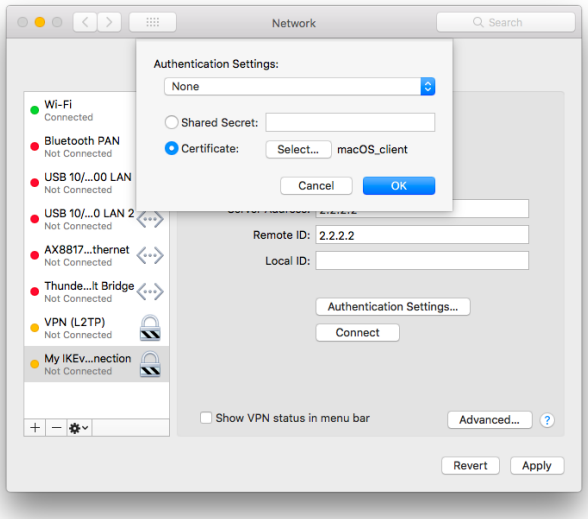
macOS client configuration

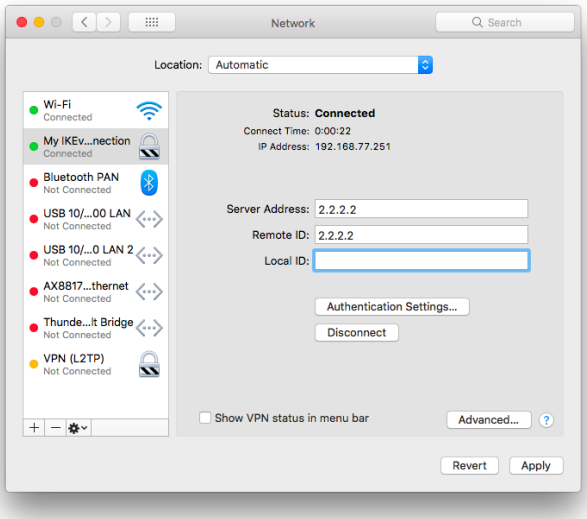
Open the PKCS12 format certificate file on the macOS computer and install the certificate in the "System" keychain. It is necessary to mark the CA certificate as trusted manually since it is self-signed. Locate the certificate macOS Keychain Access app under the System tab and mark it as Always Trust.





You can now proceed to System Preferences -> Network and add a new configuration by clicking the + button. Select Interface: VPN, VPN Type: IKEv2 and name your connection. Remote ID must be set equal to common-name or subjAltName of server's certificate. Local ID can be left blank. Under Authentication Settings select None and choose the client certificate. You can now test the connectivity.





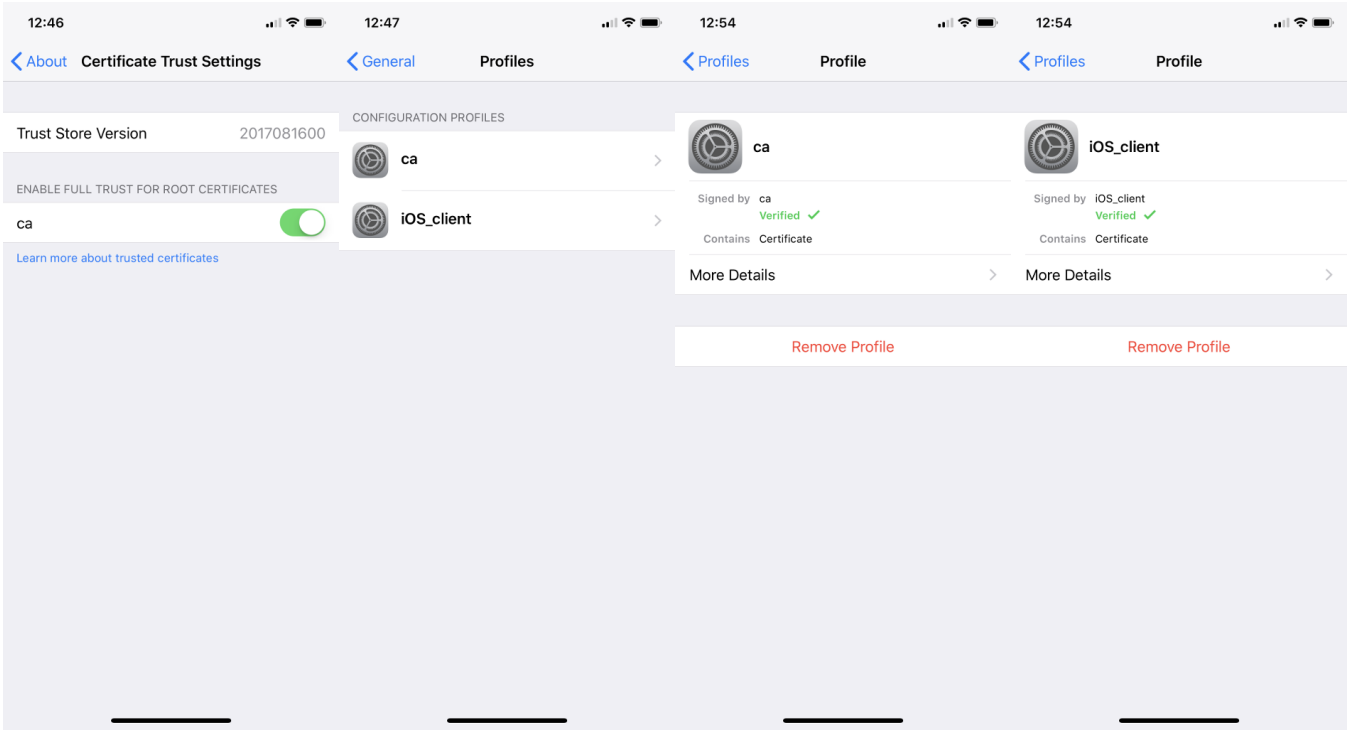
Currently, macOS is compatible with the following Phase 1 (profiles) and Phase 2 (proposals) proposal sets:

Phase 1		
Hash Algorithm	**Encryption Algorithm**	**DH Group**
SHA256	AES-256-CBC	modp2048
SHA256	AES-256-CBC	ecp256
SHA256	AES-256-CBC	modp1536
SHA1	AES-128-CBC	modp1024
SHA1	3DES	modp1024

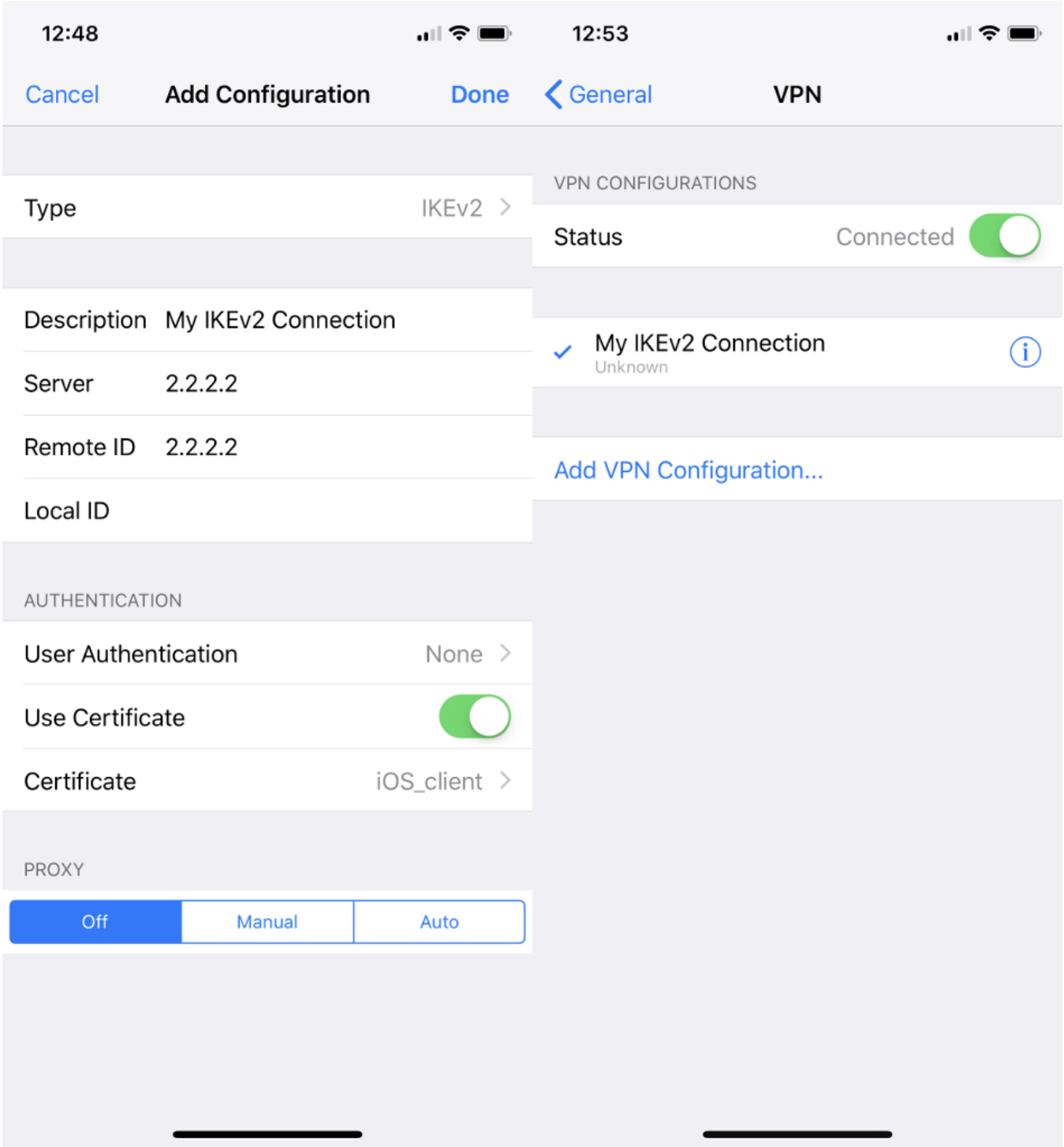
Phase 2		
Hash Algorithm	**Encryption Algorithm**	**PFS Group**
SHA256	AES-256-CBC	none
SHA1	AES-128-CBC	none
SHA1	3DES	none

iOS client configuration

Typically PKCS12 bundle contains also a CA certificate, but iOS does not install this CA, so a self-signed CA certificate must be installed separately using PEM format. Open these files on the iOS device and install both certificates by following the instructions. It is necessary to mark the self-signed CA certificate as trusted on the iOS device. This can be done in Settings -> General -> About -> Certificate Trust Settings menu. When it is done, check whether both certificates are marked as "verified" under the Settings -> General -> Profiles menu.



You can now proceed to Settings -> General -> VPN menu and add a new configuration. Remote ID must be set equal to common-name or subjAltName of server's certificate. Local ID can be left blank.



Currently, iOS is compatible with the following Phase 1 (profiles) and Phase 2 (proposals) proposal sets:

Phase 1		
Hash Algorithm	**Encryption Algorithm**	**DH Group**
SHA256	AES-256-CBC	modp2048
SHA256	AES-256-CBC	ecp256
SHA256	AES-256-CBC	modp1536

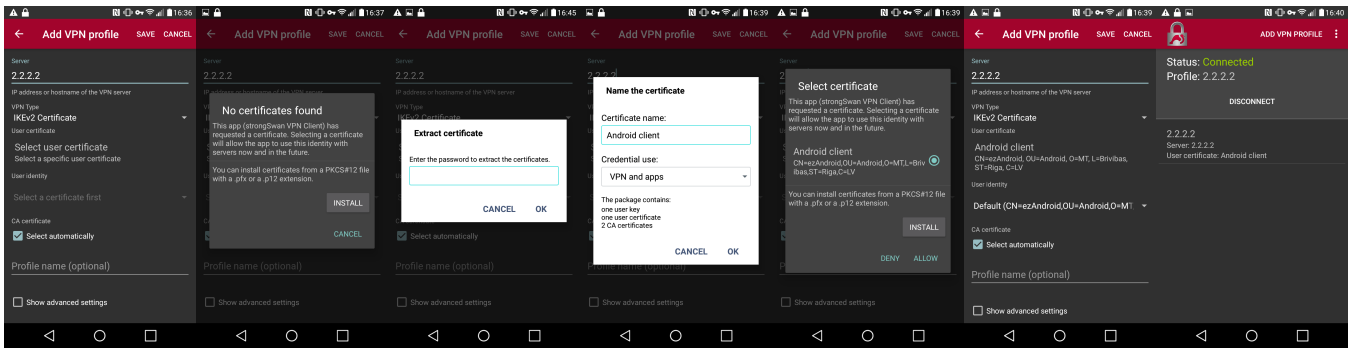
SHA1	AES-128-CBC	modp1024
SHA1	3DES	modp1024

Phase 2		
Hash Algorithm	**Encryption Algorithm**	**PFS Group**
SHA256	AES-256-CBC	none
SHA1	AES-128-CBC	none
SHA1	3DES	none

⚠ If you are connected to the VPN over WiFi, the iOS device can go into sleep mode and disconnect from the network.

Android (strongSwan) client configuration

Currently, there is no IKEv2 native support in Android, however, it is possible to use strongSwan from Google Play Store which brings IKEv2 to Android. StrongSwan accepts PKCS12 format certificates, so before setting up the VPN connection in strongSwan, make sure you download the PKCS12 bundle to your Android device. When it is done, create a new VPN profile in strongSwan, type in the server IP, and choose "IKEv2 Certificate" as VPN Type. When selecting a User certificate, press Install and follow the certificate extract procedure by specifying the PKCS12 bundle. Save the profile and test the connection by pressing on the VPN profile.



It is possible to specify custom encryption settings in strongSwan by ticking the "Show advanced settings" checkbox. Currently, strongSwan by default is compatible with the following Phase 1 (profiles) and Phase 2 (proposals) proposal sets:

Phase 1		
Hash Algorithm	**Encryption Algorithm**	**DH Group**
SHA*	AES-*-CBC	modp2048
SHA*	AES-*-CBC	ecp256
SHA*	AES-*-CBC	ecp384
SHA*	AES-*-CBC	ecp521
SHA*	AES-*-CBC	modp3072
SHA*	AES-*-CBC	modp4096
SHA*	AES-*-CBC	modp6144
SHA*	AES-*-CBC	modp8192
SHA*	AES-*-GCM	modp2048

SHA*	AES-*-GCM	ecp256
SHA*	AES-*-GCM	ecp384
SHA*	AES-*-GCM	ecp521
SHA*	AES-*-GCM	modp3072
SHA*	AES-*-GCM	modp4096
SHA*	AES-*-GCM	modp6144
SHA*	AES-*-GCM	modp8192

Phase 2		
Hash Algorithm	**Encryption Algorithm**	**PFS Group**
none	AES-256-GCM	none
none	AES-128-GCM	none
SHA256	AES-256-CBC	none
SHA512	AES-256-CBC	none
SHA1	AES-256-CBC	none
SHA256	AES-192-CBC	none
SHA512	AES-192-CBC	none
SHA1	AES-192-CBC	none
SHA256	AES-128-CBC	none
SHA512	AES-128-CBC	none
SHA1	AES-128-CBC	none

Linux (strongSwan) client configuration

Download the PKCS12 certificate bundle and move it to /etc/ipsec.d/private directory.

Add exported passphrase for the private key to /etc/ipsec.secrets file where "strongSwan_client.p12" is the file name and "1234567890" is the passphrase.

```
: P12 strongSwan_client.p12 "1234567890"
```

Add a new connection to /etc/ipsec.conf file

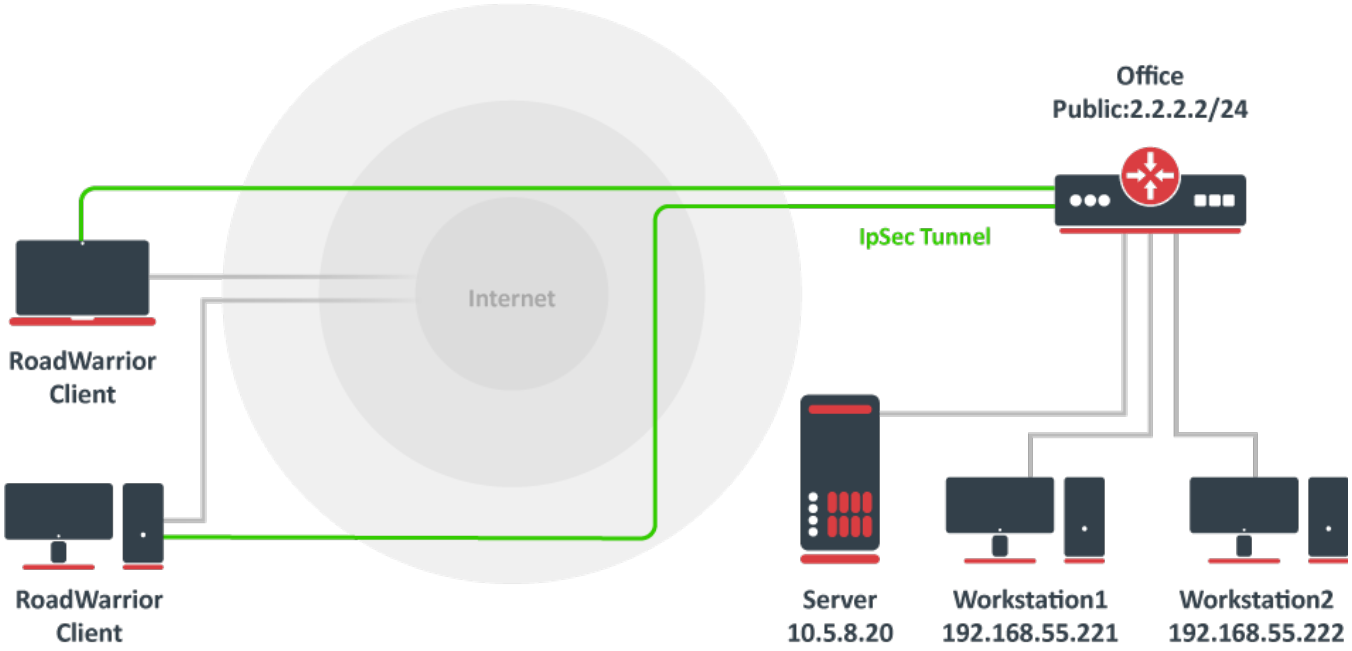
```
conn "ikev2"
keyexchange=ikev2
ike=aes128-sha1-modp2048
esp=aes128-sha1
leftsourceip=%modeconfig
leftcert=strongSwan_client.p12
leftfirewall=yes
right=2.2.2.2
rightid="CN=2.2.2.2"
rightsubnet=0.0.0.0/0
auto=add
```

You can now restart (or start) the ipsec daemon and initialize the connection

```
$ ipsec restart
$ ipsec up ikev2
```

Road Warrior setup using IKEv2 with EAP-MSCHAPv2 authentication handled by User Manager (RouterOS v7)

This example explains how to establish a secure IPsec connection between a device connected to the Internet (road warrior client) and a device running RouterOS acting as an IKEv2 server and User Manager. It is possible to run User Manager on a separate device in network, however in this example both User Manager and IKEv2 server will be configured on the same device (Office).



RouterOS server configuration

Requirements

For this setup to work there are several prerequisites for the router:

1. Router's IP address should have a valid public DNS record - IP Cloud could be used to achieve this.
2. Router should be reachable through port TCP/80 over the Internet - if the server is behind NAT, port forwarding should be configured.
3. User Manager package should be installed on the router.

Generating Let's Encrypt certificate

During the EAP-MSCHAPv2 authentication, TLS handshake has to take place, which means the server has to have a certificate that can be validated by the client. To simplify this step, we will use Let's Encrypt certificate which can be validated by most operating systems without any intervention by the user. To generate the certificate, simply enable SSL certificate under the Certificates menu. By default the command uses the dynamic DNS record provided by IP Cloud, however a custom DNS name can also be specified. Note that, the DNS record should point to the router.

```
/certificate enable-ssl-certificate
```

If the certificate generation succeeded, you should see the Let's Encrypt certificate installed under the Certificates menu.

```
/certificate print detail where name~"letsencrypt"
```

Configuring User Manager

First of all, allow receiving RADIUS requests from the localhost (the router itself):

```
/user-manager router
add address=127.0.0.1 comment=localhost name=local shared-secret=test
```

Enable the User Manager and specify the Let's Encrypt certificate (replace the name of the certificate to the one installed on your device) that will be used to authenticate the users.

```
/user-manager
set certificate="letsencrypt_2021-04-09T07:10:55Z" enabled=yes
```

Lastly add users and their credentials that clients will use to authenticate to the server.

```
/user-manager user
add name=user1 password=password
```

Configuring RADIUS client

For the router to use RADIUS server for user authentication, it is required to add a new RADIUS client that has the same shared secret that we already configured on User Manager.

```
/radius
add address=127.0.0.1 secret=test service=ipsec
```

IPsec (IKEv2) server configuration

Add a new Phase 1 profile and Phase 2 proposal entries with pfs-group=none:

```
/ip ipsec profile
add name=ike2
/ip ipsec proposal
add name=ike2 pfs-group=none
```

Mode config is used for address distribution from IP/Pools.

```
/ip pool
add name=ike2-pool ranges=192.168.77.2-192.168.77.254
/ip ipsec mode-config
add address-pool=ike2-pool address-prefix-length=32 name=ike2-conf
```

Since that the policy template must be adjusted to allow only specific network policies, it is advised to create a separate policy group and template.

```
/ip ipsec policy group
add name=ike2-policies
/ip ipsec policy
add dst-address=192.168.77.0/24 group=ike2-policies proposal=ike2 src-address=0.0.0.0/0 template=yes
```

Create a new IPsec peer entry which will listen to all incoming IKEv2 requests.

```
/ip ipsec peer
add exchange-mode=ike2 name=ike2 passive=yes profile=ike2
```

Lastly create a new IPsec identity entry that will match all clients trying to authenticate with EAP. Note that generated Let's Encrypt certificate must be specified.

```
/ip ipsec identity
add auth-method=eap-radius certificate="letsencrypt_2021-04-09T07:10:55Z" generate-policy=port-strict mode-
config=ike2-conf peer=ike2 \
policy-template-group=ike2-policies
```

(Optional) Split tunnel configuration

Split tunneling is a method that allows road warrior clients to only access a specific secured network and at the same time send the rest of the traffic based on their internal routing table (as opposed to sending all traffic over the tunnel). To configure split tunneling, changes to mode config parameters are needed.

For example, we will allow our road warrior clients to only access the 10.5.8.0/24 network.

```
/ip ipsec mode-conf
set [find name="rw-conf"] split-include=10.5.8.0/24
```

It is also possible to send a specific DNS server for the client to use. By default, system-dns=yes is used, which sends DNS servers that are configured on the router itself in IP/DNS. We can force the client to use a different DNS server by using the static-dns parameter.

```
/ip ipsec mode-conf
set [find name="rw-conf"] system-dns=no static-dns=10.5.8.1
```

⚠ Split networking is not a security measure. The client (initiator) can still request a different Phase 2 traffic selector.

(Optional) Assigning static IP address to user

Static IP address to any user can be assigned by use of RADIUS Framed-IP-Address attribute.

```
/user-manager user
set [find name="user1"] attributes=Framed-IP-Address:192.168.77.100 shared-users=1
```

⛔ To avoid any conflicts, the static IP address should be excluded from the IP pool of other users, as well as shared-users should be set to 1 for the specific user.

(Optional) Accounting configuration

To keep track of every user's uptime, download and upload statistics, RADIUS accounting can be used. By default RADIUS accounting is already enabled for IPsec, but it is advised to configure Interim Update timer that sends statistic to the RADIUS server regularly. If the router will handle a lot of simultaneous sessions, it is advised to increase the update timer to avoid increased CPU usage.

```
/ip ipsec settings
set interim-update=1m
```

Basic L2TP/IPsec setup

This example demonstrates how to easily set up an L2TP/IPsec server on RouterOS for road warrior connections (works with Windows, Android, iOS, macOS, and other vendor L2TP/IPsec implementations).

RouterOS server configuration

The first step is to enable the L2TP server:

```
/interface l2tp-server server
set enabled=yes use-ipsec=required ipsec-secret=mySecret default-profile=default
```

use-ipsec is set to **required** to make sure that only IPsec encapsulated L2TP connections are accepted.

Now what it does is enables an L2TP server and creates a dynamic IPsec peer with a specified secret.

```
[admin@MikroTik] /ip ipsec peer> print
0 D address=0.0.0.0/0 local-address=0.0.0.0 passive=yes port=500
auth-method=pre-shared-key secret="123" generate-policy=port-strict
exchange-mode=main-l2tp send-initial-contact=yes nat-traversal=yes
hash-algorithm=sha1 enc-algorithm=3des,aes-128,aes-192,aes-256
dh-group=modp1024 lifetime=1d dpd-interval=2m dpd-maximum-failures=5
```


Care must be taken if static IPsec peer configuration exists.

The next step is to create a VPN pool and add some users.

```
/ip pool add name=vpn-pool range=192.168.99.2-192.168.99.100

/ppp profile
set default local-address=192.168.99.1 remote-address=vpn-pool

/ppp secret
add name=user1 password=123
add name=user2 password=234
```

Now the router is ready to accept L2TP/IPsec client connections.

RouterOS client configuration

For RouterOS to work as L2TP/IPsec client, it is as simple as adding a new L2TP client.

```
/interface l2tp-client
add connect-to=1.1.1.1 disabled=no ipsec-secret=mySecret name=l2tp-out1 \
password=123 use-ipsec=yes user=user1
```

It will automatically create dynamic IPsec peer and policy configurations.

Troubleshooting/FAQ

Phase 1 Failed to get a valid proposal

```
[admin@MikroTik] /log> print
(..)
17:12:32 ipsec,error no suitable proposal found.
17:12:32 ipsec,error 10.5.107.112 failed to get valid proposal.
17:12:32 ipsec,error 10.5.107.112 failed to pre-process ph1 packet (side: 1, status 1).
17:12:32 ipsec,error 10.5.107.112 phase1 negotiation failed.
(..)
```

Peers are unable to negotiate encryption parameters causing the connection to drop. To solve this issue, enable IPSec to debug logs and find out which parameters are proposed by the remote peer, and adjust the configuration accordingly.

```
[admin@MikroTik] /system logging> add topics=ipsec,!debug
```

```
[admin@MikroTik] /log> print
(..)
17:21:08 ipsec rejected hashtype: DB(prop#1:trns#1):Peer(prop#1:trns#1) = MD5:SHA
17:21:08 ipsec rejected enctype: DB(prop#1:trns#2):Peer(prop#1:trns#1) = 3DES-CBC:AES-CBC
17:21:08 ipsec rejected hashtype: DB(prop#1:trns#2):Peer(prop#1:trns#1) = MD5:SHA
17:21:08 ipsec rejected enctype: DB(prop#1:trns#1):Peer(prop#1:trns#2) = AES-CBC:3DES-CBC
17:21:08 ipsec rejected hashtype: DB(prop#1:trns#1):Peer(prop#1:trns#2) = MD5:SHA
17:21:08 ipsec rejected hashtype: DB(prop#1:trns#2):Peer(prop#1:trns#2) = MD5:SHA
17:21:08 ipsec,error no suitable proposal found.
17:21:08 ipsec,error 10.5.107.112 failed to get valid proposal.
17:21:08 ipsec,error 10.5.107.112 failed to pre-process ph1 packet (side: 1, status 1).
17:21:08 ipsec,error 10.5.107.112 phase1 negotiation failed.
(..)
```

In this example, the remote end requires SHA1 to be used as a hash algorithm, but MD5 is configured on the local router. Setting before the column symbol (:) is configured on the local side, parameter after the column symbol (:) is configured on the remote side.

"phase1 negotiation failed due to time up" what does it mean?

There are communication problems between the peers. Possible causes include - misconfigured Phase 1 IP addresses; firewall blocking UDP ports 500 and 4500; NAT between peers not properly translating IPsec negotiation packets. This error message can also appear when a local-address parameter is not used properly. More information available here.

Random packet drops or connections over the tunnel are very slow, enabling packet sniffer/torch fixes the problem?

Problem is that before encapsulation packets are sent to Fasttrack/FastPath, thus bypassing IPsec policy checking. The solution is to exclude traffic that needs to be encapsulated/decapsulated from Fasttrack, see configuration example here.

How to enable ike2?

For basic configuration enabling ike2 is very simple, just change exchange-mode in peer settings to ike2.

fatal NO-PROPOSAL-CHOSEN notify message?

Remote peer sent notify that it cannot accept proposed algorithms, to find the exact cause of the problem, look at remote peers debug logs or configuration and verify that both client and server have the same set of algorithms.

I can ping only in one direction?

A typical problem in such cases is strict firewall, firewall rules allow the creation of new connections only in one direction. The solution is to recheck firewall rules, or explicitly accept all traffic that should be encapsulated/decapsulated.

Can I allow only encrypted traffic?

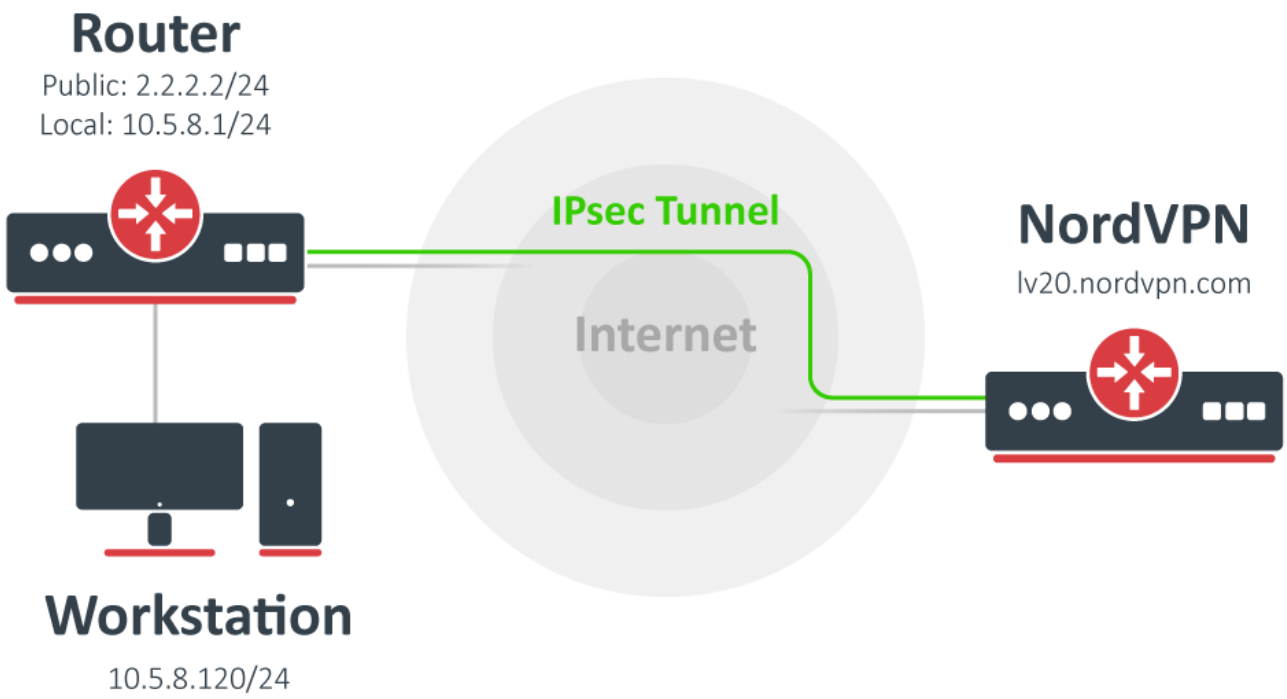
Yes, you can, see "Allow only IPsec encapsulated traffic" examples.

I enable IKEv2 REAUTH on StrongSwan and got the error 'initiator did not reauthenticate as requested'

RouterOS does not support rfc4478, reauth must be disabled on StrongSwan.

IKEv2 EAP between NordVPN and RouterOS

Starting from RouterOS v6.45, it is possible to establish IKEv2 secured tunnel to NordVPN servers using EAP authentication. This manual page explains how to configure it.



Installing the root CA

Start off by downloading and importing the NordVPN root CA certificate.

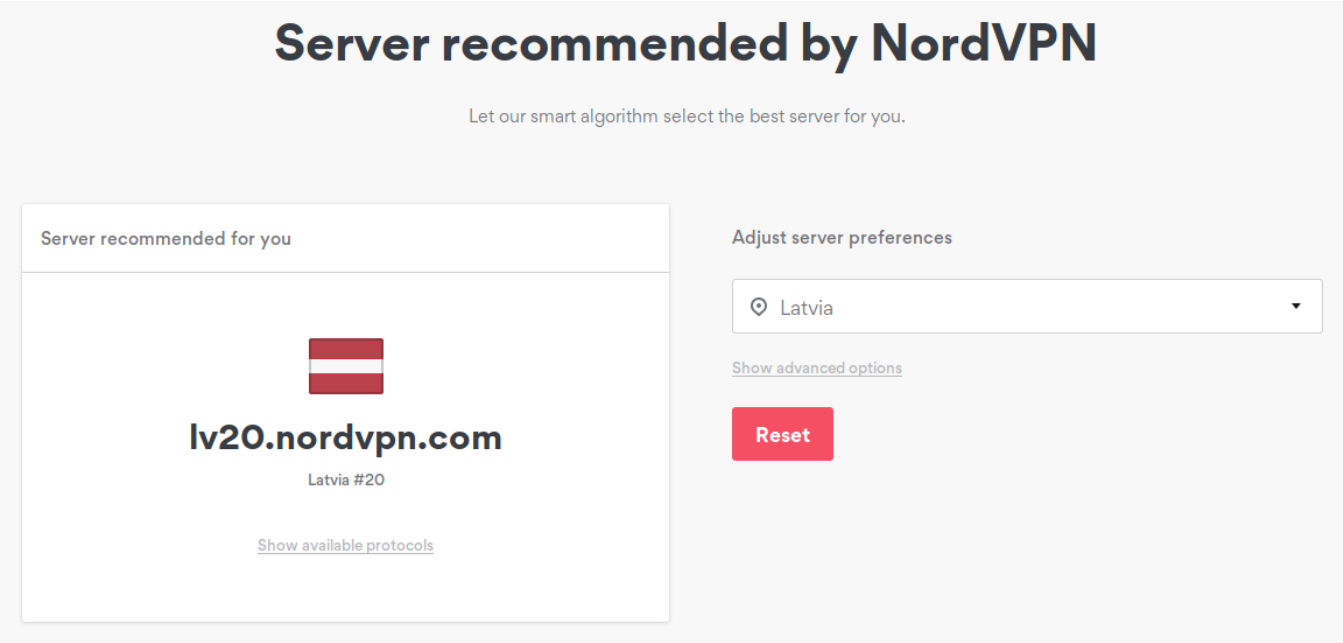
```
/tool fetch url="https://downloads.nordvpn.com/certificates/root.der"
/certificate import file-name=root.der
```

There should now be the trusted NordVPN Root CA certificate in System/Certificates menu.

```
[admin@MikroTik] > /certificate print where name~"root.der"
Flags: K - private-key, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked, E - expired, T -
trusted
 #         NAME            COMMON-NAME            SUBJECT-ALT-NAME
FINGERPRINT
 0       T root.der_0      NordVPN Root CA
8b5a495db498a6c2c8c...
```

Finding out the server's hostname

Navigate to https://nordvpn.com/servers/tools/ and find out the recommended server's hostname. In this case, it is lv20.nordvpn.com.



Setting up the IPsec tunnel

It is advised to create a separate Phase 1 profile and Phase 2 proposal configurations to not interfere with any existing or future IPsec configuration.

```
/ip ipsec profile
add name=NordVPN
/ip ipsec proposal
add name=NordVPN pfs-group=none
```

While it is possible to use the default policy template for policy generation, it is better to create a new policy group and template to separate this configuration from any other IPsec configuration.

```
/ip ipsec policy group
add name=NordVPN
/ip ipsec policy
add dst-address=0.0.0.0/0 group=NordVPN proposal=NordVPN src-address=0.0.0.0/0 template=yes
```

Create a new mode config entry with responder=no that will request configuration parameters from the server.

```
/ip ipsec mode-config
add name=NordVPN responder=no
```

Lastly, create peer and identity configurations. Specify your NordVPN credentials in username and password parameters.

```
/ip ipsec peer
add address=lv20.nordvpn.com exchange-mode=ike2 name=NordVPN profile=NordVPN
/ip ipsec identity
add auth-method=eap certificate="" eap-methods=eap-mschapv2 generate-policy=port-strict mode-config=NordVPN
peer=NordVPN policy-template-group=NordVPN username=support@mikrotik.com password=secret
```

Verify that the connection is successfully established.

```
/ip ipsec
active-peers print
installed-sa print
```

Choosing what to send over the tunnel

If we look at the generated dynamic policies, we see that only traffic with a specific (received by mode config) source address will be sent through the tunnel. But a router in most cases will need to route a specific device or network through the tunnel. In such a case, we can use source NAT to change the source address of packets to match the mode config address. Since the mode config address is dynamic, it is impossible to create a static source NAT rule. In RouterOS it is possible to generate dynamic source NAT rules for mode config clients.

Option 1: Sending all traffic over the tunnel

In this example, we have a local network 10.5.8.0/24 behind the router and we want all traffic from this network to be sent over the tunnel. First of all, we have to make a new IP/Firewall/Address list which consists of our local network.

```
/ip firewall address-list
add address=10.5.8.0/24 list=local
```

It is also possible to specify only single hosts from which all traffic will be sent over the tunnel. Example:

```
/ip firewall address-list
add address=10.5.8.120 list=local
add address=10.5.8.23 list=local
```

When it is done, we can assign newly created IP/Firewall/Address list to mode config configuration.

```
/ip ipsec mode-config
set [ find name=NordVPN ] src-address-list=local
```

Verify correct source NAT rule is dynamically generated when the tunnel is established.

```
[admin@MikroTik] > /ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic
 0  D ;;; ipsec mode-config
      chain=srcnat action=src-nat to-addresses=192.168.77.254 src-address-list=local dst-address-list=!local
```

Warning

Make sure the dynamic mode config address is not a part of the local network.

It is also possible to combine both options (1 and 2) to allow access to specific addresses only for specific local addresses/networks

Option 2: Accessing certain addresses over the tunnel

It is also possible to send only specific traffic over the tunnel by using the connection-mark parameter in the Mangle firewall. It works similarly as Option 1 - a dynamic NAT rule is generated based on configured connection-mark parameter under mode config.

First of all, set the connection-mark under your mode config configuration.

```
/ip ipsec mode-config
set [ find name=NordVPN ] connection-mark=NordVPN
```

When it is done, a NAT rule is generated with the dynamic address provided by the server:

```
[admin@MikroTik] > /ip firewall nat print
Flags: X - disabled, I - invalid, D - dynamic
 0  D ;;; ipsec mode-config
      chain=srcnat action=src-nat to-addresses=192.168.77.254 connection-mark=NordVPN
```

After that, it is possible to apply this connection-mark to any traffic using Mangle firewall. In this example, access to mikrotik.com and 8.8.8.8 is granted over the tunnel.

Create a new address list:

```
/ip firewall address-list
add address=mikrotik.com list=VPN
add address=8.8.8.8 list=VPN
```

Apply connection-mark to traffic matching the created address list:

```
/ip firewall mangle
add action=mark-connection chain=prerouting dst-address-list=VPN new-connection-mark=NordVPN passthrough=yes
```

It is also possible to combine both options (1 and 2) to allow access to specific addresses only for specific local addresses/networks

L2TP

Overview

Layer Two Tunneling Protocol "L2TP" extends the PPP model by allowing the L2 and PPP endpoints to reside on different devices interconnected by a packet-switched network. L2TP includes PPP authentication and accounting for each L2TP connection. Full authentication and accounting of each connection may be done through a RADIUS client or locally. L2TP traffic uses UDP protocol for both control and data packets. UDP port 1701 is used only for link establishment, further traffic is using any available UDP port (which may or may not be 1701). This means that L2TP can be used with most firewalls and routers (even with NAT) by enabling UDP traffic to be routed through the firewall or router. L2TP standard is defined in RFC 2661. The L2TPv3 support added in 7.1 version. Support IPv4, IPv6.

Introduction

It may be useful to use L2TP just as any other tunneling protocol with or without encryption. The L2TP standard says that the most secure way to encrypt data is using L2TP over IPsec (Note that it is the default mode for Microsoft L2TP client) as all L2TP control and data packets for a particular tunnel appear as homogeneous UDP/IP data packets to the IPsec system.

Multilink PPP (MP) is supported in order to provide MRRU (the ability to transmit full-sized 1500 and larger packets) and bridging over PPP links (using Bridge Control Protocol (BCP) that allows sending raw Ethernet frames over PPP links). This way it is possible to setup bridging without EoIP. The bridge should either have an administratively set MAC address or an Ethernet-like interface in it, as PPP links do not have MAC addresses.

L2TP does not provide encryption mechanisms for tunneled traffic. IPsec can be used for additional security layers.

L2TP Client

Properties

Property	Description
add-default-route (*yes* | *no*; Default: **no**)	Whether to add L2TP remote address as a default route.
allow (*mschap2* | *mschap1* | *chap* | *pap*; Default: **mschap2, mschap1, chap, pap**)	Allowed authentication methods.

connect-to (*IP|IPv6*; Default:)	Remote address of L2TP server (if the address is in VRF table, VRF should be specified)
	`/interface l2tp-client`
	`add connect-to=192.168.88.1@vrf1 name=l2tp-out1 user=l2tp-client)`

comment (*string*; Default:)	Short description of the tunnel.
default-route-distance (*byte*; Default:)	Since v6.2, sets distance value applied to auto created default route, if add-default-route is also selected
dial-on-demand (*yes* | *no*; Default: **no**)	connects only when outbound traffic is generated. If selected, then route with gateway address from 10.112.112.0/24 network will be added while connection is not established.
disabled (*yes* | *no*; Default: **yes**)	Enables/disables tunnel.
keepalive-timeout (*integer [1..4294967295]* ; Default: **60s**)	Since v6.0rc13, tunnel keepalive timeout in seconds.
max-mru (*integer*; Default: **1450**)	Maximum Receive Unit. Max packet size that L2TP interface will be able to receive without packet fragmentation.
max-mtu (*integer*; Default: **1450**)	Maximum Transmission Unit. Max packet size that L2TP interface will be able to send without packet fragmentation.

mrru (*disabled* | *integer*; Default: **disabled**)	Maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.
name (*string*; Default:)	Descriptive name of the interface.
password (*string*; Default: **""**)	Password used for authentication.
profile (*name*; Default: **default-encryption**)	Specifies which PPP profile configuration will be used when establishing the tunnel.
user (*string*; Default:)	User name used for authentication.
use-ipsec (*yes* | *no*; Default: **no**)	When this option is enabled, dynamic IPSec peer configuration and policy is added to encapsulate L2TP connection into IPSec tunnel.
allow-fast-path (*yes* | *no*; Default:)	Allow to forward packets without additional processing in the Linux kernel.
l2tp-proto-version (l2tpv2 | *l2tpv3-ip* | *l2tpv3-udp* | *l2tpv*; Default: **l2tpv2**)	Specify protocol version.
l2tpv3-cookie-length (0 | *4-bytes* | *8-bytes* ; Default: **0**)	Configures an L2TPv3 pseudowire static session cookie.
l2tpv3-digest-hash (*md5* | *none* | *sha1* ; Default: **md5**)	Specifies which hash function to be used.
use-peer-dns (*yes* | *no* | *exclusively*; Default: **no**)	To use peer dns.
copy-from	To copy created peer.
src-address	Specify source address.
l2tpv3-circuit-id	Set the virtual circuit identifier to bind the one end of the L2TPv3 control channel.
ipsec-secret (*string*; Default:)	Preshared key used when use-ipsec is enabled.

L2TP Server

An interface is created for each tunnel established to the given server. There are two types of interfaces in the L2TP server's configuration

- Static interfaces are added administratively if there is a need to reference the particular interface name (in firewall rules or elsewhere) created for the particular user;
- Dynamic interfaces are added to this list automatically whenever a user is connected and its username does not match any existing static entry (or in case the entry is active already, as there can not be two separate tunnel interfaces referenced by the same name);

Dynamic interfaces appear when a user connects and disappear once the user disconnects, so it is impossible to reference the tunnel created for that use in router configuration (for example, in firewall), so if you need persistent rules for that user, create a static entry for him/her. Otherwise, it is safe to use a dynamic configuration.

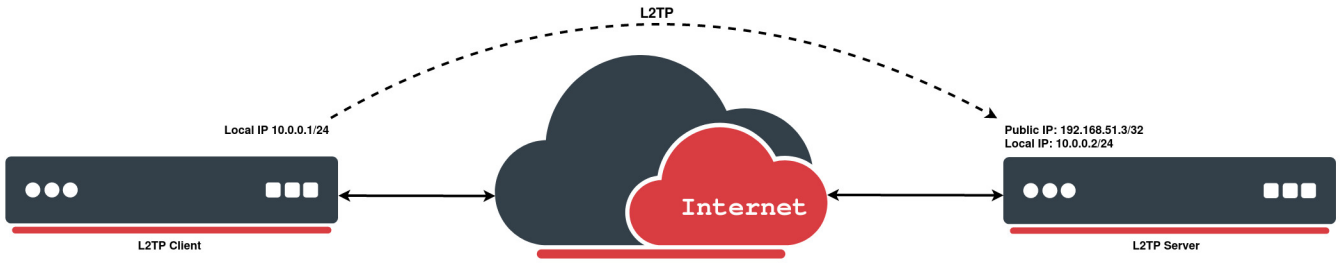
in both cases PPP users must be configured properly - static entries do not replace PPP configuration.

Properties

Property	Description
authentication (*pap* | *chap* | *mschap1* | *mschap2*; Default: **mschap1,mschap2**)	Authentication methods that server will accept.
default-profile (*name*; Default: **default-encryption**)	default profile to use
enabled (*yes* | *no*; Default: **no**)	Defines whether L2TP server is enabled or not.

max-mru (*integer*; Default: **1450**)	Maximum Receive Unit. Max packet size that L2TP interface will be able to receive without packet fragmentation.
keepalive-timeout (*integer*; Default: **30**)	If server during keepalive-timeout period does not receive any packets, it will send keepalive packets every second, five times. If the server still does not receive any response from the client, then the client will be disconnected after 5 seconds. Logs will show 5x "LCP missed echo reply" messages and then disconnect.
max-mtu (*integer*; Default: **1450**)	Maximum Transmission Unit. Max packet size that L2TP interface will be able to send without packet fragmentation.
use-ipsec (*no* / *yes* / *require*; Default: **no**)	When this option is enabled, dynamic IPSec peer configuration is added to suite most of the L2TP road-warrior setups. When require is selected server will accept only those L2TP connection attempts that were encapsulated in the IPSec tunnel.
ipsec-secret (*string*; Default:)	Preshared key used when use-ipsec is enabled
accept-proto-version (all / l2tpv2 / l2tpv3; Default: **all**) cli-only	Specify protocol version.
accept-pseudowire-type (all / ether / ppp; Default: **all**)	Set the pseudowire signaling protocol for specific pseudowire type.
allow-fast-path (*no* / *yes*; Default: **no**)	To forward packets without additional processing in the Linux kernel.
caller-id-type (ip-address / number; Default: **ip-address**)	If same source IP address is used for multiple clients set id type to number.
max-sessions (unlimited / number; Default: **unlimited**)	Set number of needed sessions.
one-session-per-host (*no* / *yes* / ; Default: **no**)	To allow one session per host.
l2tpv3-circuit-id (Default:)	Set the virtual circuit identifier to bind the one end of the L2TPv3 control channel.
l2tpv3-cookie-length (0 / 4-bytes / 8-bytes; Default: **0**)	Configures an L2TP pseudowire static session cookie.
l2tpv3-digest-hash (md5 / none / sha1; Default: **md5**)	Specifies which hash function to be used.
l2tpv3-ether-interface-list (Default:)	Set your interface list for example the default ones- all, dynamic, none, static.
mrru (*disabled* / *integer*; Default: **disabled**)	Maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.

Quick Example



L2TP Server

On the servers side we will enable L2TP-server and create a PPP profile for a particular user:

```
[admin@MikroTik] > /interface l2tp-server server set enabled=yes
[admin@MikroTik] > /ppp secret add local-address=10.0.0.2 name=MT-User password=StrongPass profile=default-
encryption remote-address=10.0.0.1 service=l2tp
```

L2TP Client

L2TP client setup in the RouterOS is very simple. In the following example, we already have a preconfigured 3 unit setup. We will take a look more detailed on how to set up L2TP client with username "MT-User", password "StrongPass" and server 192.168.51.3:

```
[admin@MikroTik] > /interface l2tp-client \
add connect-to=192.168.51.3 disabled=no name=MT-User password=StrongPass user=MT-User
[admin@MikroTik] > /interface l2tp-client print
Flags: X - disabled, R - running
0 R name="MT-User" max-mtu=1450 max-mru=1450 mrru=disabled connect-to=192.168.51.3 user="MT-User"
password="StrongPass" profile=default-encryption keepalive-timeout=60 use-ipsec=no ipsec-secret=""
allow-fast-path=no add-default-route=no dial-on-demand=no allow=pap,chap,mschap1,mschap2
```

L2TP Ether

Overview

Layer 2 Tunnel Protocol Version 3 (L2TPv3) is a draft from the Internet Engineering Task Force (IETF) working group. It introduces various improvements to the original L2TP, allowing the encapsulation of Layer 2 (L2) payloads within L2TP. More precisely, L2TPv3 outlines the protocol for tunnelling Layer 2 payloads across an IP core network using L2 virtual private networks (VPNs).

Properties

Property	Description
connect-to (*IP*; Default:)	Remote address of L2TP server.
comment (st*ring*; Default:)	Short description of the tunnel.
disabled (*yes* | *no*; Default: **yes**)	Enables/disables tunnel.
mac-address (string; Default: **auto**)	Set desired mac address of interface.
unmanaged-mode (*yes* | *no*; Default: **no**)	Set unmanaged mode active, the configuration for additional settings will be possible, such as: **peer-cookie, send-cookie, local-tunnel-id, local-session-id, remote-tunnel-id, remote-session-id, local-address.**
local-tunnel-id (string; Default: disabled)	Set value for local-tunnel-id, an integer required.
local-session-id (string; Default: disabled)	Set value for local-session-id, an integer required.
remote-tunnel-id (string; Default: **disabled)**	Set value for remote-tunnel-id, an integer required.
remote-session-id (string; Default: **disabled)**	Set value for remote-session-id, an integer required.
peer-cookie (string; Default: **disabled)**	Sets optional peer cookie. To enable cookie enter remote cookie value (8 or 16 character hex string value expected) to disable leave empty.
send-cookie (string; Default: **disabled)**	Sets optional cookie. To enable cookie enter remote cookie value (8 or 16 character hex string value expected) to disable leave empty.

mtu (*auto*; Default: **1420**)	Maximum Transmission Unit. Max packet size that L2TP interface will be able to send without packet fragmentation.
name (*string*; Default:)	Descriptive name of the interface.
local-address (*IP address*; Default:)	Set local address for **unmanaged** mode.
use-ipsec (*yes* / *no*; Default: **no**)	When this option is enabled, dynamic IPSec peer configuration and policy is added to encapsulate L2TP connection into IPSec tunnel.
allow-fast-path (*yes* / *no*; Default: **no**)	Allow to forward packets without additional processing in the Linux kernel.
l2tp-proto-version (*l2tpv3-ip* / *l2tpv3-udp* / ; Default: ***l2tpv3-udp***)	Specify protocol version.
cookie-length (0 / *4-bytes* / *8-bytes* ; Default: **0**)	Configures an L2TPv3 pseudowire static session cookie.
digest-hash (*md5* / *none* / *sha1* ; Default: **md5**)	Specifies which hash function to be used.
use-l2-specific-sublayer (*yes* / *no*; Default: **no**)	Specify source address.
circuit-id	Set the virtual circuit identifier to bind the one end of the L2TPv3 control channel, this works as identifier for each redundant pseudowire.
ipsec-secret (*string*; Default:)	Preshared key used when use-ipsec is enabled.

LAC and LNS setup with Cisco as LAC

Overview

LAC/LNS setup or otherwise known as Virtual Private DialUp Network (VPDN) allows long-distance point-to-point connection between remote dial-up users and private networks.

Dial-up client uses PPPOE to connect to a L2TP access concentrator (LAC), LAC determines that session should be forwarded through a IP network to the L2TP Network Server (LNS), creates L2TP tunnel and forwards PPP frames to the server where client is authenticated and session established (see diagram below).

At the time of writing this article RouterOS cannot be used in LAC role. For this reason article will demonstrate how to set up very basic network with RouterOS as LNS and Cisco router as LAC.

Configuration

We will be using simple configuration to demonstrate very basics of VPDN setup. Lets assume that LAC will forward to the LNS clients with FQDN name containing mt.lv domain.

Client

For the sake of simplicity lets assume that client is RouterOS router:

```
/interface pppoe-client add interface=ether1 user=good_worker@mt.lv password=strongpass
```

LAC

Lets assume that client is connected to the GigabitEthernet1 port and IP address of the LNS server is 10.155.101.231

```
aaa new-model
!
aaa authentication ppp default local
!
vpdn enable
vpdn aaa attribute nas-ip-address vpdn-nas
vpdn search-order domain dnis
!
vpdn-group LAC
 request-dialin
  protocol l2tp
  domain mt.lv
 initiate-to ip 10.155.101.231
 source-ip 10.155.101.216
 local name LAC
 l2tp tunnel password 0 tunnelpass
!
bba-group pppoe MAIN-BBA
 virtual-template 1
!
interface GigabitEthernet1
 pppoe enable group MAIN-BBA
!
interface Virtual-Template1
 description pppoe MAIN-BBA
 no ip address
 no peer default ip address
 ppp mtu adaptive
 ppp authentication chap
!
```

Note that this setup does not authenticate client nor locally nor via RADIUS, does not actually check domain name, does not control L2 access for the sake of simplicity. If you want to use those features refer to Cisco configuration manuals.

LNS

On the LNS we need to enable L2TP server and set up method to authenticate the L2TP connection from the LAC.

```
/interface l2tp-server server
set enabled=yes
/ppp l2tp-secret
add address=10.155.101.216/32 secret=tunnelpass
```

Now the actual user authentication. In this case we will be using local authentication method for the sake of simplicity.

```
/ip pool
add name=pool0 ranges=192.168.99.2-192.168.99.99
/ppp profile
set default local-address=192.168.99.1 remote-address=pool0
/ppp secret
add name=good_worker@mt.lv password=strongpass
```

Status Check

On the LNS you can see all successfully connected clients by checking l2tp server interfaces or checking active ppp connections:

```
[admin@CHR_v6_bgp] /interface l2tp-server> print
Flags: X - disabled, D - dynamic, R - running
# NAME USER MTU CLIENT-ADDRESS UPTIME ENCODING
0 DR <l2tp-... good_worker@mt.lv 1450 10.155.101.216 6h13m49s

[admin@CHR_v6_bgp] /ppp active> print
Flags: R - radius
# NAME SERVICE CALLER-ID ADDRESS UPTIME ENCODING
0 good_worker@mt.lv l2tp 10.155.101.216 192.168.99.2 6h15m57s
```

On the LAC we can also see active client sessions and active L2TP tunnel between LAC and LNS:

```
csrLAC#show vpdn

L2TP Tunnel and Session Information Total tunnels 1 sessions 1

LocTunID RemTunID Remote Name State Remote Address Sessn L2TP Class/
Count VPDN Group
26090 11 CHR_v6_bgp est 10.155.101.231 50 LAC

LocID RemID TunID Username, Intf/ State Last Chg Uniq ID
Vcid, Circuit
18521 16 26090 good_worker@mt.lv, Gi1 est 06:17:07 571
```

Session Establishment

Lets look closely on how clients sessions gets authenticated and established over the LAC.

- Client initiates PPPoE call
- LAC and Client begins LCP negotiation
- after CHAP has been negotiated, LAC sends CHAP challenge
- Client sends CHAP response
- LAC checks whether client session should be forwarded to the LNS based on received domain name. Check can be done locally or using RADIUS server. Client also can be authenticated here before forwarding session.
- LAC brings up an L2TP tunnel
- LNS checks if the LAC is allowed to open a tunnel and run the authentication process. The Tunnel is up and ready to forward VPDN sessions.
- LAC forwards negotiated with the client LCP options, username and password to the LNS
- LNS authenticates the client locally or using RADIUS and sends CHAP response
- IP Control Protocol (IPCP) phase is performed, IP addresses and routes are installed. At this point sessions is considered established.

OpenVPN

Overview

The OpenVPN security model is based on SSL, the industry standard for secure communications via the internet. OpenVPN implements OSI layer 2 or 3 secure network extensions using the SSL/TLS protocol. Support IPv4, IPv6.

Introduction

OpenVPN has been ported to various platforms, including Linux and Windows, and its configuration is likewise on each of these systems, so it makes it easier to support and maintain. OpenVPN can run over User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) transports, multiplexing created SSL tunnels on a single TCP/UDP port. OpenVPN is one of the few VPN protocols that can make use of a proxy, which might be handy sometimes.

Limitations

Currently, unsupported OpenVPN features:

- LZO compression
- authentication without username/password

OpenVPN username is limited to 27 characters and the password to 233 characters.

OVPN Client

Property	Description
add-default-route (*yes* | *no*; Default: **no**)	Whether to add OVPN remote address as a default route.
auth (*md5* | *sha1* | *null* | *sha256* | *sha512*; Default: **sha1**)	Allowed authentication methods.
certificate (*string* | *none*; Default: **none**)	Name of the client certificate
cipher (*null* | *aes128-cbc* | *aes128-gcm* | *aes192-cbc* | *aes192-gcm* | *aes256-cbc* | *aes256-gcm* | *blowfish128*; Default: **blowfish128**)	Allowed ciphers. In order to use GCM type ciphers, the "auth" parameter must be set to "null", because GCM cipher is also responsible for "auth", if used.
comment (*string*; Default:)	Descriptive name of an item
connect-to (*IP|IPv6*; Default:)	Remote address of the OVPN server.
disabled (*yes* | *no*; Default: **yes**)	Whether the interface is disabled or not. By default it is disabled.
mac-address (*MAC*; Default:)	Mac address of OVPN interface. Will be automatically generated if not specified.
max-mtu (*integer*; Default: **1500**)	Maximum Transmission Unit. Max packet size that the OVPN interface will be able to send without packet fragmentation.

mode (*ip* | *ethernet*; Default: **ip**)	Layer3 or layer2 tunnel mode (alternatively tun, tap)
name (*string*; Default:)	Descriptive name of the interface.
password (*string*; Default: **""**)	Password used for authentication.
port (*integer*; Default: **1194**)	Port to connect to.
profile (*name*; Default: **default**)	Specifies which PPP profile configuration will be used when establishing the tunnel.
protocol (*tcp* | *udp*; Default: **tcp**)	indicates the protocol to use when connecting with the remote endpoint.
verify-server-certificate (*yes* | *no*; Default: **no**)	Checks the certificates CN or SAN against the "connect-to" parameter. The IP or hostname must be present in the server's certificate.
tls-version (*any* | *only-1.2*; Default: **any**)	Specifies which TLS versions to allow
use-peer-dns (*yes* | *no*; Default: **no**)	Whether to add DNS servers provided by the OVPN server to IP/DNS configuration.
route-nopull (*yes* | *no*; Default: **no**)	Specifies whether to allow the OVPN server to add routes to the OVPN client instance routing table.
user (*string*; Default:)	User name used for authentication.

Also, it is possible to import the OVPN client configuration from a .ovpn configuration file. Such a file usually is provided from the OVPN server side and already includes configuration so you need to worry only about a few parameters.

```
/interface/ovpn-client/import-ovpn-configuration ovpn-password=securepassword \
key-passphrase=certificatekeypassphrase ovpn-user=myuserid skip-cert-import=no
```

OVPN client supports tls authentication. The configuration of tls-auth can be added only by importing .ovpn configuration file. Using tls-auth requires that you generate a shared-secret key, this key should be added to the client configuration file .ovpn.

```
key-direction 1
<tls-auth>
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
-----END OpenVPN Static key V1-----
</tls-auth>
```

OVPN Server

An interface is created for each tunnel established to the given server. There are two types of interfaces in the OVPN server's configuration

- Static interfaces are added administratively if there is a need to reference the particular interface name (in firewall rules or elsewhere) created for the particular user.
- Dynamic interfaces are added to this list automatically whenever a user is connected and its username does not match any existing static entry (or in case the entry is active already, as there can not be two separate tunnel interfaces referenced by the same name).

Dynamic interfaces appear when a user connects and disappear once the user disconnects, so it is impossible to reference the tunnel created for that use in router configuration (for example, in the firewall), so if you need a persistent rule for that user, create a static entry for him/her. Otherwise, it is safe to use dynamic configuration.

In both cases PPP users must be configured properly - static entries do not replace PPP configuration.

Properties

Property	Description
auth (*md5* | *sha1* | *null* | *sha256* | *sha512*; Default: **sha1,md5,sha256,sha512**)	Authentication methods that the server will accept.
certificate (*name* | *none*; Default: **none**)	Name of the certificate that the OVPN server will use.
cipher (*null* | *aes128-cbc* | *aes128-gcm* | *aes192-cbc* | *aes192-gcm* | *aes256-cbc* | *aes256-gcm* | *blowfish128*; Default: **aes128-cbc,blowfish128**)	Allowed ciphers.
default-profile (*name*; Default: **default**)	Default profile to use.
enabled (*yes* | *no*; Default: **no**)	Defines whether the OVPN server is enabled or not.
protocol (*tcp* | *udp*; Default: tcp)	Indicates the protocol to use when connecting with the remote endpoint.
keepalive-timeout (*integer* | *disabled*; Default: **60**)	Defines the time period (in seconds) after which the router is starting to send keepalive packets every second. If no traffic and no keepalive responses have come for that period of time (i.e. 2 * keepalive-timeout), not responding client is proclaimed disconnected
mac-address (*MAC*; Default:)	Automatically generated MAC address of the server.
max-mtu (*integer*; Default: **1500**)	Maximum Transmission Unit. Max packet size that the OVPN interface will be able to send without packet fragmentation.
mode (*ip* | *ethernet*; Default: **ip**)	Layer3 or layer2 tunnel mode (alternatively tun, tap)
netmask (*integer*; Default: **24**)	Subnet mask to be applied to the client.
port (*integer*; Default: **1194**)	Port to run the server on.
require-client-certificate (*yes* | *no*; Default: **no**)	If set to yes, then the server checks whether the client's certificate belongs to the same certificate chain.
redirect-gateway (*def1* | *disabled* | *ipv6;* Default: **disabled**)	Specifies what kind of routes the OVPN client must add to the routing table. `def1` – Use this flag to override the default gateway by using 0.0.0.0/1 and 128.0.0.0/1 rather than 0.0.0.0/0. This has the benefit of overriding but not wiping out the original default gateway. `disabled` - Do not send redirect-gateway flags to the OVPN client. `ipv6` - Redirect IPv6 routing into the tunnel on the client side. This works similarly to the def1 flag, that is, more specific IPv6 routes are added (2000::/4 and 3000::/4), covering the whole IPv6 unicast space.
enable-tun-ipv6 (*yes* | *no;* Default: **no**)	Specifies if IPv6 IP tunneling mode should be possible with this OVPN server.
ipv6-prefix-len (*integer;* Default: **64**)	Length of IPv6 prefix for IPv6 address which will be used when generating OVPN interface on the server side.
reneg-sec (*integer;* Default: **3600)**	Key renegotiate seconds, the time the server periodically renegotiates the secret key for the data channel.
push-routes (*string*; Default:)	Push route support are added in 7.14, the maximum of possible input is limited to 1400 characters.
tls-version (any | *only-1.2 ;* Default: **any**)	TLS protocol setting.
tun-server-ipv6 (*IPv6 prefix;* Default: **::**)	IPv6 prefix address which will be used when generating the OVPN interface on the server side.

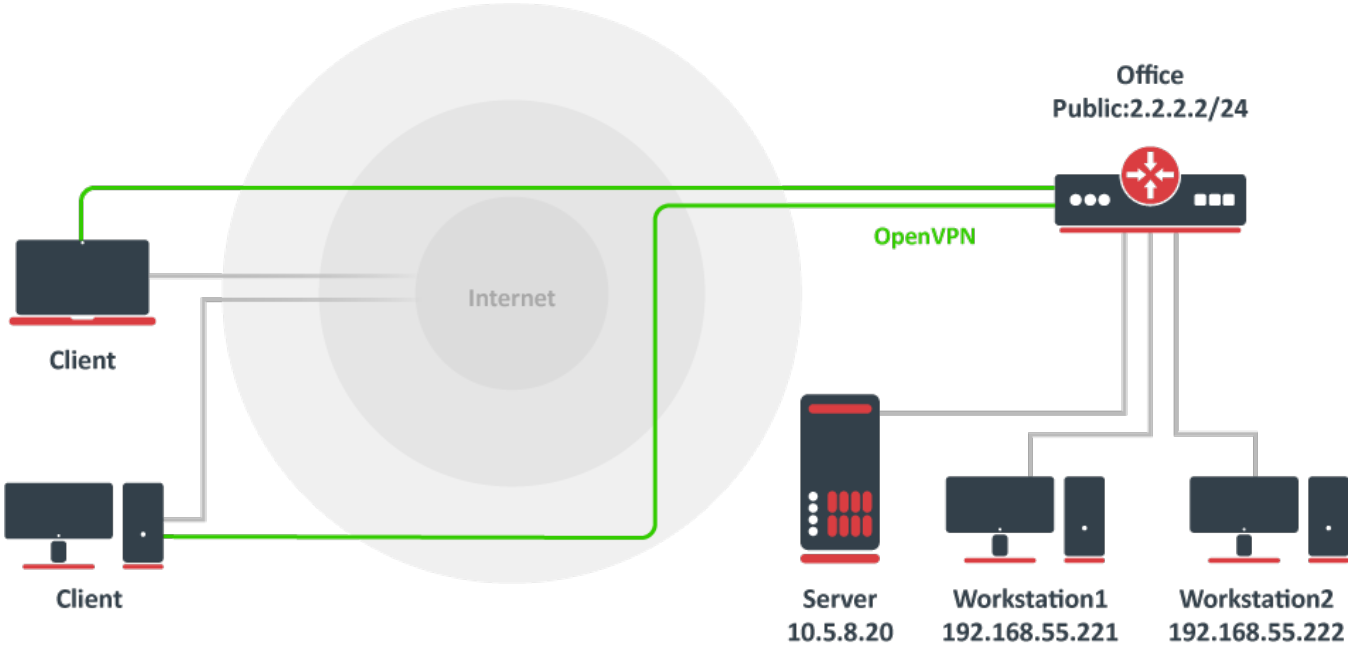
Also, it is possible to prepare a .ovpn file for the OVPN client which can be easily imported on the end device.

```
/interface/ovpn-server/server/export-client-configuration ca-certificate=myCa.crt \
client-certificate=client1.crt client-cert-key=client1.key server-address=192.168.88.1
```

It is very important that the date on the router is within the range of the installed certificate's date of expiration. To overcome any certificate verification problems, enable **NTP** date synchronization on both the server and the client.

Example

Setup Overview



Assume that Office public IP address is 2.2.2.2 and we want two remote OVPN clients to have access to 10.5.8.20 and 192.168.55.0/24 networks behind the office gateway.

Creating Certificates

All certificates can be created on the RouterOS server using the certificate manager. See example >>.

For the simplest setup, you need only an OVPN server certificate.

Server Config

The first step is to create an IP pool from which client addresses will be assigned and some users.

```
/ip pool add name=ovpn-pool range=192.168.77.2-192.168.77.254

/ppp profile add name=ovpn local-address=192.168.77.1 remote-address=ovpn-pool
/ppp secret
add name=client1 password=123 profile=ovpn
add name=client2 password=234 profile=ovpn
```

Assume that the server certificate is already created and named "server"

```
/interface ovpn-server server set enabled=yes certificate=server
```

Client Config

Push route support are added in 7.14, the maximum of possible input is limited to 1400 characters.
example: route network/IP [netmask] [gateway] [metric].

```
/interface ovpn-server server set push-routes="192.168.102.0 255.255.255.0 192.168.109.1 9"
```

To add manually which networks you want to access over the tunnel.

```
/interface ovpn-client
add name=ovpn-client1 connect-to=2.2.2.2 user=client1 password=123 disabled=no
/ip route
add dst-address=10.5.8.20 gateway=ovpn-client1
add dst-address=192.168.55.0/24 gateway=ovpn-client1
/ip firewall nat add chain=srcnat action=masquerade out-interface=ovpn-client1
```

PPPoE

Overview

Point to Point over Ethernet (PPPoE) is simply a method of encapsulating PPP packets into Ethernet frames. PPPoE is an extension of the standard Point to Point Protocol (PPP) and it the successor of PPPoA. PPPoE standard is defined in RFC 2516. The PPPoE client and server work over any Layer2 Ethernet level interface on the router, for example, Wireless, Ethernet, EoIP, etc. Generally speaking, PPPoE is used to hand out IP addresses to clients based on authentication by username (and also if required, by workstation) as opposed to workstation only authentication where static IP addresses or DHCP are used. It is advised not to use static IP addresses or DHCP on the same interfaces as PPPoE for obvious security reasons.

Introduction

PPPoE provides the ability to connect a network of hosts over a simple bridging access device to a remote Access Concentrator.

Supported connections:

- MikroTik RouterOS PPPoE client to any PPPoE server;
- MikroTik RouterOS server (access concentrator) to multiple PPPoE clients (clients are available for almost all operating systems and most routers);

PPPoE Operation

PPPoE has two distinct stages(phases):

1. Discovery phase;
2. Session phase;

Discovery phase

There are four steps to the Discovery stage. When it completes, both peers know the PPPoE *SESSION_ID* and the peer's Ethernet address, which together define the PPPoE session uniquely:

1. **PPPoE Active Discovery Initialization (PADI) -** The PPPoE client sends out a *PADI* packet to the broadcast address. This packet can also populate the "service-name" field if a service name has been entered in the dial-up networking properties of the PPPoE client. If a service name has not been entered, this field is not populated
2. **PPPoE Active Discovery Offer (PADO) -** The PPPoE server, or Access Concentrator, should respond to the *PADI* with a *PADO* if the Access Concentrator is able to service the "service-name" field that had been listed in the *PADI* packet. If no "service-name" field had been listed, the Access Concentrator will respond with a *PADO* packet that has the "service-name" field populated with the service names that the Access Concentrator can service. The *PADO* packet is sent to the unicast address of the PPPoE client
3. **PPPoE Active Discovery Request (PADR) -** When a *PADO* packet is received, the PPPoE client responds with a *PADR* packet. This packet is sent to the unicast address of the Access Concentrator. The client may receive multiple *PADO* packets, but the client responds to the first valid *PA*

DO that the client received. If the initial *PADI* packet had a blank "service-name" field filed, the client populates the "service-name" field of the *PADR* packet with the first service name that had been returned in the *PADO* packet.

4. **PPPoE Active Discovery Session Confirmation (PADS) -** When the *PADR* is received, the Access Concentrator generates a unique session identification (ID) for the Point-to-Point Protocol (PPP) session and returns this ID to the PPPoE client in the *PADS* packet. This packet is sent to the unicast address of the client.

PPPoE session termination:

- **PPPoE Active Discovery Terminate (PADT) -** Can be sent anytime after a session is established to indicate that a PPPoE session terminated. It can be sent by either server or client.

Session phase

When the discovery stage is completed, both peers know *PPPoE Session ID* and other peer's *Ethernet (MAC) address* which together defines the PPPoE session. PPP frames are encapsulated in PPPoE session frames, which have Ethernet frame type **0x8864**.
When a server sends confirmation and a client receives it, PPP Session is started that consists of the following stages:

1. **LCP negotiation** stage
2. **Authentication (CHAP/PAP)** stage
3. **IPCP negotiation** stage - where the client is assigned an IP address.

> ⚠ If any process fails, the LCP negotiation establishment phase is started again.

PPPoE server sends *Echo-Request* packets to the client to determine the state of the session, otherwise, the server will not be able to determine that session is terminated in cases when a client terminates session without sending *Terminate-Request* packet.

MTU

Typically, the largest Ethernet frame that can be transmitted without fragmentation is 1500 bytes. PPPoE adds another 6 bytes of overhead and the PPP field adds two more bytes, leaving 1492 bytes for IP datagram. Therefore max PPPoE MRU and MTU values must not be larger than 1492.

TCP stacks try to avoid fragmentation, so they use an MSS (Maximum Segment Size). By default, MSS is chosen as MTU of the outgoing interface minus the usual size of the TCP and IP headers (40 bytes), which results in 1460 bytes for an Ethernet interface. Unfortunately, there may be intermediate links with lower MTU which will cause fragmentation. In such a case TCP stack performs path MTU discovery. Routers that cannot forward the datagram without fragmentation are supposed to drop the packet and send *ICMP-Fragmentation-Required* to originating host. When a host receives such an ICMP packet, it tries to lower the MTU. This should work in the ideal world, however in the real world many routers do not generate fragmentation-required datagrams, also many firewalls drop all ICMP datagrams.

The workaround for this problem is to adjust MSS if it is too big.

PPPoE Client

Properties

Property	Description
ac-name (*string*; Default: **""**)	Access Concentrator name, this may be left blank and the client will connect to any access concentrator on the broadcast domain
add-default-route (*yes*|*no*; Default: **no**)	Enable/Disable whether to add default route automatically
allow (*mschap2*|*mschap1*|*chap*|*pap*; Default: **mschap2,mschap1,chap,pap**)	allowed authentication methods, by default all methods are allowed
default-route-distance (*byte [0..255]*; Default:**1**)	sets distance value applied to auto created default route, if add-default-route is also selected
dial-on-demand (*yes*|*no*; Default: **no**)	connects to AC only when outbound traffic is generated. If selected, then route with gateway address from 10.112.112.0/24 network will be added while connection is not established.

interface (*string*; Default:)	interface name on which client will run
keepalive-timeout (*integer*; Default:**60**)	Sets keepalive timeout in seconds.
max-mru (*integer*; Default: **1460**)	Maximum Receive Unit
max-mtu (*integer*; Default: **1460**)	Maximum Transmission Unit
mrru (*integer: 512..65535|disabled*; Default: **disabled**)	maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.
name (*string*; Default: **pppoe-out[i]**)	name of the PPPoE interface, generated by RouterOS if not specified
password (*string*; Default:)	password used to authenticate
profile (*string*; Default: **default**)	Specifies which PPP profile configuration will be used when establishing the tunnel.
service-name (*string*; Default: **""**)	specifies the service name set on the access concentrator, can be left blank to connect to any PPPoE server
use-peer-dns (*yes|no*; Default: **no**)	enable/disable getting DNS settings from the peer
user (*string*; Default: **""**)	username used for authentication

Status

Command `/interface pppoe-client monitor` will display current PPPoE status.

Available read only properties:

Property	Description
ac-mac (*MAC address*)	MAC address of the access concentrator (AC) the client is connected to
ac-name (*string*)	name of the Access Concentrator
active-links (*integer*)	Number of bonded MLPPP connections, ('1' if not using MLPPP)
encoding (*string*)	encryption and encoding (if asymmetric, separated with '/') being used in this connection
local-address (*IP Address*)	IP Address allocated to client
remote-address (*IP Address*)	Remote IP Address allocated to server (ie gateway address)
mru (*integer*)	effective MRU of the link
mtu (*integer*)	effective MTU of the link
service-name (*string*)	used service name
status (*string*)	current link status. Available values are: • dialing, • verifying password..., • connected, • disconnected.
uptime (*time*)	connection time displayed in days, hours, minutes and seconds

Scanner

PPPoE Scanner allows scanning all active PPPoE servers in the layer2 broadcast domain. Command to run scanner is as follows:

```
/interface pppoe-client scan [interface]
```

Available read only properties:

Property	Description
service (*string*)	Service name configured on server
mac-address (*MAC*)	Mac address of detected server
ac-name (*string*)	name of the Access Concentrator

⚠ For Windows, some connection instructions may use the form where the "phone number", such as "MikroTik_AC\mt1", is specified to indicate that "MikroTik_AC" is the access concentrator name and "mt1" is the service name.

⚠ Specifying MRRU means enabling MP (Multilink PPP) over a single link. This protocol is used to split big packets into smaller ones. Under Windows, it can be enabled in the Networking tab, Settings button, "Negotiate multi-link for single link connections". MRRU is hardcoded to 1614 on Windows. This setting is useful to overcome PathMTU discovery failures. The MP setting should be enabled on both peers.

PPPoE Server

There are two types of interface (tunnel) items in PPPoE server configuration - static users and dynamic connections. An interface is created for each tunnel established to the given server. Static interfaces are added administratively if there is a need to reference the particular interface name (in firewall rules or elsewhere) created for the particular user. Dynamic interfaces are added to this list automatically whenever a user is connected and its username does not match any existing static entry (or in case the entry is active already, as there can not be two separate tunnel interfaces referenced by the same name - set *one-session-per-host* value if this is a problem). Dynamic interfaces appear when a user connects and disappear once the user disconnects, so it is impossible to reference the tunnel created for that use in router configuration (for example, in firewall), so if you need a persistent rule for that user, create a static entry for him/her. Otherwise, it is safe to use a dynamic configuration.

⚠ In both cases PPP users must be configured properly - static entries do not replace PPP configuration.

Access concentrator

```
/interface pppoe-server server
```

Properties

Property	Description
authentication (*mschap2* | *mschap1* | *chap* | *pap*; Default: **"mschap2, mschap1, chap, pap"**)	Authentication algorithm
default-profile (*string*; Default: **"default"**)	
interface (*string*; Default: **""**)	Interface that the clients are connected to
keepalive-timeout (*time*; Default: **"10"**)	Defines the time period (in seconds) after which the router is starting to send keepalive packets every second. If there is no traffic and no keepalive responses arrive for that period of time (i.e. 2 * keepalive-timeout), the non responding client is proclaimed disconnected.
max-mru (*integer*; Default: **"1480"**)	Maximum Receive Unit. The optimal value is the MTU of the interface the tunnel is working over reduced by 20 (so, for 1500-byte Ethernet link, set the MTU to 1480 to avoid fragmentation of packets)
max-mtu (*integer*; Default: **"1480"**)	Maximum Transmission Unit. The optimal value is the MTU of the interface the tunnel is working over reduced by 20 (so, for 1500-byte Ethernet link, set the MTU to 1480 to avoid fragmentation of packets)
max-sessions (*integer*; Default: **"0"**)	Maximum number of clients that the AC can serve. '0' = no limitations.

mrru (*integer: 512..65535* | *disabled*; Default: **"disabled"**)	Maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.
one-session-per-host (*yes* | *no*; Default: **"no"**)	Allow only one session per host (determined by MAC address). If a host tries to establish a new session, the old one will be closed.
service-name (*string*; Default: **""**)	The PPPoE service name. Server will accept clients which sends PADI message with service-names that matches this setting or if service-name field in PADI message is not set.

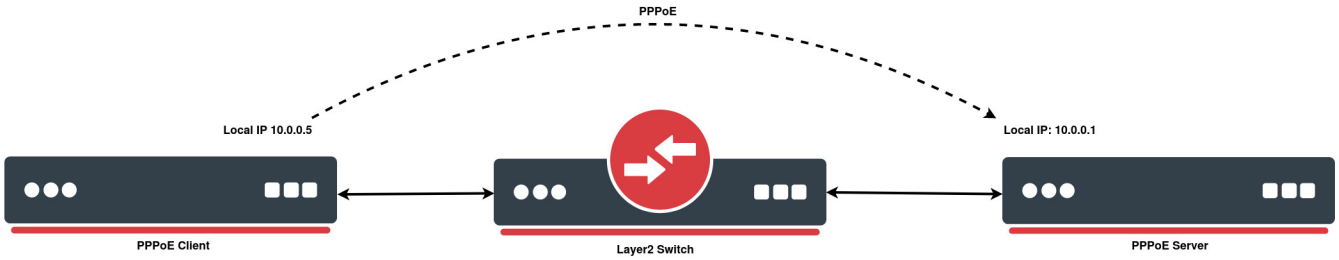
The PPPoE server (access concentrator) supports multiple servers for each interface - with differing service names. The access concentrator name and PPPoE service name are used by clients to identify the access concentrator to register with. The access concentrator name is the same as the identity of the router displayed before the command prompt. The identity may be set within the */system identity* submenu.

Do not assign an IP address to the interface you will be receiving the PPPoE requests on.

Specifying MRRU means enabling MP (Multilink PPP) over a single link. This protocol is used to split big packets into smaller ones. Their MRRU is hardcoded to 1614. This setting is useful to overcome PathMTU discovery failures. The MP setting should be enabled on both peers.

The default *keepalive-timeout* value of 10s is OK in most cases. If you set it to 0, the router will not disconnect clients until they explicitly log out or the router is restarted. To resolve this problem, the one-session-per-host property can be used.

Quick Example



PPPoE Client

To configure MikroTik RouterOS to be a PPPoE client, just add a PPPoE-client with the following parameters as in the example:

```
[admin@MikroTik] > interface pppoe-client add interface=ether2 password=StrongPass service-name=pppoeservice
name=PPPoE-Out disabled=no user=MT-User
[admin@MikroTik] > interface pppoe-client print
Flags: X - disabled, I - invalid, R - running
 0  R name="PPPoE-Out" max-mtu=auto max-mru=auto mrru=disabled interface=ether2 user="MT-User"
      password="StrongPass" profile=default keepalive-timeout=10 service-name="pppoeservice" ac-name=""
      add-default-route=no dial-on-demand=no use-peer-dns=no allow=pap,chap,mschap1,mschap2
```

PPPoE Server

To configure MikroTik RouterOS to be an Access Concentrator (PPPoE Server):

- add an IP address pool for the clients from 10.0.0.2-10.0.0.5;
- add PPP profile;
- add PPP secret (username/password);
- add the PPPoE server itself;

```
[admin@MikroTik] > /ip pool
add name=pppoe-pool ranges=10.0.0.2-10.0.0.5
[admin@MikroTik] > /ppp profile
add local-address=10.0.0.1 name=for-pppoe remote-address=pppoe-pool
[admin@MikroTik] > /ppp secret
add name=MT-User password=StrongPass profile=for-pppoe service=pppoe
[admin@MikroTik] > /interface pppoe-server server
add default-profile=for-pppoe disabled=no interface=ether3 service-name=pppoeservice
```

IPv6 PD over PPP

Summary

This example demonstrates how to set up PPPoE server and client to use IPv6 Prefix Delegation.

IPv6 Prefixes can be delegated over PPP interfaces. When client connects, PPP will automatically add dynamic DHCPv6-PD server. This allows to run DHCPv6 client on PPP interfaces.

Configuration

Server

dhcpv6-pd-pool parameter under PPP Profiles is used to enable PPP-PD. PPP will use specified IPv6 pool to create a dynamic DHCP server.

So first step is to add IPv6 pool:

```
/ipv6 pool
add name=myPool prefix=2001:db8:7501:ff00::/60 prefix-length=62
```

Now we can configure PPP profile and add PPPoE server

```
/ppp profile set default dhcpv6-pd-pool=myPool

/interface pppoe-server server
add service-name=test interface=ether1
```

Client

On client side we need to set up PPPoE client interface and run DHCP client on it.

```
/interface pppoe-client
add name=client-test interface=ether1 user=a1 service-name=test

/ipv6 dhcp-client
add interface=client-test pool-name=ppp-test pool-prefix-length=64
```

Testing status

On server side check if dynamic DHCP server is added and prefix is bound to specific client:

```
[admin@RB1100] /ipv6 dhcp-server> print
Flags: D - dynamic, X - disabled, I - invalid
 #    NAME              INTERFACE            ADDRESS-POOL            LEASE-TIME
 0 D  <pppoe-a1>        <pppoe-a1>           myPool                  3d

[admin@RB1100] /ipv6 dhcp-server binding> print
Flags: X - disabled, D - dynamic
 #   ADDRESS                                      DU       IAID SER.. STATUS
 1 D 2001:db8:7501:ff04::/62                                247 <pp.. bound
```

On client side, check if DHCP client is bound and pool is added:

```
[admin@x86-test] /ipv6 dhcp-client> print
Flags: D - dynamic, X - disabled, I - invalid
 #    INTERFACE           STATUS        PREFIX                          EXPIRES-AFTER
0    client-test          bound         2001:db8:7501:ff04::/62         2d23h18m17s

[admin@x86-test] /ipv6 pool> print
Flags: D - dynamic
 #   NAME                      PREFIX                                  PREFIX-LENGTH
 0 D ppp-test                  2001:db8:7501:ff04::/62                            64
```

MLPPP over single and multiple links

Summary

Standards: `RFC 1990`

Multi-Link Point to Point Protocol (MP, Multi-Link PPP, MultiPPP or MLPPP) is a method of splitting, recombining, and sequencing data across multiple logical data links.

In a situation where we have multiple DSL links a pair of devices, performance by “widening the pipe” between two devices can be increased by using Multi-Link PPP, without going to a newer, more expensive technology.

Large packets are actually split into bits and sent evenly over ALL logical data links. This is done instantaneously with NO loss of bandwidth. It is important to understand that other end of the link needs to use the same protocol to recombine your data.

Multilink is based on an LCP option negotiation that allows to indicate to its peer that it is capable of combining multiple physical links.

MLPPP over single link

Typically size of the packet sent over PPP link is reduced due to overhead. MP can be used to transmit and receive full frame over single ppp link. To make it work the Multilink Protocol uses additional LCP configuration options **Multilink Maximum Received Reconstructed Unit (MRRU)**

To enable Multi-link PPP over single link you must specify MRRU (Maximum Receive Reconstructed Unit) option. If both sides support this feature there are no need for MSS adjustment (in firewall mangle). Study shows that MRRU is less CPU expensive that 2 mangle rules per client. MRRU allows to divide packet to multiple channels therefore increasing possible MTU and MRU (up to 65535 bytes)

Under Windows it can be enabled in Networking tag, Settings button, "Negotiate multi-link for single link connections". Their MRRU is hard coded to 1614.

MTU will be reduced by 4 bytes to work properly when MPPE encryption is enabled

Configuration Example

Let's configure pppoe server compatible with Windows clients and MRRU enabled.

```
[admin@RB800] /interface pppoe-server server> add service-name=myPPP interface=ether1 mrru=1614
[admin@RB800] /interface pppoe-server server> print
Flags: X - disabled
 0   service-name="myPPP" interface=ether1 max-mtu=1480 max-mru=1480 mrru=1614
     authentication=pap,chap,mschap1,mschap2 keepalive-timeout=10 one-session-per-host=no
     max-sessions=0 default-profile=default
```

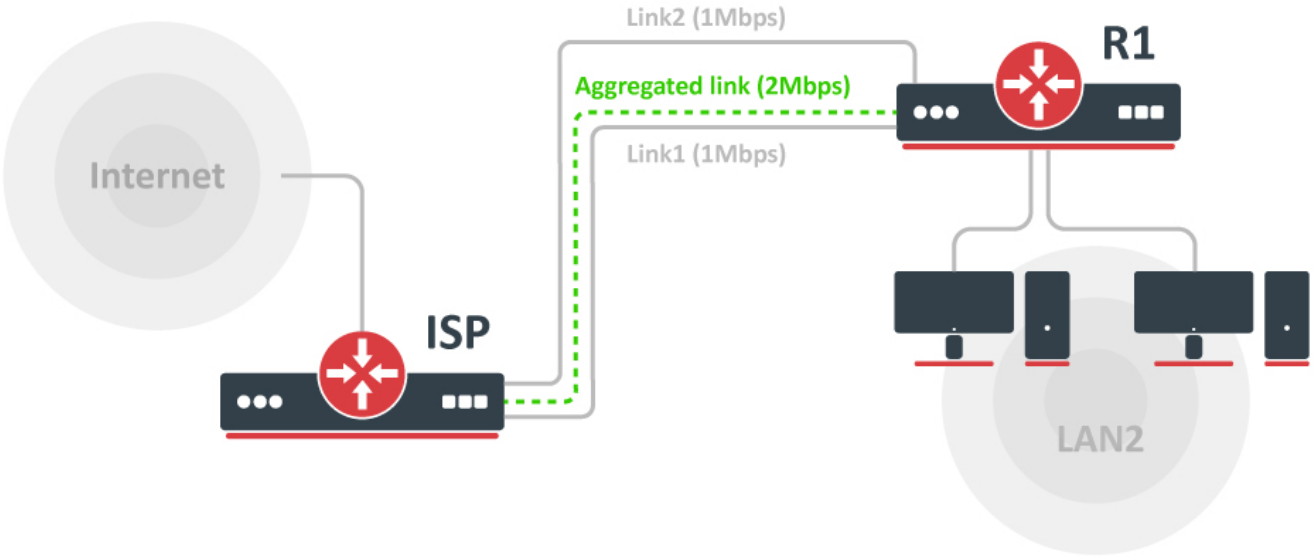
In short - standard PPP link - just specify MRRU in both sides.

MLPPP over multiple links

MLPPP over multiple links allow to create a single PPP link over multiple physical connections. All PPP links must come from the same server (server must have MLPPP over multiple links support) and all PPP links must have same user name and password.

And to enable MLPPP you just need to create PPP client and specify multiple interfaces instead of single interface. RouterOS has MLPPP client support only. Presently there are no MLPPP server support available.

Configuration Example



ISP gives to its client two physical links (DSL lines) 1Mbps each. To get aggregated 2Mbps pipe we have to set up MLPPP. Consider ISP router is pre-configured to support MLPPP.

Configuration on router (R1) is:

```
/interface pppoe-client
   add service-name=ISP interface=ether1,ether2 user=xxx password=yyy disabled=no \
   add-default-route=yes use-peer-dns=yes
```

```
[admin@RB800] /interface pppoe-client> print
Flags: X - disabled, R - running
 0    name="pppoe-out1" max-mtu=1480 max-mru=1480 mrru=disabled interface=ether1,ether2
      user="xxx" password="yyy" profile=default service-name="ISP" ac-name="" add-default-route=yes
      dial-on-demand=no use-peer-dns=yes allow=pap,chap,mschap1,mschap2
```

Now when PPPoE client is connected we can set up rest of configuration, local network address, enable DNS requests, set up masquerade and firewall

```
/ip address add address=192.168.88.1/24 interface=local

/ip dns set allow-remote-request=yes

/ip firewall nat
add chain=src-nat action=masquerade out-interface=pppoe-out1

/ip firewall filter
add chain=input connection-state=invalid action=drop \
        comment="Drop Invalid connections"
add chain=input connection-state=established action=accept \
        comment="Allow Established connections"
add chain=input protocol=icmp action=accept \
        comment="Allow ICMP"
add chain=input src-address=192.168.88.0/24 action=accept \
        in-interface=!pppoe-out1
add chain=input action=drop comment="Drop everything else"
```

For more advanced router and customer protection check firewall examples.

PPTP

Overview

PPTP has many known security issues and we do not recommend using it. However, this protocol is integrated into common operating systems, and it is easy to set it up. PPTP can be useful in networks where security is not of concern.

PPTP traffic uses TCP port 1723 and IP protocol GRE (Generic Routing Encapsulation, IP protocol ID 47), as assigned by the Internet Assigned Numbers Authority (IANA). PPTP can be used with most firewalls and routers by enabling traffic destined for TCP port 1723 and protocol 47 traffic to be routed through the firewall or router. PPTP includes PPP authentication and accounting for each PPTP connection. Full authentication and accounting of each connection may be done through a RADIUS client or locally.

PPTP Client

Properties

Property	Description
add-default-route (*yes* | *no*; Default: **no**)	Whether to add PPTP remote address as a default route.
allow (*mschap2* | *mschap1* | *chap* | *pap*; Default: **mschap2, mschap1, chap, pap**)	Allowed authentication methods.
connect-to (*IP*; Default:)	Remote address of PPTP server
default-route-distance (*byte [0..255]*; Default: **1**)	sets distance value applied to auto created default route, if add-default-route is also selected
dial-on-demand (*yes* | *no*; Default: **no**)	connects to PPTP server only when outbound traffic is generated. If selected, then route with gateway address from 10.112.112.0/24 network will be added while connection is not established.
disabled (*yes* | *no*; Default: **yes**)	Whether interface is disabled or not. By default it is disabled
keepalive-timeout (*integer*; Default: **60**)	Sets keepalive timeout in seconds.
max-mru (*integer*; Default: **1450**)	Maximum Receive Unit. Max packet size that PPTP interface will be able to receive without packet fragmentation.
max-mtu (*integer*; Default: **1450**)	Maximum Transmission Unit. Max packet size that PPTP interface will be able to send without packet fragmentation.
mrru (*disabled* | *integer*; Default: **disabled**)	Maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.
name (*string*; Default:)	Descriptive name of the interface.
password (*string*; Default: **""**)	Password used for authentication.
profile (*name*; Default: **default-encryption**)	
user (*string*; Default:)	User name used for authentication.

PPTP Server

```
/interface pptp-server
```

An interface is created for each tunnel established to the given server. There are two types of interfaces in the PPTP server's configuration:

- Static interfaces are added administratively if there is a need to reference the particular interface name (in firewall rules or elsewhere) created for the particular user;

- Dynamic interfaces are added to this list automatically whenever a user is connected and its username does not match any existing static entry (or in case the entry is active already, as there can not be two separate tunnel interfaces referenced by the same name);

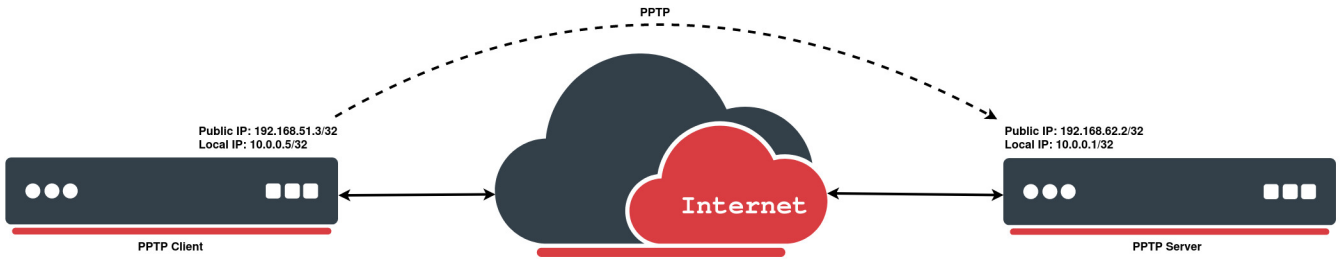
Dynamic interfaces appear when a user connects and disappear once the user disconnects, so it is impossible to reference the tunnel created for that use in router configuration (for example, in firewall), so if you need persistent rules for that user, create a static entry for him/her. Otherwise, it is safe to use a dynamic configuration.

> ⚠ In both cases PPP users must be configured properly - static entries do not replace PPP configuration.

Properties

Property	Description
authentication (*pap* | *chap* | *mschap1* | *mschap2*; Default: **mschap1,mschap2**)	Authentication methods that server will accept.
default-profile (*name*; Default: **default-encryption**)	
enabled (*yes* | *no*; Default: **no**)	Defines whether PPTP server is enabled or not.
keepalive-timeout (*time*; Default: **30**)	If server during keepalive period does not receive any packet, it will send keepalive packets every second five times. If the server does not receives response from the client, then disconnect after 5 seconds. Logs will show 5x "LCP missed echo reply" messages and then disconnect.
max-mru (*integer*; Default: **1450**)	Maximum Receive Unit. Max packet size that PPTP interface will be able to receive without packet fragmentation.
max-mtu (*integer*; Default: **1450**)	Maximum Transmission Unit. Max packet size that PPTP interface will be able to send without packet fragmentation.
mrru (*disabled* | *integer*; Default: **disabled**)	Maximum packet size that can be received on the link. If a packet is bigger than tunnel MTU, it will be split into multiple packets, allowing full size IP or Ethernet packets to be sent over the tunnel.

Example



PPTP Client

The following example demonstrates how to set up a PPTP client with username "MT-User", password "StrongPass" and server 192.168.62.2:

```
[admin@MikroTik] > interface pptp-client add connect-to=192.168.62.2 disabled=no name=pptp-out1
password=StrongPass user=MT-User
[admin@MikroTik] > interface pptp-client print
Flags: X - disabled; R - running
 0  R name="pptp-out1" max-mtu=1450 max-mru=1450 mrru=disabled connect-to=192.168.62.2 user="MT-User"
      password="StrongPass" profile=default-encryption keepalive-timeout=60 add-default-route=no
      dial-on-demand=no allow=pap,chap,mschap1,mschap2
```

PPTP Server

On the other side we simply enable the PPTP server and create a PPP secret for a particular user:

```
[admin@MikroTik] >  interface pptp-server server set enabled=yes
[admin@MikroTik] >  ppp secret add local-address=10.0.0.1 name=MT-User password=StrongPass profile=default-
encryption remote-address=10.0.0.5 service=pptp
[admin@MikroTik] >  interface pptp-server print
Flags: D - dynamic; R - running
Columns: NAME, USER, MTU, CLIENT-ADDRESS, UPTIME, ENCODING
#      NAME            USER     MTU  CLIENT-ADDRESS  UPTIM  ENCODING
0  DR  <pptp-MT-User>  MT-User  1450  192.168.51.3   44m8s  MPPE128 stateless
```

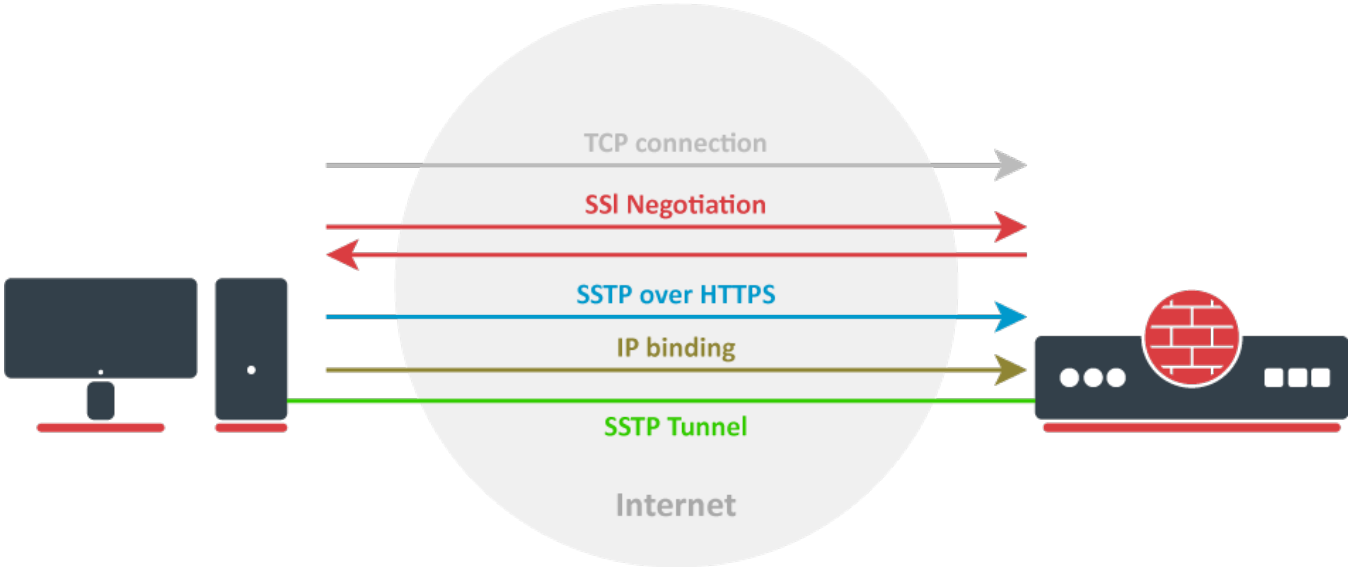
SSTP

Overview

Secure Socket Tunneling Protocol (SSTP) transports a PPP tunnel over a TLS channel. The use of TLS over TCP port 443 allows SSTP to pass through virtually all firewalls and proxy servers. Support IPv4, IPv6.

Introduction

Let`s take a look at the SSTP connection mechanism:



1. A TCP connection is established from client to server (by default on port 443);
2. SSL validates the server certificate. If a certificate is valid, a connection is established otherwise the connection is turned down. (But see note below);
3. The client sends SSTP control packets within the HTTPS session which establishes the SSTP state machine on both sides;
4. PPP negotiation over SSTP. The client authenticates to the server and binds IP addresses to the SSTP interface;

SSTP tunnel is now established and packet encapsulation can begin;

> ⚠ Starting from v5.0beta2 SSTP does not require certificates to operate and can use any available authentication type. This feature will work only between two MikroTik routers, as it is not in accordance with Microsoft standards. Otherwise to establish secure tunnels **mschap** authentication and client/server certificates from the same chain should be used.

Certificates

To set up a secure SSTP tunnel, certificates are required. On the server, authentication is done only by *username* and *password,* but on the client - the server is authenticated using a server certificate. It is also used by the client to cryptographically bind SSL and PPP authentication, meaning - the clients send a special value over SSTP connection to the server, this value is derived from the key data that is generated during PPP authentication and server certificate, this allows the server to check if both channels are secure.

If SSTP clients are on Windows PCs then the only way to set up a secure SSTP tunnel when using a self-signed certificate is by importing the "server" certificate on the SSTP server and on the Windows PC adding a CA certificate in the trusted root.

> ⚠ If your server certificate is issued by a CA which is already known by Windows, then the Windows client will work without any additional certificate imports to a trusted root.

> ⛔ RSA key length must be at least 472 bits if a certificate is used by SSTP. Shorter keys are considered as security threats.

A similar configuration on RouterOS client would be to import the CA certificate and enabling the verify-server-certificate option. In this scenario, Man-in-the-Middle attacks are not possible.

Between two Mikrotik routers, it is also possible to set up an insecure tunnel by not using certificates at all. In this case, data going through the SSTP tunnel is using anonymous DH and Man-in-the-Middle attacks are easily accomplished. This scenario is not compatible with Windows clients.

It is also possible to make a secure SSTP tunnel by adding additional authorization with a client certificate. Configuration requirements are:

- certificates on both server and client
- verification options enabled on server and client

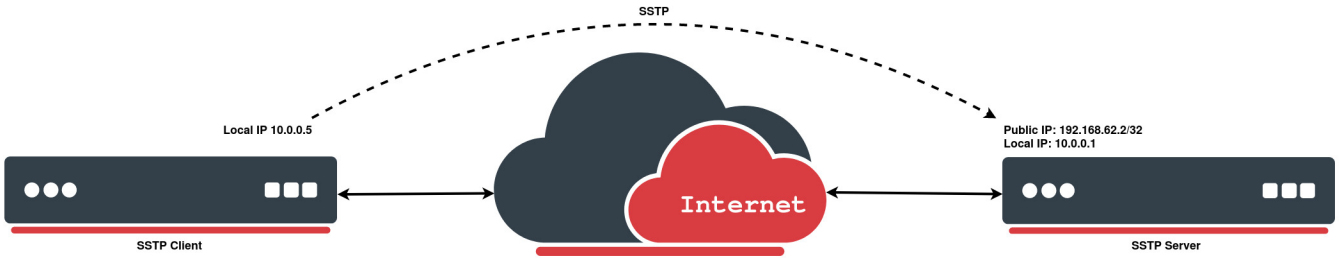
This scenario is also not possible with Windows clients, because there is no way to set up a client certificate on Windows.

Certificate Error Messages

When SSL handshake fails, you will see one of the following certificate errors:

- **certificate is not yet valid** - notBefore certificate date is after the current time;
- **certificate has expired** - certificate expiry date is before the current time;
- **cinvalid certificate purpose** - the supplied certificate cannot be used for the specified purpose;
- **cself signed certificate in a chain** - the certificate chain could be built up using the untrusted certificates but the root could not be found locally;
- **cunable to get issuer certificate locally** - CA certificate is not imported locally;
- **cserver's IP address does not match certificate** - server address verification is enabled, but the address provided in certificate does not match the server's address;

Quick Example



SSTP Client

In the following configuration example, e will create a simple SSTP client without using a certificate:

```
[admin@MikroTik > interface sstp-client add connect-to=192.168.62.2 disabled=no name=sstp-out1
password=StrongPass profile=default-encryption user=MT-User
[admin@MikroTik > interface sstp-client print
Flags: X - disabled; R - running
 0  R name="sstp-out1" max-mtu=1500 max-mru=1500 mrru=disabled connect-to=192.168.62.2:443
      http-proxy=0.0.0.0:443 certificate=none verify-server-certificate=no
      verify-server-address-from-certificate=yes user="MT-User" password="StrongPass"
      profile=default-encryption keepalive-timeout=60 add-default-route=no dial-on-demand=no
      authentication=pap,chap,mschap1,mschap2 pfs=no tls-version=any
```

SSTP Server

We will configure PPP secret for a particular user, afterwards simply enable an SSTP server:

```
[admin@MikroTik] > ppp secret add local-address=10.0.0.1 name=MT-User password=StrongPass remote-address=10.
0.0.5 service=sstp
[admin@MikroTik] > interface sstp-server server set default-profile=default-encryption enabled=yes
[admin@MikroTik] > interface sstp-server server print
                    enabled: yes
                       port: 443
                    max-mtu: 1500
                    max-mru: 1500
                       mrru: disabled
          keepalive-timeout: 60
            default-profile: default-encryption
             authentication: pap,chap,mschap1,mschap2
                certificate: none
  verify-client-certificate: no
                        pfs: no
                tls-version: any
```

WireGuard

Introduction

WireGuard® is an extremely simple yet fast and modern VPN that utilizes state-of-the-art cryptography. It aims to be faster, simpler, leaner, and more useful than IPsec while avoiding massive headaches. It intends to be considerably more performant than OpenVPN. WireGuard is designed as a general-purpose VPN for running on embedded interfaces and super computers alike, fit for many different circumstances. Initially released for the Linux kernel, it is now cross-platform (Windows, macOS, BSD, iOS, Android) and widely deployable.

Properties

Property	Description
comment (*string*; Default:)	Short description of the tunnel.
disabled (*yes* | *no*; Default: **no**)	Enables/disables the tunnel.
listen-port (*integer; Default: 13231*)	Port for WireGuard service to listen on for incoming sessions.
mtu (*integer [0..65536]*; Default: **1420**)	Layer3 Maximum transmission unit.
name (*string*; Default:)	Name of the tunnel.
private-key (*string*; Default:)	A base64 private key. If not specified, it will be automatically generated upon interface creation. Each network interface has a private key and a list of peers.

Read-only properties

Property	Description
public-key (*string*)	A base64 public key is calculated from the private key. Each peer has a public key. Public keys are used by peers to authenticate each other. They can be passed around for use in configuration files.
running (*yes* | *no*)	Whether the interface is running.

Peers

Property	Description

allowed-address (*IP/IPv6 prefix*; *Default*:)	List of IP (v4 or v6) addresses with CIDR masks from which incoming traffic for this peer is allowed and to which outgoing traffic for this peer is directed. This IP address has to be in the same subnet as WireGuard interface set on ROS. If WireGuard interface is at 192.168.99.1/24, You have to input 192.168.99.2 to the client. By adding this IP under 'Allowed Address', you are saying that only this specific client (phone for example) is permitted to connect to this peer configuration. Allowed-address range cannot overlap on one interface, so you need to set own range for each peer.
comment *(string; Default:)*	Short description of the peer.
disabled *(yes | no; Default:* ***no*** *)*	Enables/disables the peer.
endpoint-address *(IP /Hostname; Default:)*	The IP address or hostname. It is used by WireGuard to establish a secure connection between two peers.
endpoint-port (*integer:0.. 65535; Default:*)	The Endpoint port is the UDP port on which a WireGuard peer listens for incoming traffic.
interface (*string; Default:*)	Name of the WireGuard interface the peer belongs to.
persistent-keepalive (*integer:0.. 65535; Default: 0*)	A seconds interval, between 1 and 65535 inclusive, of how often to send an authenticated empty packet to the peer for the purpose of keeping a stateful firewall or NAT mapping valid persistently. For example, if the interface very rarely sends traffic, but it might at anytime receive traffic from a peer, and it is behind NAT, the interface might benefit from having a persistent keepalive interval of 25 seconds.
preshared-key (*string; Default:*)	A base64 preshared key. Optional, and may be omitted. This option adds an additional layer of symmetric-key cryptography to be mixed into the already existing public-key cryptography, for post-quantum resistance. Also can be generated automatically or entered manually, when the key is provided by the system administrator.
private-key *(auto /none; Default:* ***none****)*	A base64 private key.
public-key (*string; Default:*)	A base64 public key is calculated from the private key. Each peer has a public key. Public keys are used by peers to authenticate each other. They can be passed around for use in configuration files.
show-client-config	*Will show already created Peer configuration and generate a QR code for easier peer setup on a client device. Does not affect the WireGuard Server.*
Used for the client-server setup scenario, when the configuration is imported using a qr code for a client, configuration details on tab with qrcode will appear once it has been set in the fields:	
client-address *(IP/IPv6 prefix; Default:)*	When imported using a qr code for a client (for example, a phone), then this address for the wg interface is set on that device.

client-dns *(IP/IPv6 prefix; Default:)*	Specify when using WireGuard Server as a VPN gateway for peer traffic.
client-endpoint *(IP/IPv6 prefix; Default:)*	The IP address and port number of the WireGuard Server.
client-keepalive (*integer:0..65535; Default: 0*)	Same as **persistent-keepalive** but from peer side.
client-listen-port (*integer:0..65535; Default:*)	The local port upon which this WireGuard tunnel will listen for incoming traffic from peers, and the port from which it will source outgoing packets.
name (*string; Default:*)	Allows adding name to a peer. Name will be used as a reference for a peer in WireGuard logs. (Available from RouterOS version 7.15)
is-responder *(yes | no; Default:* ***no*** *)*	Specifies if peer is intended to be connection initiator or only responder. Should be used on WireGuard devices that are used as "servers" for other devices as clients to connect to. Otherwise router will all repeatedly try to connect "endpoint-address" or "current-endpoint-address" causing unnecessary system logs to be written.

***AllowedIPs** configuration that is provided to the client through WireGuard peer export (configuration file or QR code) can not be changed and will be "0.0.0.0/0, ::/0" at the moment. If it is necessary to change these values on remote end, then that is up to the remote peer software used for WireGuard connection.

Minimum parameters must be specified for importing on the client device by QR-code/file.

Example:

```
interface: wireguard1
public-key: v/oIzPyFm1FPHrqhytZgsKjU7mUToQHLrW+Tb5e601M=
private-key: KMwxqe/iXAU8Jn9dd1o5pPdHep2blGxNWm9I944/I24=
allowed-address: 192.168.88.3/24
client-address: 192.168.88.3/32
client-endpoint: example.com:13231
```

When using interface/wireguard/wg-import file=, you may get Could not parse error, if Wireguard import file starts with #, use it clean as per example:

```
[Interface]
Address =192.168.88.3/24
ListenPort = 13533
PrivateKey = UBLqJEFZZf9wszZSUF2BPWa9dsMX99RbEcxlNfxWffk=
```

Read-only properties

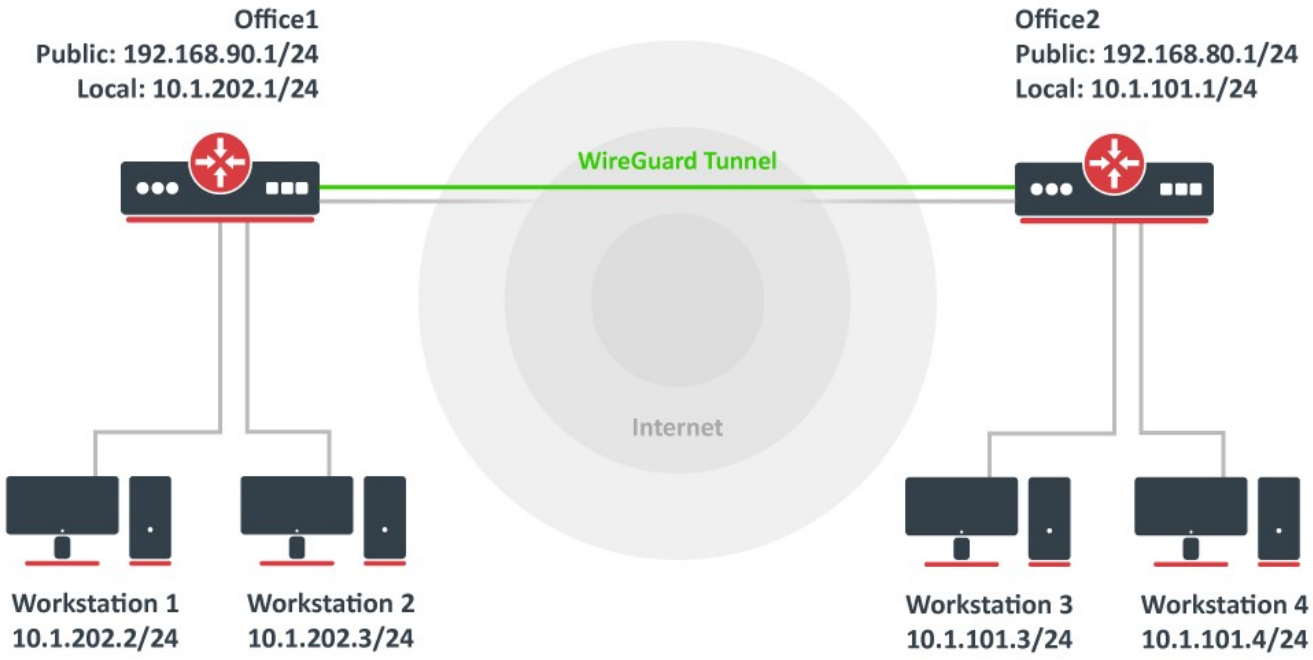
Property	Description
current-endpoint-address (*IP/IPv6*)	The most recent source IP address of correctly authenticated packets from the peer.

current-endpoint-port (*integer*)	The most recent source IP port of correctly authenticated packets from the peer.
last-handshake (*integer*)	Time in seconds after the last successful handshake.
rx (*integer*)	The total amount of bytes received from the peer.
tx (*integer*)	The total amount of bytes transmitted to the peer.

Application examples

Site to Site WireGuard tunnel

Consider setup as illustrated below. Two remote office routers are connected to the internet and office workstations are behind NAT. Each office has its own local subnet, 10.1.202.0/24 for Office1 and 10.1.101.0/24 for Office2. Both remote offices need secure tunnels to local networks behind routers.



WireGuard interface configuration

First of all, WireGuard interfaces must be configured on both sites to allow automatic private and public key generation. The command is the same for both routers:

```
/interface/wireguard
add listen-port=13231 name=wireguard1
```

Now when printing the interface details, both private and public keys should be visible to allow an exchange.

> ⚠ Any private key will never be needed on the remote side device - hence the name private.

Office1

```
/interface/wireguard print
Flags: X - disabled; R - running
 0  R name="wireguard1" mtu=1420 listen-port=13231 private-key="yKt9NJ4e5qlaSgh48WnPCDCEkDmq+VsBTt/DDEBWfEo="
      public-key="u7gYAg5tkioJDcm3hyS7pm79eADKPs/ZUGON6/fF3iI="
```

Office2

```
/interface/wireguard/print
Flags: X - disabled; R - running
 0  R name="wireguard1" mtu=1420 listen-port=13231 private-key="KMwxqe/iXAU8Jn9dd1o5pPdHep2blGxNWm9I944/I24="
      public-key="v/oIzPyFm1FPHrqhytZgsKjU7mUToQHLrW+Tb5e601M="
```

Peer configuration

Peer configuration defines who can use the WireGuard interface and what kind of traffic can be sent over it. To identify the remote peer, its public key must be specified together with the created WireGuard interface.

Office1

```
/interface/wireguard/peers
add allowed-address=10.1.101.0/24 endpoint-address=192.168.80.1 endpoint-port=13231 interface=wireguard1 \
public-key="v/oIzPyFm1FPHrqhytZgsKjU7mUToQHLrW+Tb5e601M="
```

Office2

```
/interface/wireguard/peers
add allowed-address=10.1.202.0/24 endpoint-address=192.168.90.1 endpoint-port=13231 interface=wireguard1 \
public-key="u7gYAg5tkioJDcm3hyS7pm79eADKPs/ZUGON6/fF3iI="
```

IP and routing configuration

Lastly, IP and routing information must be configured to allow traffic to be sent over the tunnel.

Office1

```
/ip/address
add address=10.255.255.1/30 interface=wireguard1
/ip/route
add dst-address=10.1.101.0/24 gateway=wireguard1
```

Office2

```
/ip/address
add address=10.255.255.2/30 interface=wireguard1
/ip/route
add dst-address=10.1.202.0/24 gateway=wireguard1
```

Firewall considerations

The default RouterOS firewall will block the tunnel from establishing properly. The traffic should be accepted in the "input" chain before any drop rules on both sites.

Office1

```
/ip/firewall/filter
add action=accept chain=input dst-port=13231 protocol=udp src-address=192.168.80.1
```

Office2

```
/ip/firewall/filter
add action=accept chain=input dst-port=13231 protocol=udp src-address=192.168.90.1
```

Additionally, it is possible that the "forward" chain restricts the communication between the subnets as well, so such traffic should be accepted before any drop rules as well.

Office1

```
/ip/firewall/filter
add action=accept chain=forward dst-address=10.1.202.0/24 src-address=10.1.101.0/24
add action=accept chain=forward dst-address=10.1.101.0/24 src-address=10.1.202.0/24
```

Office2

```
/ip/firewall/filter
add action=accept chain=forward dst-address=10.1.101.0/24 src-address=10.1.202.0/24
add action=accept chain=forward dst-address=10.1.202.0/24 src-address=10.1.101.0/24
```

RoadWarrior WireGuard tunnel

RouterOS configuration

Add a new WireGuard interface and assign an IP address to it.

```
/interface wireguard
add listen-port=13231 name=wireguard1
/ip address
add address=192.168.100.1/24 interface=wireguard1
```

Adding a new WireGuard interface will automatically generate a pair of private and public keys. You will need to configure the public key on your remote devices. To obtain the public key value, simply print out the interface details.

```
[admin@home] > /interface wireguard print
Flags: X - disabled; R - running
 0  R name="wireguard1" mtu=1420 listen-port=13231 private-key="cBPD6JNvbEQr73gJ7NmwepSrSPK3np381AWGvBk/QkU="
      public-key="VmGMh+cwPdb8//NOhuf1i1VIThypkMQrKAO9Y55ghG8="
```

For the next steps, you will need to figure out the public key of the remote device. Once you have it, add a new peer by specifying the public key of the remote device and allowed addresses that will be allowed over the WireGuard tunnel.

```
/interface wireguard peers
add allowed-address=192.168.100.2/32 interface=wireguard1 public-key="<paste public key from remote device
here>"
```

Firewall considerations

If you have default or strict firewall configured, you need to allow remote device to establish the WireGuard connection to your device.

```
/ip firewall filter
add action=accept chain=input comment="allow WireGuard" dst-port=13231 protocol=udp place-before=1
```

To allow remote devices to connect to the RouterOS services (e.g. request DNS), allow the WireGuard subnet in input chain.

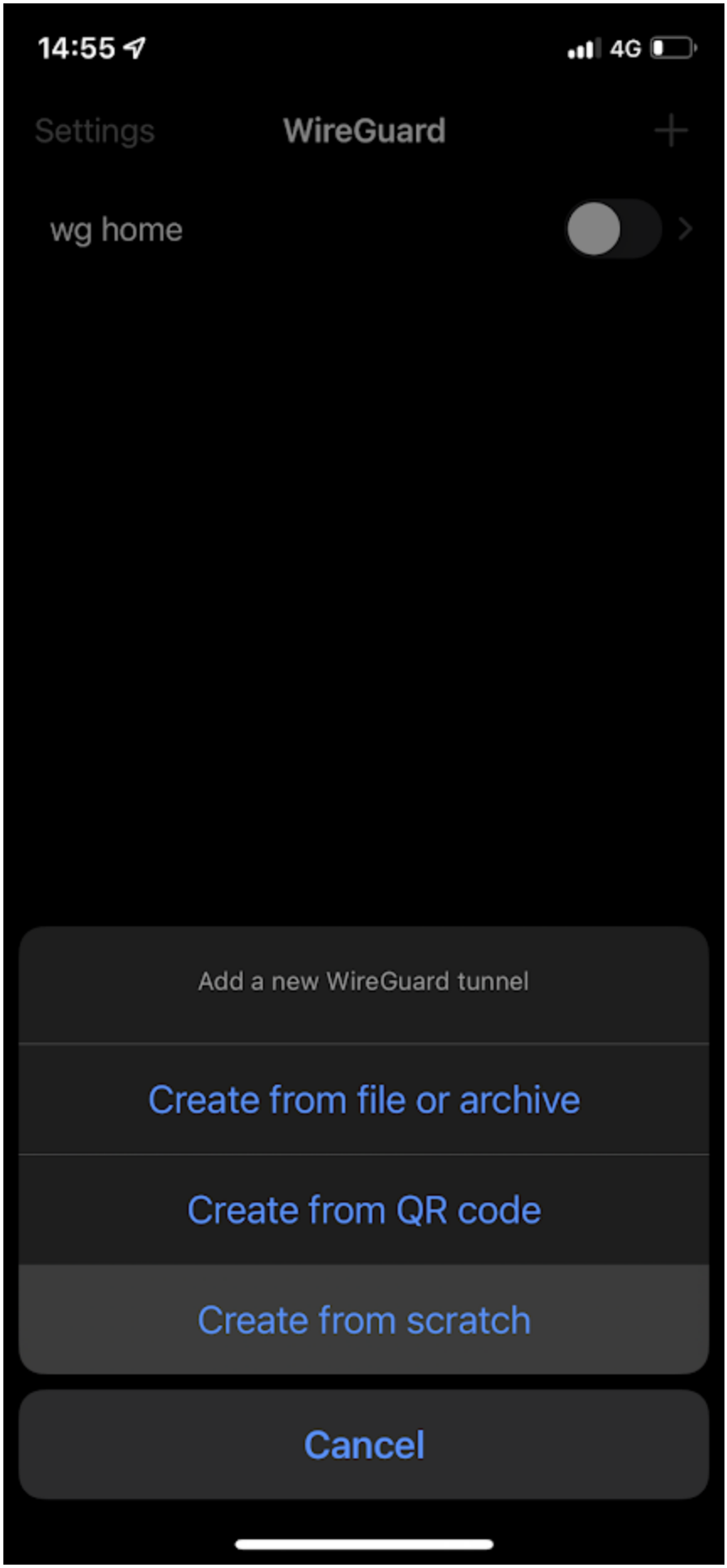
```
/ip firewall filter
add action=accept chain=input comment="allow WireGuard traffic" src-address=192.168.100.0/24 place-before=1
```

Or simply add the WireGuard interface to "LAN" interface list.

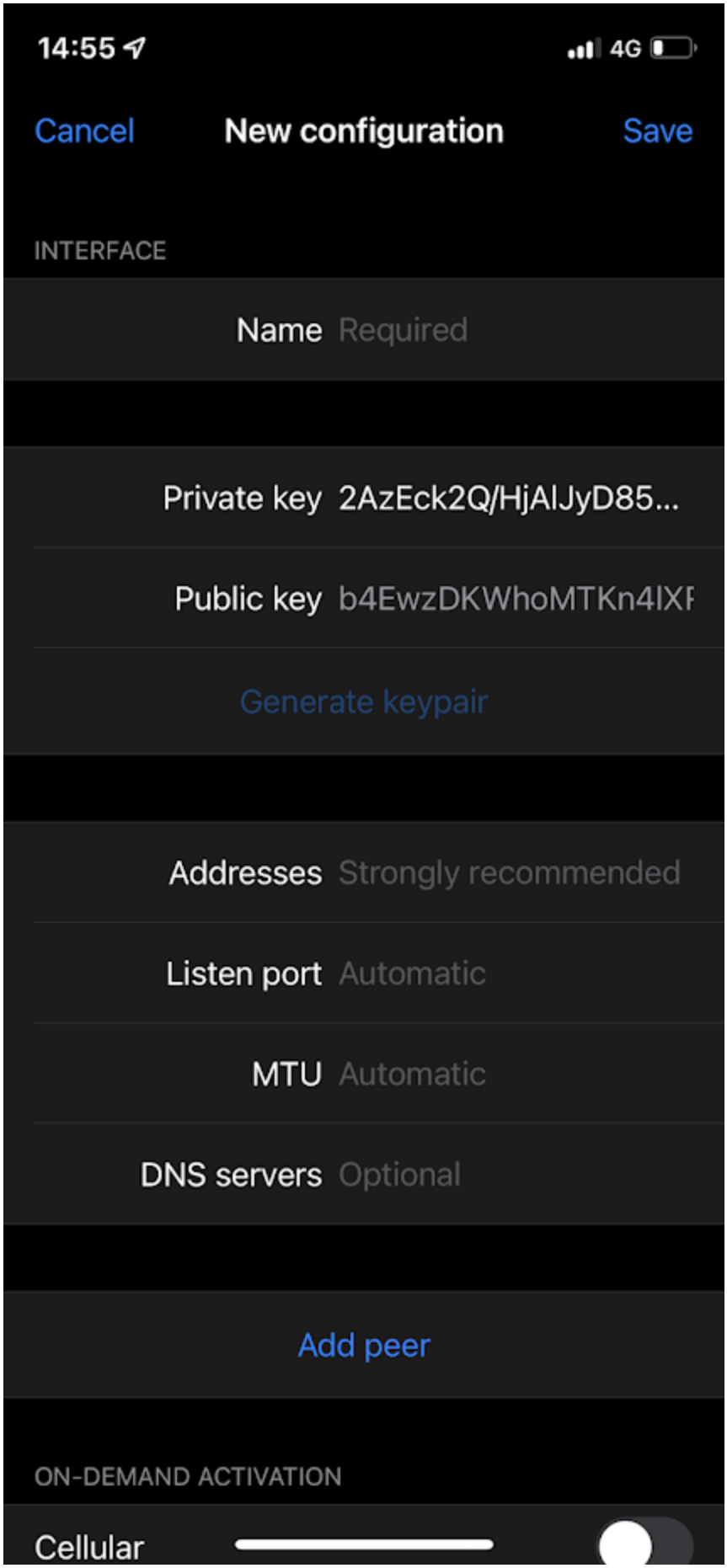
```
/interface list member
add interface=wireguard1 list=LAN
```

iOS configuration

Download the WireGuard application from the App Store. Open it up and create a new configuration from scratch.

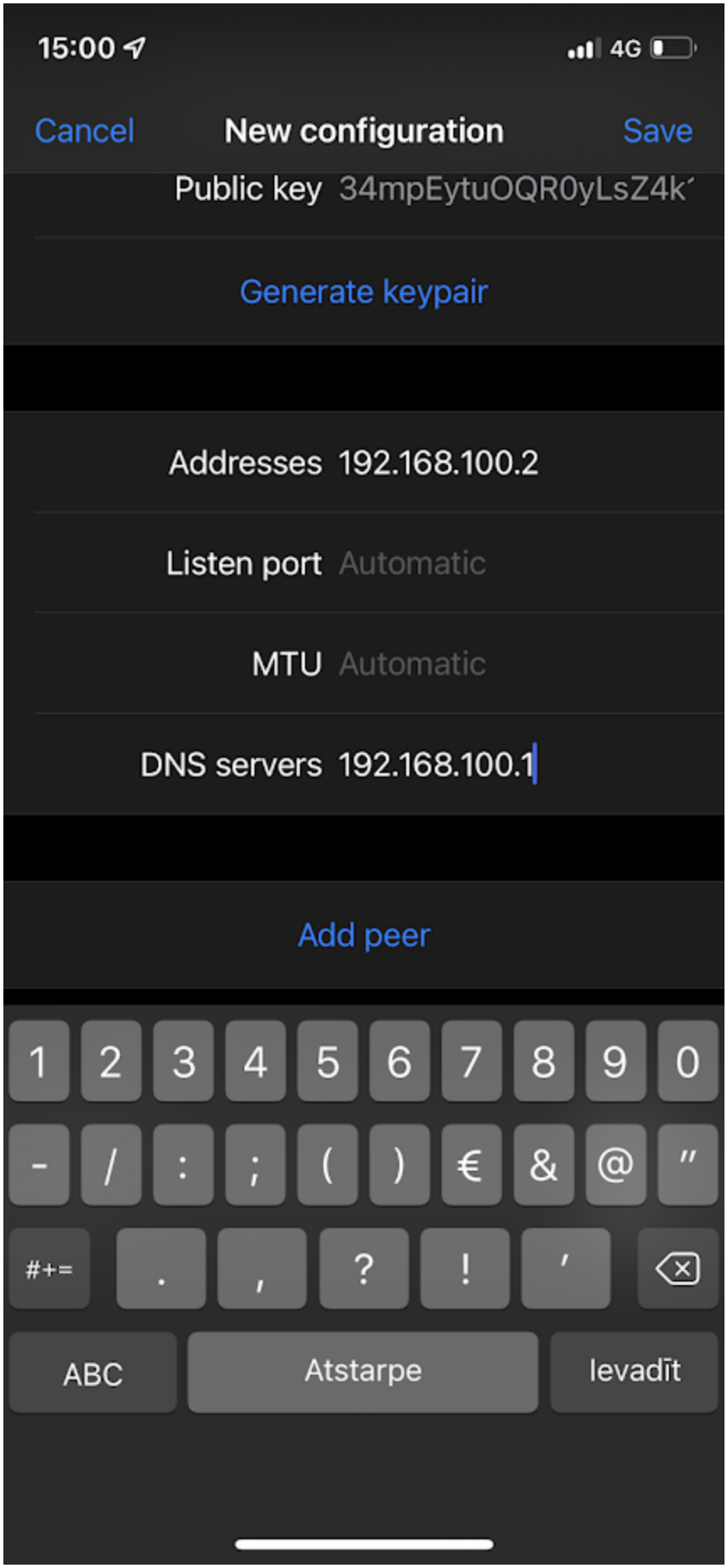

14:55
4G
Settings
WireGuard
wg home
Add a new WireGuard tunnel
Create from file or archive
Create from QR code
Create from scratch
Cancel

First of all give your connection a "Name" and choose to generate a keypair. The generated public key is necessary for peer's configuration on RouterOS side.


14:55
4G
Cancel
New configuration
Save
INTERFACE
Name Required
Private key 2AzEck2Q/HjAlJyD85...
Public key b4EwzDKWhoMTKn4lXF
Generate keypair
Addresses Strongly recommended
Listen port Automatic
MTU Automatic
DNS servers Optional
Add peer
ON-DEMAND ACTIVATION
Cellular

Specify an IP address in "Addresses" field that is in the same subnet as configured on the server side. This address will be used for communication. For this example, we used 192.168.100.1/24 on the RouterOS side, you can use 192.168.100.2 here.

If necessary, configure the DNS servers. If allow-remote-requests is set to yes under IP/DNS section on the RouterOS side, you can specify the remote WireGuard IP address here.

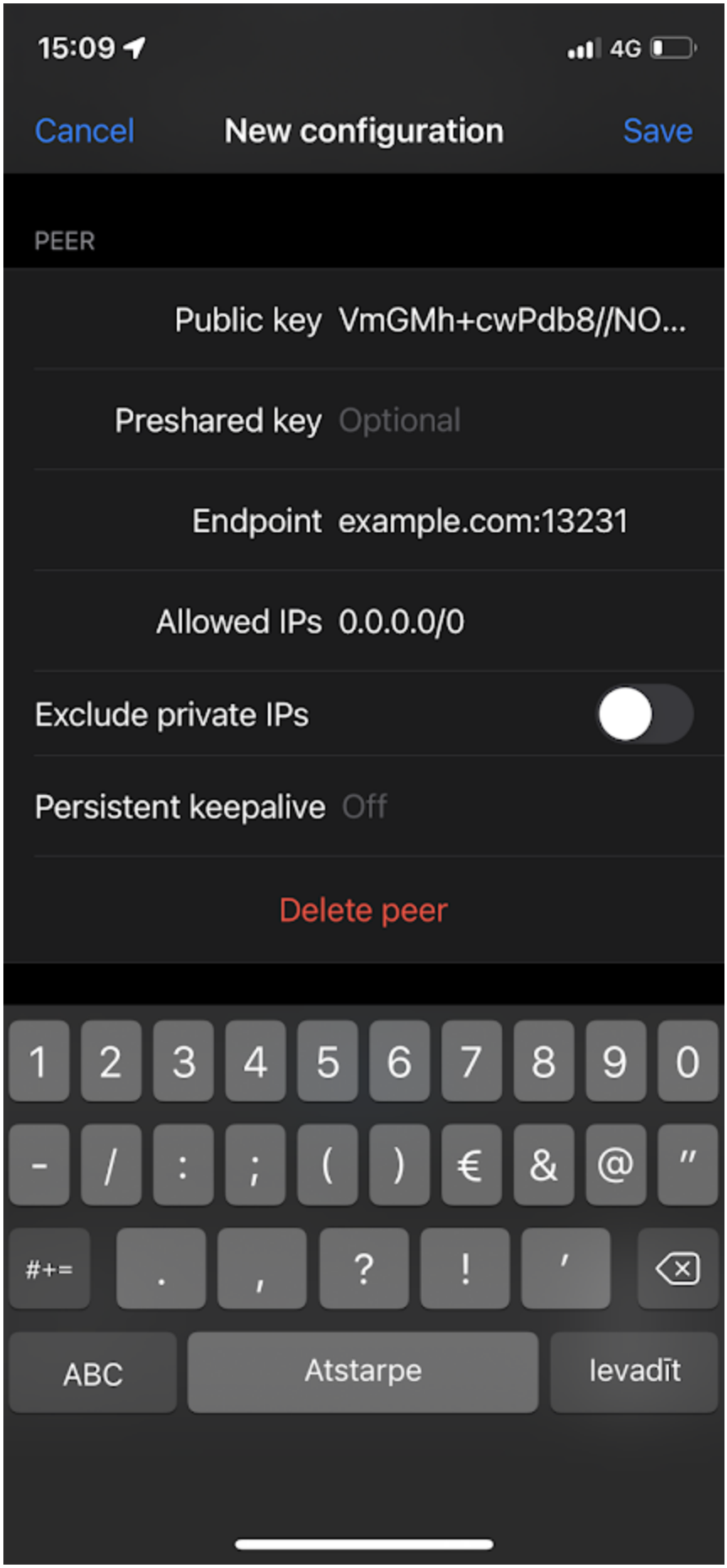

15:00
4G
Cancel
New configuration
Save
Public key 34mpEytuOQR0yLsZ4k
Generate keypair
Addresses 192.168.100.2
Listen port Automatic
MTU Automatic
DNS servers 192.168.100.1
Add peer
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) € & @ "
#+= . , ? ! '
ABC
Atstarpe
Ievadīt

Click "Add peer" which reveals more parameters.

The "Public key" value is the public key value that is generated on the WireGuard interface on RouterOS side.

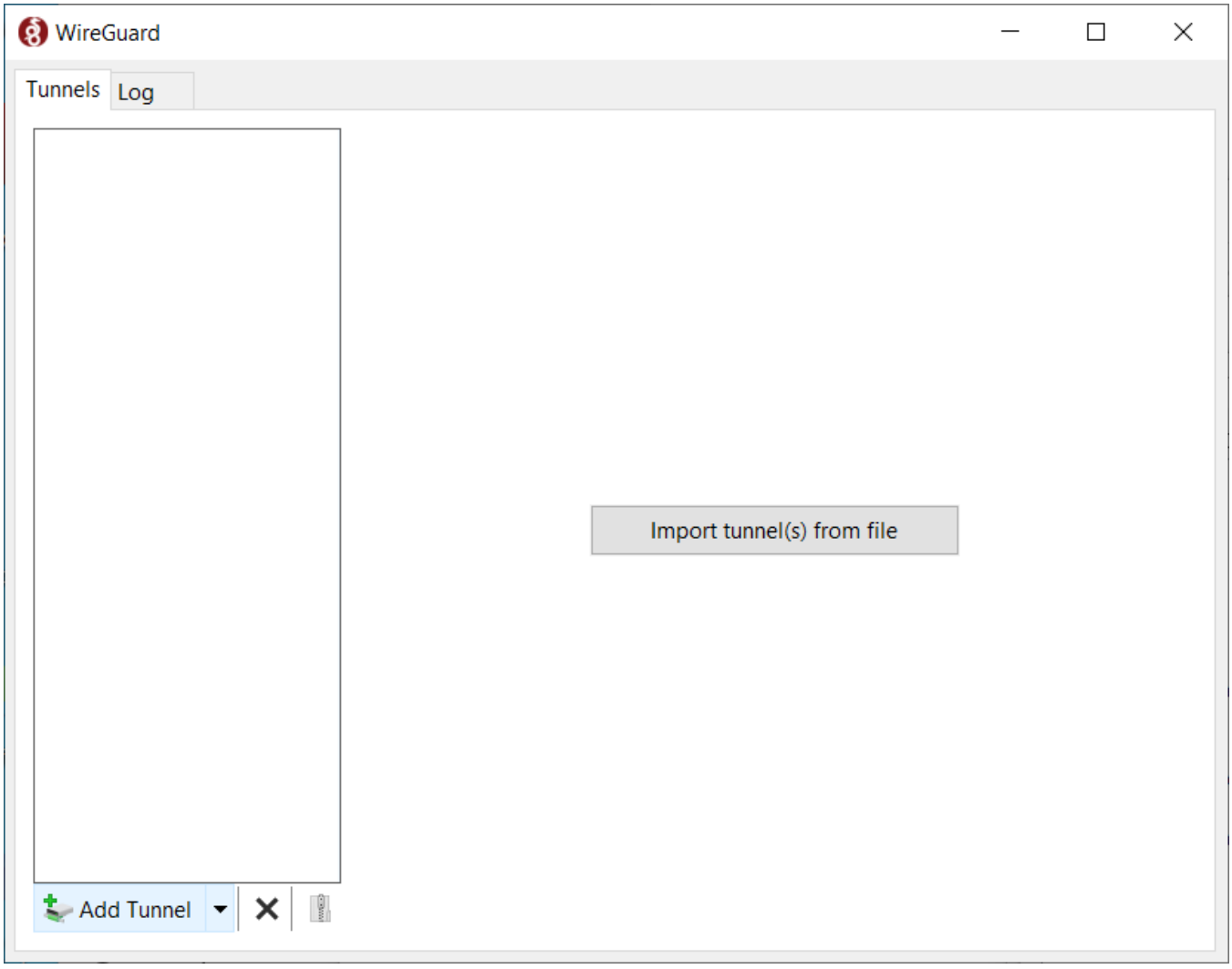
"Endpoint" is the IP or DNS with port number of the RouterOS device that the iOS device can communicate with over the Internet.

"Allowed IPs" are set to 0.0.0.0/0 to allow all traffic to be sent over the WireGuard tunnel.


15:09
4G
Cancel
New configuration
Save
PEER
Public key VmGMh+cwPdb8//NO...
Preshared key Optional
Endpoint example.com:13231
Allowed IPs 0.0.0.0/0
Exclude private IPs
Persistent keepalive Off
Delete peer
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) € & @ "
#+= . , ? ! '
ABC
Atstarpe
Ievadīt

Windows 10 configuration

Download WireGuard installer from Wireguard
Run as Administrator.



Press Ctrl+n to add new empty tunnel, add name for interface, Public key should be auto generated copy it to RouterOS peer configuration.
Add to server configuration, so full configuration looks like this (keep your auto generated PrivateKey in [Interface] section:

```
[Interface]
PrivateKey = your_autogenerated_public_key=
Address = 192.168.100.2/24
DNS = 192.168.100.1

[Peer]
PublicKey = your_MikroTik_public_KEY=
AllowedIPs = 0.0.0.0/0
Endpoint = example.com:13231
```

Save and Activate

ZeroTier

Introduction

The ZeroTier network hypervisor is a self-contained network virtualization engine that implements an Ethernet virtualization layer similar to VXLAN built atop a cryptographically secure global peer-to-peer network. It provides advanced network virtualization and management capabilities on par with an enterprise SDN switch, but across both local and wide area networks and connecting almost any kind of app or device.

MikroTik has added ZeroTier to RouterOS v7.1rc2 as a separate package for the **ARM/ARM64** architecture.

Wait, so what can I use it for?

- Hosting a game server at home (useful for LAN only games) or simply creating a LAN party with your friends;
- Accessing LAN devices behind NAT directly;
- Accessing LAN devices via SSH without opening port to the Internet;
- Using your local Pi-Hole setup from anywhere via the Internet;

Video tutorial

- ZeroTier

Required Network Configuration

What ports does ZeroTier use?

It listens on three 3 UDP ports:

- 9993 - The default
- A random, high numbered port derived from your ZeroTier address
- A random, high numbered port for use with UPnP/NAT-PMP mappings

That means your *peers* could be listening on any port. To talk with them directly, you need to be able to send them to any port.

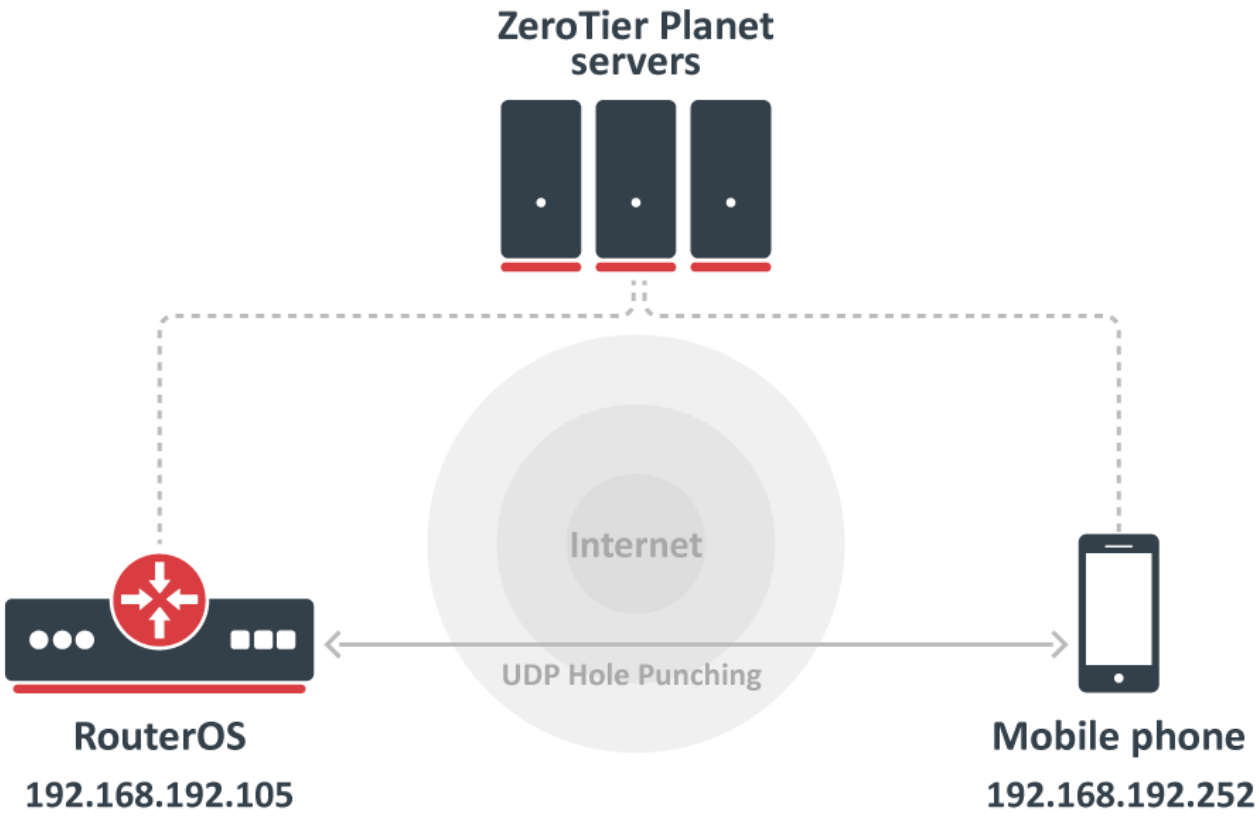
Recommended Local Network and Internet Gateway Configuration

These ZeroTier recommended guidelines are consistent with the vast majority of typical deployments using commodity gateways and access points:

- Don't restrict outbound UDP traffic.
- Supporting either UPnP or NAT-PMP on your network can greatly improve performance by allowing ZeroTier endpoints to map external ports and avoid NAT traversal entirely.
- IPv6 is recommended and can greatly improve direct connection reliability if supported on both ends of a direct link. If present it should be implemented without NAT (NAT is wholly unnecessary with IPv6 and only adds complexity) and with a stateful firewall that permits bidirectional UDP conversations.

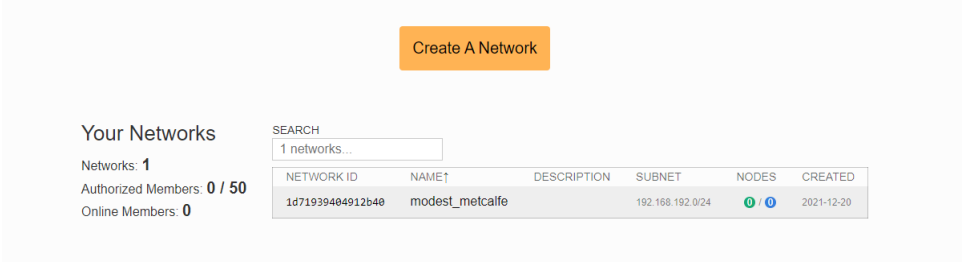
- Don't use "symmetric" NAT. Use "full cone" or "port restricted cone" NAT. Symmetric NAT is extremely hostile to peer-to-peer traffic and will degrade VoIP, video chat, games, WebRTC, and many other protocols as well as ZeroTier.
- No more than one layer of NAT should be present between ZeroTier endpoints and the Internet. Multiple layers of NAT introduce connection instability due to chaotic interactions between states and behaviors at different levels. **No Double NAT.**
- NATs should have a port mapping or connection timeout no shorter than 60 seconds.
- Place no more than about 16,000 devices behind each NAT-managed external IP address to ensure that each device can map a sufficient number of ports.
- Switches and wireless access points should allow direct local traffic between local devices. Turn off any "local isolation" features. Some switches might allow finer-grained control, and on these, it would be sufficient to allow local UDP traffic to/from 9993 (or in general).

Configuration example



By default, ZeroTier is designed to be zero-configuration. A user can start a new ZeroTier node without having to write configuration files or provide the IP addresses of other nodes. It's also designed to be fast. Any two devices in the world should be able to locate each other and communicate almost instantly so the following example will enable ZeroTier on RouterOS device and connect one mobile phone using the ZeroTier application.

1. Register on my.zerotier.com and **Create A Network**, obtain the *Network ID*, in this example: *1d71939404912b40*;



2. Download and Install ZeroTier NPK package in RouterOS, you can find under in the "Extra packages", upload package on the device and reboot the unit;
3. Enable the default (official) ZeroTier instance:

```
[admin@mikrotik] > zerotier/enable zt1
```

4. Add a new network, specifying the network ID you created in the ZeroTier cloud console:

```
[admin@mikrotik] zerotier/interface/add network=1d71939404912b40 instance=zt1
```

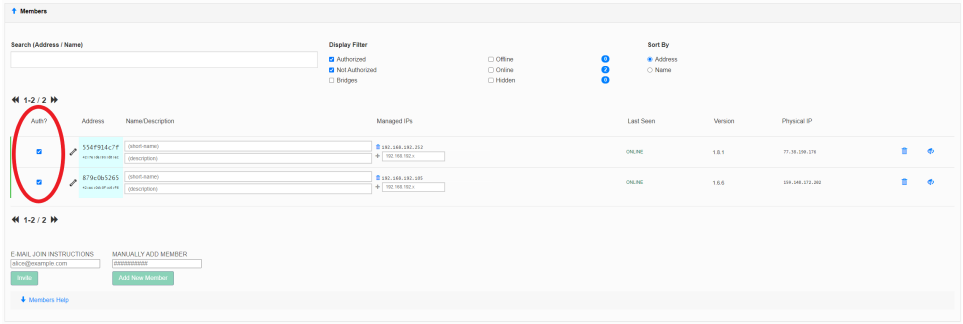
5. Verify ZeroTier configuration:

```
[admin@MikroTik] > zerotier/interface/print
Flags: R - RUNNING
Columns: NAME, MAC-ADDRESS, NETWORK, NETWORK-NAME, STATUS
#   NAME       MAC-ADDRESS        NETWORK           NETWORK-NAME     STATUS
0 R zerotier1  42:AC:0D:0F:C6:F6  1d71939404912b40  modest_metcalfe  OK
```

6. Now you might need to allow connections from the ZeroTier interface to your router, and **optionally**, to your other LAN interfaces:

```
/ip firewall filter add action=accept chain=forward in-interface=zerotier1 place-before=0
/ip firewall filter add action=accept chain=input in-interface=zerotier1 place-before=0
```

7. Install a ZeroTier client on your smartphone or computer, follow the ZeroTier manual on how to connect to the same network from there.
8. If **"Access Control"** is set to **"Private"**, you must authorize nodes before they become members:



9.
```
[admin@MikroTik] > ip/address/print where interface~"zero"
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS             NETWORK        INTERFACE
3 D 192.168.192.105/24  192.168.192.0  zerotier1

[admin@MikroTik] > ping 192.168.192.252 count=3
SEQ HOST                                     SIZE TTL TIME
STATUS

0 192.168.192.252                             56  64 407us
1 192.168.192.252                             56  64 452us
2 192.168.192.252                             56  64 451us
sent=3 received=3 packet-loss=0% min-rtt=407us avg-rtt=436us max-rtt=452us
```

You should specify routes to specific internal subnets in the ZeroTier cloud console, to make sure you can access those networks when connecting from other devices.

Peer

```
zerotier/peer/
```

ZeroTier`s peer is an informative section with a list of nodes that your node knows about. Nodes can not talk to each other unless they are joined and authorized on the same network.

```
[admin@Home] > zerotier/peer/print
Columns: INSTANCE, ZT-ADDRESS, LATENCY, ROLE, PATH
# INSTANCE  ZT-ADDRESS  LATENCY  ROLE    PATH
0 zt1       61d294b9cb  186ms    PLANET  active,preferred,50.7.73.34/9993,recvd:4s526ms
1 zt1       62f865ae71  270ms    PLANET  active,preferred,50.7.252.138/9993,recvd:4s440ms,sent:9s766ms
2 zt1       778cde7190  132ms    PLANET  active,preferred,103.195.103.66/9993,recvd:4s579ms,sent:9s766ms
3 zt1       992fcf1db7  34ms     PLANET  active,preferred,195.181.173.159/9993,recvd:4s675ms,sent:4s712ms
4 zt1       159924d630  130ms    LEAF    active,preferred,34.121.192.xx/21002,recvd:3s990ms,sent:3s990ms
```

Parameters

```
[admin@MikroTik] > zerotier/
```

Property	Description
name (*string*; default: **zt1**)	Instance name.
port (*number;* default: **9993**)	Port number the instance listen to.
identity (*string*; default)	Instance 40-bit unique address.
interface (string; default: **all)**	List of interfaces that are used in order to discover ZeroTier peers, by using ARP and IP type connections.
route-distance (number; default: **1**)	Route distance for routes obtained from planet/moon servers.

```
[admin@MikroTik] > zerotier/interface/
```

Property	Description
allow-default (*string; yes* | *no)*	A network can override the systems default route (force VPN mode).
allow-global (*string; yes* | *no)*	ZeroTier IP addresses and routes can overlap public IP space.
allow-managed (*string; yes* | *no)*	ZeroTier managed IP addresses and routes are assigned.
arp-timeout (*number*; default: **auto**)	ARP timeouts value.
comment (*string*; Default:)	Descriptive comment for the interfaces.
copy-from	Allows copying existing interfaces configuration.
disable-running-check (*string; yes* | *no)*	Force interface in "running" state.
instance (*string*; Default: **zt1**)	ZeroTier instance name.
name (s*tring*; default: **zerotier1**)	A short name.
network (*string*; Default)	16-digit network ID.

Controller

RouterOS implements ZeroTier functionality in the role of a node where most of the network configuration must be done on the ZeroTier webpage dashboard. However, in situations where you would prefer to do all the configuration on your own device, RouterOS offers to host your own controller

A common misunderstanding is to conflate network controllers with root servers (planet and moons). Root servers are connection facilitators that operate at the VL1 level. Network controllers are configuration managers and certificate authorities that belong to the VL2 level. Generally, root servers don't join or control virtual networks and network controllers are not root servers, though it is possible to have a node do both.

```
/zerotier/controller/
```

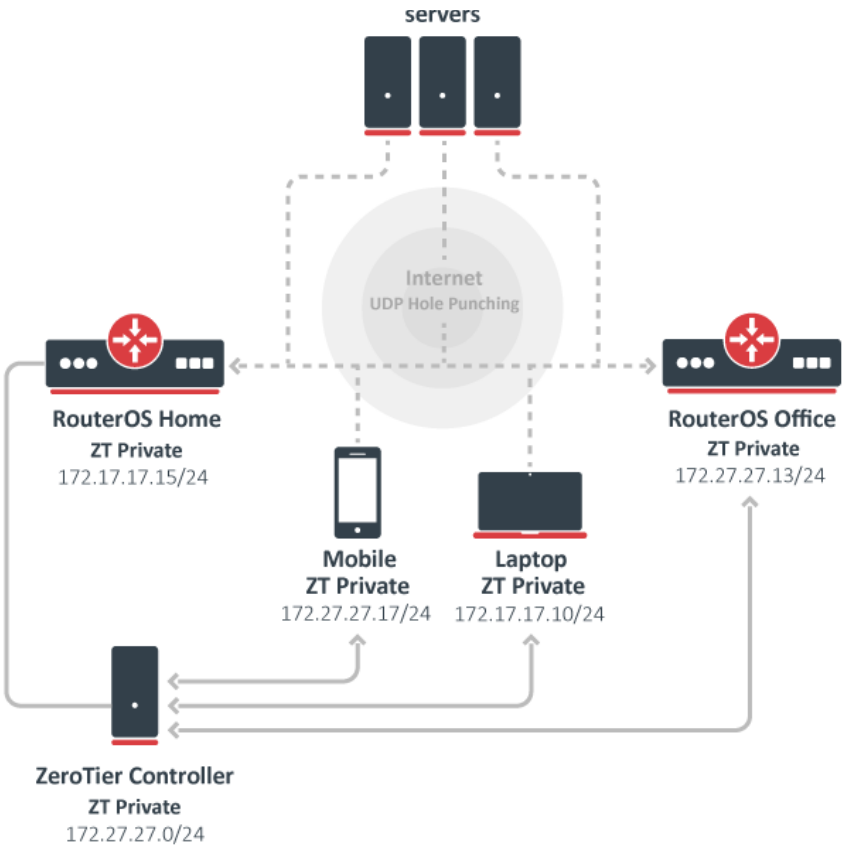
Every ZeroTier instance has a self-hosting network controller that can be used to host virtual networks. A controller is responsible for admitting members to the network, and issuing default configuration information including certificates. Controllers can in theory host up to 2^24 networks and serve many millions of devices (or more), but we recommend spreading large numbers of networks across many controllers for load balancing and fault tolerance reasons.

Parameters

Property	Description
broadcast *(yes | no; Default:* ***yes****)*	Allow receiving broadcast (*FF:FF:FF:FF:FF:FF*) packets.
comment (*string*; Default:)	Descriptive comment for the controller.
copy-from (*string*; Default:)	Copies an existing item. It takes default values of a new item's properties from another item. If you do not want to make an exact copy, you can specify new values for some properties. When copying items that have names, you will usually have to give a new name to a copy.
instance (*string*; Default: **zt1**)	ZeroTier instance name.
ip-range (*IP*; Default:)	IP range, *for example, 172.16.16.1-172.16.16.254*.
ip6-6plane *(yes | no; Default:* ***no****)*	An option gives every member a /80 within a /40 network but uses NDP emulation to route *all* IPs under that /80 to their owner. The `6plane` mode is great for use cases like Docker since it allows every member to assign IPv6 addresses within its /80 that just work instantly and globally across the network.
ip6-rfc4193 *(yes | no; Default:* ***no****)*	The *rfc4193* mode gives every member a /128 on a /88 network.
ip6-range (*IPv6*; Default:)	IPv6 range, *for example fd00:feed:feed:beef::-fd00:feed:feed:beef:ffff:ffff:ffff:ffff*.
mtu *(integer;* Default: **2800**)	Network MTU.
multicast-limit (*integer*: Default: **32**)	Maximum recipients for a multicast packet.
name (*string*; Default:)	A short name for this controller.
network (*string*; Default)	16-digit network ID.
private *(yes | no; Default:* ***yes****)*	Enables access control.
routes (*IP@GW*; Default:)	Push routes in the following format: *Routes ::= Route[,Routes]* *Route ::= Dst[@Gw]*

Configuration example

In the following example, we will use RouterOS built-in ZeroTier controller to send our new network hosts appropriate certificates, credentials, and configuration information. The controller will operate from the "RouterOS Home" device and we will join in our network 3 units: mobile phone, laptop, RouterOS Office device, but theoretically, you can join up to 100 devices in one network.



RouterOS Home

First, we enable the default instance which operates at the **VL1** level :

```
[admin@Home] /zerotier> print
Columns: NAME, PORT, IDENTITY.PUBLIC
# NAME  PORT  IDENTITY.
PUBLIC

;;; ZeroTier Central controller - https://my.zerotier.com/
0 zt1   9993  879c0b5265:0:
d5fd2d17805e011d9b93ce8779385e427c8f405e520eea9284809d8444de0335a817xxb21aa4ba153bfbc229ca34d94e08de96d925a4aaa1
9b252da546693a28
```

Now we create a new network via the controller section which will operate at the **VL2** level. Each network has its own controller and each network ID is generated from the controller address and controller ID combination.

Note that we use the ***private=yes*** option for a more secure network:

```
[admin@Home] /zerotier> controller/add name=ZT-private instance=zt1 ip-range=172.27.27.10-172.27.27.20
private=yes routes=172.27.27.0/24
[admin@Home] /zerotier> controller/print
Columns: INSTANCE, NAME, NETWORK, PRIVATE
# INSTANCE  NAME        NETWORK           PRIVATE
0 zt1       ZT-private  879c0b5265a99e4b  yes
```

Add our new network under the interface section:

```
[admin@Home] /zerotier> interface/add network=879c0b5265a99e4b name=myZeroTier instance=zt1
[admin@Home] /zerotier> interface/print interval=1
```

```
Columns: NAME, MAC-ADDRESS, NETWORK, STATUS
# NAME        MAC-ADDRESS        NETWORK           STATUS
0 myZeroTier  4A:19:35:6E:00:6E  879c0b5265a99e4b  ACCESS_DENIED
```

Each new peer asks for a controller to join the network, in this situation, we have *ACCESS_DENIED* status and we have to authorize a new peer, that is because we used the **private=yes** option.

After authorization, each member in the network receives information from the controller about new peers and approval they can exchange packets with them:

```
[admin@Home] /zerotier> controller/member/print
Columns: NETWORK, ZT-ADDRESS
#  NETWORK     ZT-ADDRESS
0  ZT-private  879a0b5265
[admin@Home] /zerotier> controller/member/set 0 authorized=yes
```

Verify newly configured IP address and route:

```
[admin@Home] /zerotier> /ip/address/print where interface~"Zero"
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS          NETWORK      INTERFACE
4 D 172.27.27.15/24  172.27.27.0  myZeroTier

[admin@Home] /zerotier> /ip/route/pr where gateway~"Zero"
Flags: D - DYNAMIC; A - ACTIVE; c, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
    DST-ADDRESS     GATEWAY     DISTANCE
DAc 172.27.27.0/24  myZeroTier         0
```

RouterOS Office

Configuration on the Office device. We will enable the default instance and ask a controller to join the *879c0b5265a99e4b* network:

```
[admin@office] /zerotier> interface/add network=879c0b5265a99e4b instance=zt1 name=ZT-interface
[admin@office] /zerotier> interface/print interval=1
Columns: NAME, MAC-ADDRESS, NETWORK, STATUS
# NAME          MAC-ADDRESS        NETWORK           STATUS
0 ZT-interface  4A:40:1C:38:97:BA  879c0b5265a99e4b  ACCESS_DENIED
```

As previously, because our network is private, we have to authorize a new peer via "RouterOS home device". After that verify from controller received IP address and route:

```
[admin@Home] /zerotier> controller/member/print
Flags: A - AUTHORIZED
Columns: NETWORK, ZT-ADDRESS, IP-ADDRESS, LAST-SEEN
#    NETWORK     ZT-ADDRESS  IP-ADDRESS    LAST-SEEN
0 A  ZT-private  879a0b5265  172.27.27.15
1 A  ZT-private  554a914c7f  172.27.27.17
2 A  ZT-private  a83ac6032a  172.27.27.10
3    ZT-private  deba5dc5b1  172.27.27.13  3s348ms
[admin@Home] /zerotier> controller/member/set 3 authorized=yes
[admin@Home] /zerotier> controller/member/print
Flags: A - AUTHORIZED
Columns: NETWORK, ZT-ADDRESS, IP-ADDRESS, LAST-SEEN
#    NETWORK     ZT-ADDRESS  IP-ADDRESS    LAST-SEEN
0 A  ZT-private  879a0b5265  172.27.27.15
1 A  ZT-private  554a914c7f  172.27.27.17
2 A  ZT-private  a83ac6032a  172.27.27.10
3 A  ZT-private  deba5dc5b1  172.27.27.13  4s55ms
```

Verify via ZeroTier obtained IP address and route:

```
[admin@office] /zerotier> /ip/address/print where interface~"ZT"
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS           NETWORK      INTERFACE
0 D 172.27.27.13/24  172.27.27.0  ZT-interface

[admin@office] /zerotier> /ip/route/print where gateway~"ZT"
Flags: D - DYNAMIC; A - ACTIVE; c, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
    DST-ADDRESS     GATEWAY       DISTANCE
DAc 172.27.27.0/24  ZT-interface         0
```

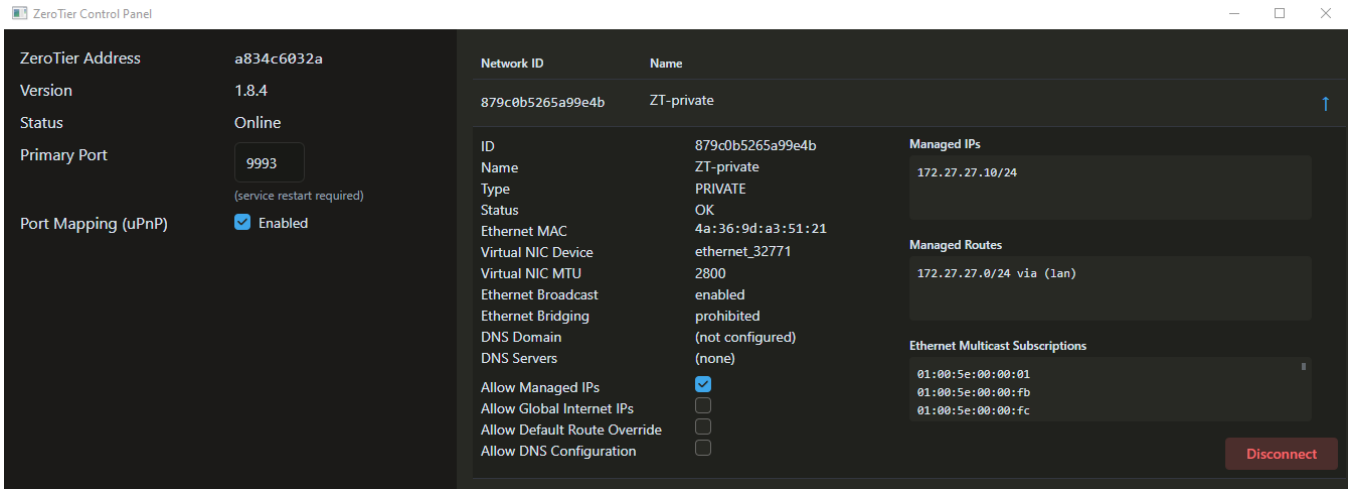
Other devices

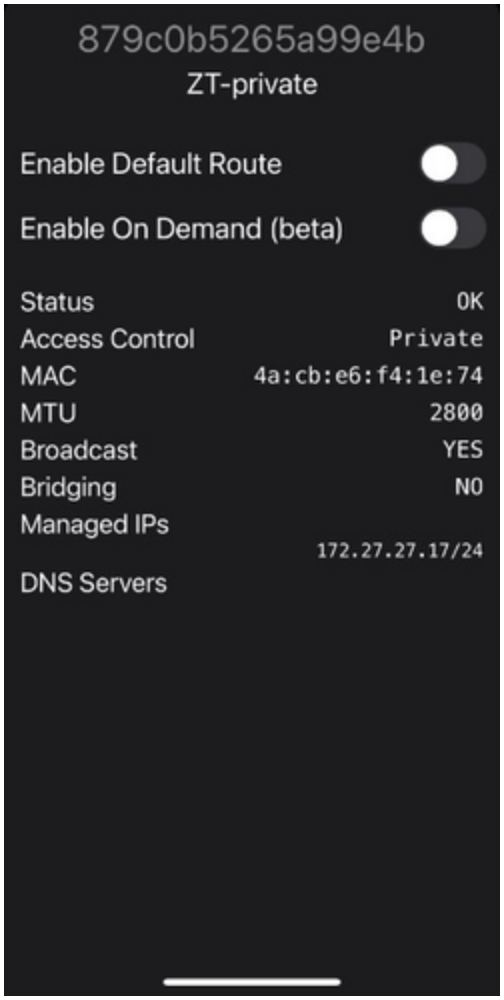
Download the ZeroTier app for your mobile phone or computer and join your newly created network:

1) Via our Laptop ZeroTier application we join the *879c0b5265a99e4b* network;

2) User Zerotier mobile app to join the *879c0b5265a99e4b* network;

> ⚠ Also all other new hosts you have to authorize under the */zerotier/controller/member/* section.




879c0b5265a99e4b
ZT-private
Enable Default Route
Enable On Demand (beta)
Status
OK
Access Control
Private
MAC
4a:cb:e6:f4:1e:74
MTU
2800
Broadcast
YES
Bridging
NO
Managed IPs
172.27.27.17/24
DNS Servers

Wired Connections

In This Section:

Ethernet

Summary

MikroTik RouterOS supports various types of Ethernet interfaces - ranging from 10Mbps to 10Gbps Ethernet over copper twisted pair, 1Gbps, 10Gbps, 25Gbps SFP/SFP+/SFP28 interfaces and 40Gbps, 100Gbps QSFP+/QSFP28 interfaces. Certain RouterBoard devices are equipped with a combo interface that simultaneously contains two interface types (e.g. Ethernet over twisted pair and SFP/SFP+ interface) allowing to select the most suitable option or creating a physical link failover. Through RouterOS, it is possible to control different Ethernet related properties like link speed, auto-negotiation, duplex mode, etc, monitor a transceiver diagnostic information and see a wide range of Ethernet related statistics.

For additional information about MikroTik SFP and QSFP type of interfaces and their compatibility, please refer to wired interface compatibility page.

Auto-negotiation and forced link mode

Auto-negotiation is a communication method and a set of steps employed by Ethernet devices connected via twisted pair cables. It enables these devices to agree on key transmission settings, including speed, duplex mode, and flow control. During this process, the connected devices initially exchange information about their capabilities concerning these settings. Afterward, they mutually select the best possible transmission mode that both devices can support effectively.

In RouterOS, auto-negotiation is always enabled by default and it selects the best possible transmission mode based on configured advertise arguments which acepts multiple values. Other option is to force the interface into single speed, but for this property to work it requires disabled auto-negotiation.

```
# Enabled auto-negotiation (default) with multiple advertise values
/interface ethernet
set [ find default-name=ether2 ] advertise=100M-baseT-full,1G-baseT-full

# Disabled auto-negotiation with single speed value
/interface ethernet
set [ find default-name=ether3 ] auto-negotiation=no speed=100M-baseT-full
```

When it comes to SFP and QSFP type of interfaces, auto-negotiation procedure differs significantly. Prior to RouterOS version 7.12, enabled auto-negotiation attempts to guess the maximum speed of the transceiver and interface (making the advertise setting inapplicable for SFP/QSFP interfaces), or disabled auto-negotation together with speed property attempts to force the speed.

After the RouterOS version 7.12, you can now utilize the advertise bits to specify the desired link modes you want to use. The speed arguments have also been revised to provide clearer representation, as the previous values were too ambiguous. Additionally, the full-duplex setting has been removed because the new link modes already encompass the duplex options. The fec-mode setting also plays important role for establishing a working link between SFP28, QSFP+ and QSFP28 interfaces. The same mode should be used on both link ends, otherwise FEC mismatch could result in non-working link or even false link-ups. RouterOS uses a disabled FEC mode as the default setting, but it can be changed to fec74 (aka FC-FEC) or fec91 (aka RS-FEC). For more information on FEC mode options, refer to the property description.

Link Modes (for advertise and speed properties)	Description	Supported FEC modes
10M-baseT-half	10M twisted-pair half-duplex	
10M-baseT-full	10M twisted-pair full-duplex	
100M-baseT-half	100M twisted-pair half-duplex	
100M-baseT-full	100M twisted-pair full-duplex	

1G-baseT-half	1G twisted-pair half-duplex	
1G-baseT-full	1G twisted-pair full-duplex	
1G-baseX	1G optical-fiber	
2.5G-baseT	2.5G twisted-pair full-duplex	
2.5G-baseX	2.5G optical-fiber	
5G-baseT	5G twisted-pair full-duplex	
10G-baseT	10G twisted-pair full-duplex	
10G-baseCR	10G twinaxial-copper	
10G-baseSR-LR	10G optical-fiber	
25G-baseCR	25G twinaxial-copper	fec74, fec91*
25G-baseSR-LR	25G optical-fiber	fec74, fec91*
40G-baseCR4	4x10G twinaxial-copper	fec74
40G-baseSR4-LR4	4x10G optical-fiber	fec74
50G-baseCR2	2x25G twinaxial-copper	fec74, fec91
50G-baseSR2-LR2	2x25G optical-fiber	fec74, fec91
100G-baseCR4	4x25G twinaxial-copper	fec91
100G-baseSR4-LR4	4x25G optical-fiber	fec91

Notes

The first number followed by M or G (e.g. 10M, 100M, 1G) represent the data transmission rates.

The symbols after "base" indicate the interface types. The **T** stands for twisted pair followed by duplex mode (either half or full). The **SR-LR** is shorthand for "short-range" and "long-range" optical modules, it also includes all other variants like LRM, ER, ZR, etc. These link modes should be used with optical modules. The **CR** stands for twinaxial copper and should be used with direct attach cable (DAC). For 1Gbps DAC the **1G-baseX** link mode is appropriate.

The last digit represent the number of lines, with "CR**2**" meaning two lines and "CR**4**" indicating four lines.

The CCR2004-1G-2XS-PCIe device does not support fec91 on SFP28 interfaces.

Each interface expose all the **supported** link mode capabilities. They can be monitored by the "`/interface ethernet monitor`" command:

```
[admin@MikroTik] > /interface/ethernet/monitor qsfp28-1-1
                     name: qsfp28-1-1
                supported: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full,
                           1G-baseX,2.5G-baseT,2.5G-baseX,5G-baseT,10G-baseT,10G-baseSR-LR,10G-baseCR,40G-
baseSR4-LR4,
                           40G-baseCR4,25G-baseSR-LR,25G-baseCR,50G-baseSR2-LR2,50G-baseCR2,100G-baseSR4-LR4,
100G-baseCR4
...
```

Then we can set advertise bits with the modes that we would like to use. When transceiver is inserted, RouterOS SFP handling tries to create a list of all the modes that this module and interface could support. These values can be monitored as **sfp-supported** and **advertising**:

```
[admin@MikroTik] > /interface/ethernet/monitor qsfp28-1-1
                     name: qsfp28-1-1
                supported: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full,
                           1G-baseX,2.5G-baseT,2.5G-baseX,5G-baseT,10G-baseT,10G-baseSR-LR,10G-baseCR,40G-
baseSR4-LR4,
                           40G-baseCR4,25G-baseSR-LR,25G-baseCR,50G-baseSR2-LR2,50G-baseCR2,100G-baseSR4-LR4,
100G-baseCR4
            sfp-supported: 10G-baseSR-LR,25G-baseSR-LR,100G-baseSR4-LR4
              advertising: 10G-baseSR-LR,25G-baseSR-LR,100G-baseSR4-LR4
...
```

The list of link modes are chosen like this:

link-modes = (interface-supported & advertising-bits) & sfp-supported-modes

In the end, the maximum rate of the link-modes is selected.

> RouterOS SFP/QSFP auto-negotiation does not support the same mechanisms as Ethernet interfaces, this is just extra way to influence guessing the link mode to set for interface.

One application for this is to configure all of the link modes within QSFP's. The overall configuration process starts with the topmost enabled port. If the chosen mode is valid and supported, it will be applied. If that particular link mode requires multiple lanes, the advertise and speed configuration of the next interface is ignored, but the interface should remain **enabled**. The next available free lane or port follows a similar process.

```
/interface ethernet
set [ find default-name=qsfp28-1-1 ] advertise=50G-baseCR2
set [ find default-name=qsfp28-1-3 ] advertise=25G-baseCR
set [ find default-name=qsfp28-1-4 ] advertise=25G-baseCR
```

In the situation when a warning message appears in the log records after connecting the DAC cable or optical module, as example:

```
10:20:47 interface,warning sfp-sfpplus1 module auto-initialization failed, try forced-mode
```

This may indicate that the connected cable or module has a corrupted or bad EEPROM checksum, which causes the automatic connection configuration to fail. This functionality was introduced in RouterOS version 7.12 and may affect some links that worked in the past by mistake with wrongly assumed module /cable attributes, which could lead to various problems related to link connection and device functionality.

In such cases, you can attempt to set the link mode manually, and this might help establish a working link. The following forced port setting examples are provided:

- For DAC cables and connection speeds of 1G/10G/25G:

```
/interface ethernet
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=1G-baseT-full
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=10G-baseCR
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=25G-baseCR
```

> Note: When selecting the interface speed setting, pay attention to what rates your DAC cable supports (check cable specification data)

- For optical modules and connection speeds of 1G/10G/25G:

```
/interface ethernet
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=1G-baseX
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=10G-baseSR-LR
set [ find default-name=sfp-sfpplus1 ] auto-negotiation=no speed=25G-baseSR-LR
```

Note: When selecting the interface speed setting, pay attention to what rates your optical module supports (check module specification data)

Properties

Sub-menu: `/interface ethernet`

This section describes the Ethernet interface configuration options.

Property	Description
advertise (since RouterOS v7.12: *10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4;* Default:) (older RouterOS: *10M-full | 10M-half | 100M-full | 100M-half | 1000M-full | 1000M-half | 2500M-full | 5000M-full | 10000M-full*; Default:)	Advertised link modes, only applies when auto-negotiation is enabled. Advertising higher speeds than the actual interface supported speed can result in undefined behavior. Multiple options are allowed.
arp (*disabled | enabled | local-proxy-arp | proxy-arp | reply-only*; Default: **enabled**)	Address Resolution Protocol mode: • disabled - the interface will not use ARP • enabled - the interface will use ARP • local-proxy-arp - the router performs proxy ARP on the interface and sends replies to the same interface • proxy-arp - the router performs proxy ARP on the interface and sends replies to other interfaces • reply-only - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the ARP table. No dynamic entries will be automatically stored in the ARP table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*auto | integer*; Default: **auto**)	How long the ARP record is kept in the ARP table after no packets are received from IP. Value `auto` equals to the value of `arp-timeout` in IP /Settings, default is 30s.
auto-negotiation (*yes | no*; Default: **yes**)	When enabled, the interface "advertises" its maximum capabilities to achieve the best connection possible. • **Note1:** Auto-negotiation should not be disabled on one end only, otherwise Ethernet Interfaces may not work properly. • **Note2:** Gigabit Ethernet and NBASE-T Ethernet links cannot work with auto-negotiation disabled.
bandwidth (*integer/integer*; Default: **unlimited/unlimited**)	Sets max rx/tx bandwidth in kbps that will be handled by an interface. TX limit is supported on all Atheros switch-chip ports. RX limit is supported only on Atheros8327/QCA8337 switch-chip ports.
cable-setting (*default | short | standard*; Default: **default**)	Changes the cable length setting (only applicable to NS DP83815/6 cards)
combo-mode (*auto | copper | sfp*; Default: **auto**)	When auto mode is selected, the port that was first connected will establish the link. In case this link fails, the other port will try to establish a new link. In case of a reboot, any of the two ports can be running, it depends on which port will successfully establish the link first. When sfp mode is selected, the interface will only work through SFP/SFP+ cage. When copper mode is selected, the interface will only work through RJ45 Ethernet port.
comment (*string*; Default:)	Descriptive name of an item

disable-running-check (*yes* / *no*; Default: **yes**)	Disable running check. If this value is set to 'no', the router automatically detects whether the NIC is connected with a device in the network or not. Default value is 'yes' because older NICs do not support it. (relevant only to CHR and x86)
fec-mode (*auto* / *fec74* / *fec91* / *off;* Default: **auto**)	Changes Forward Error Correction (FEC) mode for SFP28, QSFP+ and QSFP28 interfaces. Same mode should be used on both link ends, otherwise FEC mismatch could result in non-working link or even false link-ups. **It is recommended to enable FEC, particularly when creating link between CRS3xx, CRS5xx series switches. Some optical modules might rely on FEC functionality in MACs.** • auto - same as off • fec74 - enables IEEE 802.3 clause 74 FEC (aka FC-FEC), can be used with 25Gbps, 40Gbps and 50Gbps link modes • fec91 - enables IEEE 802.3 clause 91 FEC (aka RS-FEC), can be used with 25Gbps, 50Gbps and 100Gbps link modes • off - disabled FEC.
tx-flow-control (*on* / *off* / *auto*; Default: **off**)	When set to on, the port will generate pause frames to the upstream device to temporarily stop the packet transmission. Pause frames are only generated when some routers output interface is congested and packets cannot be transmitted anymore. **auto** is the same as **on** except when auto-negotiation=yes flow control status is resolved by taking into account what other end advertises.
rx-flow-control (*on* / *off* / *auto*; Default: **off**)	When set to on, the port will process received pause frames and suspend transmission if required. **auto** is the same as **on** except when auto-negotiation=yes flow control status is resolved by taking into account what other end advertises.
full-duplex (*yes* / *no*; Default: **yes**)	Defines whether the transmission of data appears in two directions simultaneously, only applies when auto-negotiation is disabled. Since RouterOS v7.12, the setting is replaced with new speed link modes.
l2mtu (*integer [0..65536]*; Default:)	Layer2 Maximum transmission unit. Read more.
mac-address (*MAC*; Default:)	Media Access Control number of an interface.
mdix-enable (*yes* / *no*; Default: **yes**)	Whether the MDI/X auto cross over cable correction feature is enabled for the port (Hardware specific, e.g. ether1 on RB500 can be set to yes/no. Fixed to 'yes' on other hardware.)
mtu (*integer [0..65536]*; Default: **1500**)	Layer3 Maximum transmission unit
name (*string*; Default:)	Name of an interface
orig-mac-address (*read-only: MAC*; Default:)	Original Media Access Control number of an interface.
passthrough-interface (*interface*; Default:)	Sets an interface in passthrough mode on CCR2004-1G-2XS-PCIe device. By default, the PCIe interface will show up as four virtual Ethernet interfaces. Two interfaces in passthrough mode to the 25G SFP28 cages. The remaining two virtual Ethernet-PCIe interfaces are bridged with the Gigabit Ethernet port for management access.
poe-out (*auto-on* / *forced-on* / *off*; Default: **off**)	Poe Out settings. `Read more`.
poe-priority (*integer [0..99]*; Default:)	Poe Out settings. `Read more`.
sfp-shutdown-temperature (*integer*; Default: **95** | **80**)	The temperature in Celsius at which the interface will be temporarily turned off due to too high detected SFP module temperature (introduced v6.48). The default value for SFP/SFP+/SFP28 interfaces is 95, and for QSFP+ /QSFP28 interfaces 80 (introduced v7.6).
sfp-rate-select (*high* / *low*; Default: **high**)	Allows to control rate select pin for SFP ports.

speed (since RouterOS v7.12: *10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4;* Default:) (older RouterOS: *10Mbps | 10Gbps | 100Mbps | 1Gbps | 2.5Gbps | 5Gbps | 25Gbps | 40Gbps | 100Gbps*; Default:)	Sets interface data transmission speed which takes effect only when auto-negotiation is disabled. Setting higher speeds than the actual interface supported speed can result in undefined behavior. Single option is allowed.

Read-only properties

Property	Description
running (*yes | no*)	Whether interface is running. Note that some interface does not have running check and they are always reported as "running".
slave (*yes | no*)	Whether interface is configured as a slave of another interface (for example Bonding or Bridge)
switch (*integer*)	ID to which switch chip interface belongs to.

Menu specific commands

Property	Description
blink (*[id, name]*)	Blink Ethernet leds
monitor (*[id, name]*)	Monitor ethernet status. Read more.
reset-counters (*[id, name]*)	Reset stats counters. Read more.
reset-mac-address (*[id, name]*)	Reset MAC address to manufacturers default.
cable-test (*string*)	Shows detected problems with cable pairs. `Read More`.

Monitor

To print out a current link rate and other Ethernet related properties or to see detailed diagnostics information for transceivers, use "`/interface ethernet monitor`" command. The provided information can differ for different interface types (e.g. Ethernet over twisted pair or SFP interface) or for different transceivers (e.g. SFP and QSFP).

Properties

Property	Description
advertising (since RouterOS v7.12: *10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4*) (older RouterOS: *10M-full | 10M-half | 100M-full | 100M-half | 1000M-full | 1000M-half | 2500M-full | 5000M-full | 10000M-full*)	Advertised link modes, only applies when auto-negotiation is enabled

auto-negotiation (*disabled* | *done* | *failed* | *incomplete*)	Current auto-negotiation status: • disabled - negotiation disabled • done - negotiation completed • failed - negotiation failed • incomplete - negotiation not completed yet
default-cable-settings (*short* | *standard*)	Default cable length setting (only applicable to NS DP83815/6 cards) • short - support short cables • standard - support standard cables
fec (*fec74* | *fec91* | *off*)	Currect FEC mode.
full-duplex (*yes* | *no*)	Whether transmission of data occurs in two directions simultaneously
link-partner-advertising (since RouterOS v7.12: *10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4*) (older RouterOS: *10M-full | 10M-half | 100M-full | 100M-half | 1000M-full | 1000M-half | 2500M-full | 5000M-full | 10000M-full*)	Link partner advertised link modes, only applies when auto-negotiation is enabled
rate (*10Mbps | 100Mbps | 1Gbps | 2.5Gbps | 5Gbps | 10Gbps | 25Gbps | 40Gbps | 50Gbps | 100Gbps*)	Actual data rate of the connection.
status (*link-ok | no-link | unknown*)	Current link status of an interface • link-ok - the card is connected to the network • no-link - the card is not connected to the network • unknown - the connection is not recognized (if the card does not report connection status)
tx-flow-control (*yes* | *no*)	Whether TX flow control is used
rx-flow-control (*yes* | *no*)	Whether RX flow control is used
combo-state (*copper* | *sfp*)	Used combo-mode for combo interfaces
sfp-module-present (*yes* | *no*)	Whether a transceiver is in cage

sfp-rx-lose (*yes* | *no*)	Whether a receiver signal is lost
sfp-tx-fault (*yes* | *no*)	Whether a transceiver transmitter is in fault state
sfp-type (*SFP/SFP+/SFP28/SFP56* | *DWDM-SFP/SFP+* | *QSFP* | *QSFP+* | *QSFP28/QSFP56*)	Used transceiver type
sfp-connector-type (*SC* | *LC* | *optical-pigtail* | *copper-pigtail* | *multifiber-parallel-optic-1x12* | *no-separable-connector* | *RJ45*)	Used transceiver connector type
sfp-link-length-9um (*m*)	Transceiver supported link length for single mode 9 /125um fiber
sfp-link-length-50um (*m*)	Transceiver supported link length for multi mode 50 /125um fiber (OM2)
sfp-link-length-62um (*m*)	Transceiver supported link length for multi mode 62.5 /125um fiber (OM1)
sfp-link-length-copper (*m*)	Supported link length of copper transceiver
sfp-vendor-name (*string*)	Transceiver manufacturer
sfp-vendor-part-number (*string*)	Transceiver part number
sfp-vendor-revision (*string*)	Transceiver revision number
sfp-vendor-serial (*string*)	Transceiver serial number
sfp-manufacturing-date (*date*)	Transceiver manufacturing date
sfp-wavelength (*nm*)	Transceiver transmitter optical signal wavelength
sfp-temperature (*C*)	Transceiver temperature
sfp-supply-voltage (*V*)	Transceiver supply voltage
sfp-tx-bias-current (*mA*)	Transceiver Tx bias current
sfp-tx-power (*dBm*)	Transceiver transmitted optical power
sfp-rx-power (*dBm*)	Transceiver received optical power
sfp-supported (*10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4*)	Module supported link modes. This property only applies to certain devices.
supported (*10M-baseT-half | 10M-baseT-full| 100M-baseT-half | 100M-baseT-full | 1G-baseT-half | 1G-baseT-full | 1G-baseX | 2.5G-baseT | 2.5G-baseX | 5G-baseT | 10G-baseT | 10G-baseSR-LR | 10G-baseCR | 40G-baseSR4-LR4 | 40G-baseCR4 | 25G-baseSR-LR | 25G-baseCR | 50G-baseSR2-LR2 | 50G-baseCR2 | 100G-baseSR4-LR4 | 100G-baseCR4*)	Shows the supported interface hardware link mode capabilities.
eeprom-checksum (*good* | *bad*)	Whether EEPROM checksum is correct
eeprom (*hex dump*)	Raw EEPROM of the transceiver

Example output of an Ethernet status:

```
[admin@MikroTik] > /interface ethernet monitor ether2
                     name: ether2
                   status: link-ok
          auto-negotiation: done
                     rate: 1Gbps
                full-duplex: yes
           tx-flow-control: no
           rx-flow-control: no
                 supported: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full
              advertising: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full
  link-partner-advertising: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full
```

Example output of an SFP status:

```
[admin@MikroTik] > /interface ethernet monitor sfp3
                     name: sfp3
                   status: link-ok
          auto-negotiation: done
                     rate: 1Gbps
                full-duplex: no
           tx-flow-control: no
           rx-flow-control: no
                 supported: 10M-baseT-half,10M-baseT-full,100M-baseT-half,100M-baseT-full,1G-baseT-half,1G-
baseT-full,1G-baseX
             sfp-supported: 1G-baseX
               advertising: 1G-baseX
  link-partner-advertising:
        sfp-module-present: yes
               sfp-rx-loss: no
              sfp-tx-fault: no
                  sfp-type: SFP/SFP+/SFP28/SFP56
        sfp-connector-type: LC
       sfp-link-length-om1: 500m
       sfp-link-length-om2: 550m
           sfp-vendor-name: Mikrotik
    sfp-vendor-part-number: S-85DLC05D
         sfp-vendor-serial: SG85M31401687
    sfp-manufacturing-date: 13-04-24
          sfp-wavelength: 850nm
         sfp-temperature: 33C
        sfp-supply-voltage: 3.237V
       sfp-tx-bias-current: 2mA
             sfp-tx-power: -5.792dBm
             sfp-rx-power: -5.22dBm
          eeprom-checksum: good
                  eeprom: 0000: 03 04 07 00 00 00 01 00  00 00 00 03 0d 00 00 00  ........ ........
                          0010: 37 32 00 00 4d 69 6b 72  6f 74 69 6b 20 20 20 20  72..Mikr otik
                          0020: 20 20 20 20 00 00 00 00  53 2d 38 35 44 4c 43 30      .... S-85DLC0
                          0030: 35 44 20 20 20 20 20 20  00 00 00 00 03 52 00 50  5D       .....R.P
                          0040: 00 1a 00 00 53 47 38 35  4d 33 31 34 30 31 36 38  ....SG85 M3140168
                          0050: 37 20 20 20 31 33 30 34  32 34 20 20 68 b0 01 f3  7   1304 24  h...
                          0060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
                          *
                          0080: 64 00 d2 ff 5a 00 d7 ff  8c a0 75 30 88 b8 79 18  d...Z... ..u0..y.
                          0090: 75 30 00 fa 61 a8 01 f4  1f 07 03 1a 18 a5 03 e7  u0..a... ........
                          00a0: 31 2e 00 13 27 12 00 27  00 00 00 00 00 00 00 00  1...'..' ........
                          00b0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
                          00c0: 00 00 00 00 3f 80 00 00  00 00 00 00 01 00 00 00  ....?... ........
                          00d0: 01 00 00 00 01 00 00 00  01 00 00 00 00 00 00 23  ........ .......#
                          00e0: 21 7c 7e 72 05 27 0a 4b  0b be 00 00 00 00 00 94  !|~r.'.K ........
                          00f0: 00 00 00 00 00 00 00 00  20 21 2a ff ff ff ff 00  ........  !*.....
```

Detect Cable Problems

A cable test can detect problems or measure the approximate cable length if the cable is unplugged on the other end and there is, therefore, "no-link". RouterOS will show:

- which cable pair is damaged
- the distance to the problem
- how exactly the cable is broken - short-circuited or open-circuited

This also works if the other end is simply unplugged - in that case, the total cable length will be shown.

Here is an example output:

```
[admin@CCR] > interface ethernet cable-test ether2
name: ether2
status: no-link
cable-pairs: open:4,open:4,open:4,open:4
```

In the above example, the cable is not shorted but “open” at 4 meters distance, all cable pairs are equally faulty at the same distance from the switch chip.

Currently `cable-test` is implemented on the following devices:

Devices	
CCR series devices	RB952Ui-5ac2nD
CRS1xx series devices	RB962UiGS-5HacT2HnT
CRS2xx series devices	RB1100AHx2
OmniTIK series devices	RB1100x4
RB450G series devices	RBD52G-5HacD2HnD
RB951 series devices	RBcAPGi-5acD2nD
RB2011 series devices	RBmAP2n
RB4011 series devices	RBmAP2nD
RB750Gr2	RBwsAP-5Hac2nD
RB750UPr2	RB3011UiAS-RM
RB751U-2HnD	RBMetal 2SHPn
RB850Gx2	RBDynaDishG-5HacD
RB931-2nD	RBLDFG-5acD
RB941-2nD	RBLHGG-5acD

⚠ Currently `cable-test` is not supported on Combo ports.

Stats

Using "`/interface ethernet print stats`" command, it is possible to see a wide range of Ethernet-related statistics. The list of statistics can differ between RouterBoard devices due to different Ethernet drivers. The list below contains all available counters across all RouterBoard devices. Most of the Ethernet statistics can be remotely monitored using SNMP and MIKROTIK-MIB.

Property	Description
driver-rx-byte (*integer*)	Total count of received bytes on device CPU
driver-rx-packet (*integer*)	Total count of received packets on device CPU
driver-tx-byte (*integer*)	Total count of transmitted bytes by device CPU
driver-tx-packet (*integer*)	Total count of transmitted packets by device CPU
fc-fec-block-corrected (*integer*)	Total count of FC-FEC corrected blocks. Applies only when fec74 mode is used.
fc-fec-block-uncorrected (*integer*)	Total count of FC-FEC uncorrected blocks. Applies only when fec74 mode is used.
fc-fec-rx-block (*integer*)	Total count of FC-FEC received blocks. Applies only when fec74 mode is used.
rs-fec-corrected (*integer*)	Total count of RS-FEC corrected codewords. Applies only when fec91 mode is used.
rs-fec-symbol-error (*integer*)	Total count of RS-FEC symbol errors. Applies only when fec91 mode is used.
rs-fec-uncorrected (*integer*)	Total count of RS-FEC uncorrected codewords. Applies only when fec91 mode is used.
rx-64 (*integer*)	Total count of received 64 byte frames
rx-65-127 (*integer*)	Total count of received 65 to 127 byte frames
rx-128-255 (*integer*)	Total count of received 128 to 255 byte frames
rx-256-511 (*integer*)	Total count of received 256 to 511 byte frames
rx-512-1023 (*integer*)	Total count of received 512 to 1023 byte frames
rx-1024-1518 (*integer*)	Total count of received 1024 to 1518 byte frames
rx-1519-max (*integer*)	Total count of received frames larger than 1519 bytes
rx-align-error (*integer*)	Total count of received align error events - packets where bits are not aligned along octet boundaries
rx-broadcast (*integer*)	Total count of received broadcast frames
rx-bytes (*integer*)	Total count of received bytes
rx-carrier-error (*integer*)	Total count of received frames with carrier sense error
rx-code-error (*integer*)	Total count of received frames with code error
rx-control (*integer*)	Total count of received control or pause frames
rx-error-events (*integer*)	Total count of received frames with the active error event
rx-fcs-error (*integer*)	Total count of received frames with incorrect checksum
rx-fragment (*integer*)	Total count of received fragmented frames (not related to IP fragmentation)
rx-ip-header-checksum-error (*integer*)	Total count of received frames with IP header checksum error
rx-jabber (*integer*)	Total count of received jabbed packets - a packet that is transmitted longer than the maximum packet length
rx-length-error (*integer*)	Total count of received frames with frame length error
rx-multicast (*integer*)	Total count of received multicast frames
rx-overflow (*integer*)	Total count of received overflowed frames can be caused when device resources are insufficient to receive a certain frame
rx-pause (*integer*)	Total count of received pause frames
rx-runt (*integer*)	Total count of received frames shorter than the minimum 64 bytes, is usually caused by collisions
rx-tcp-checksum-error (*integer*)	Total count of received frames with TCP header checksum error

rx-too-long (*integer*)	Total count of received frames that were larger than the maximum supported frame size by the network device, see the max-l2mtu property
rx-too-short (*integer*)	Total count of the received frame shorter than the minimum 64 bytes
rx-udp-checksum-error (*integer*)	Total count of received frames with UDP header checksum error
rx-unicast (*integer*)	Total count of received unicast frames
rx-unknown-op (*integer*)	Total count of received frames with unknown Ethernet protocol
tx-64 (*integer*)	Total count of transmitted 64 byte frames
tx-65-127 (*integer*)	Total count of transmitted 65 to 127 byte frames
tx-128-255 (*integer*)	Total count of transmitted 128 to 255 byte frames
tx-256-511 (*integer*)	Total count of transmitted 256 to 511 byte frames
tx-512-1023 (*integer*)	Total count of transmitted 512 to 1023 byte frames
tx-1024-1518 (*integer*)	Total count of transmitted 1024 to 1518 byte frames
tx-1519-max (*integer*)	Total count of transmitted frames larger than 1519 bytes
tx-align-error (*integer*)	Total count of transmitted align error events - packets where bits are not aligned along octet boundaries
tx-broadcast (*integer*)	Total count of transmitted broadcast frames
tx-bytes (*integer*)	Total count of transmitted bytes
tx-collision (*integer*)	Total count of transmitted frames that made collisions
tx-control (*integer*)	Total count of transmitted control or pause frames
tx-deferred (*integer*)	Total count of transmitted frames that were delayed on its first transmit attempt due to already busy medium
tx-drop (*integer*)	Total count of transmitted frames that were dropped due to the already full output queue
tx-excessive-collision (*integer*)	Total count of transmitted frames that already made multiple collisions and never got successfully transmitted
tx-excessive-deferred (*integer*)	Total count of transmitted frames that were deferred for an excessive period of time due to an already busy medium
tx-fcs-error (*integer*)	Total count of transmitted frames with incorrect checksum
tx-fragment (*integer*)	Total count of transmitted fragmented frames (not related to IP fragmentation)
tx-carrier-sense-error (*integer*)	Total count of transmitted frames with carrier sense error
tx-late-collision (*integer*)	Total count of transmitted frames that made collision after being already halfway transmitted
tx-multicast (*integer*)	Total count of transmitted multicast frames
tx-multiple-collision (*integer*)	Total count of transmitted frames that made more than one collision and subsequently transmitted successfully
tx-overflow (*integer*)	Total count of transmitted overflowed frames
tx-pause (*integer*)	Total count of transmitted pause frames
tx-all-queue-drop-byte (*integer*)	Total count of transmitted bytes dropped by all output queues
tx-all-queue-drop-packet (*integer*)	Total count of transmitted packets dropped by all output queues
tx-queueX-byte (*integer*)	Total count of transmitted bytes on a certain queue, the **X** should be replaced with a queue number
tx-queueX-packet (*integer*)	Total count of transmitted frames on a certain queue, the **X** should be replaced with a queue number
tx-runt (*integer*)	Total count of transmitted frames shorter than the minimum 64 bytes, is usually caused by collisions
tx-too-short (*integer*)	Total count of transmitted frames shorter than the minimum 64 bytes

tx-rx-64 (*integer*)	Total count of transmitted and received 64 byte frames
tx-rx-64-127 (*integer*)	Total count of transmitted and received 64 to 127 byte frames
tx-rx-128-255 (*integer*)	Total count of transmitted and received 128 to 255 byte frames
tx-rx-256-511 (*integer*)	Total count of transmitted and received 256 to 511 byte frames
tx-rx-512-1023 (*integer*)	Total count of transmitted and received 512 to 1023 byte frames
tx-rx-1024-max (*integer*)	Total count of transmitted and received frames larger than 1024 bytes
tx-single-collision (*integer*)	Total count of transmitted frames that made only a single collision and subsequently transmitted successfully
tx-too-long (*integer*)	Total count of transmitted packets that were larger than the maximum packet size
tx-underrun (*integer*)	Total count of underrun frames which can be caused when device resources are insufficient to transmit a certain frame
tx-unicast (*integer*)	Total count of transmitted unicast frames

For example, the output of Ethernet stats on the hAP ac2 device:

```
[admin@MikroTik] > /interface ethernet print stats
                     name:           ether1 ether2         ether3        ether4 ether5
            driver-rx-byte:  182 334 805 898      0  5 836 927 820    24 895 692      0
          driver-rx-packet:    4 449 562 546      0  4 320 155 362       259 449      0
            driver-tx-byte:   15 881 099 971      0 70 502 669 211    60 498 056     53
          driver-tx-packet:       52 724 428      0     54 231 229       106 498      1
                   rx-bytes:  178 663 398 808      0  5 983 590 739 1 358 140 795      0
              rx-too-short:                0      0              0             0      0
                     rx-64:       12 749 144      0        362 459       125 917      0
                  rx-65-127:        9 612 406      0     20 366 513       292 189      0
                 rx-128-255:        6 259 883      0      1 672 588       261 013      0
                 rx-256-511:        2 950 578      0        211 380       278 147      0
                rx-512-1023:        3 992 258      0        185 666       163 241      0
               rx-1024-1518:      119 034 611      0      2 796 559       696 254      0
                rx-1519-max:                0      0              0             0      0
                rx-too-long:                0      0              0             0      0
               rx-broadcast:       12 025 189      0      1 006 377        64 178      0
                   rx-pause:                0      0              0             0      0
               rx-multicast:        4 687 869      0         36 188       220 136      0
               rx-fcs-error:                0      0              0             0      0
             rx-align-error:                0      0              0             0      0
                rx-fragment:                0      0              0             0      0
                rx-overflow:                0      0              0             0      0
                   tx-bytes:   16 098 535 973      0 72 066 425 886   225 001 772      0
                      tx-64:        1 063 375      0        924 855        37 877      0
                   tx-65-127:       26 924 514      0      2 442 200       959 209      0
                  tx-128-255:       14 588 113      0        924 746       295 961      0
                  tx-256-511:        1 323 733      0      1 036 515        33 252      0
                 tx-512-1023:        1 287 464      0      2 281 554         3 625      0
                tx-1024-1518:        7 537 154      0     48 212 304        64 659      0
                 tx-1519-max:                0      0              0             0      0
                 tx-too-long:                0      0              0             0      0
                tx-broadcast:              590      0        145 800       823 038      0
                    tx-pause:                0      0              0             0      0
                tx-multicast:                0      0      1 039 243        41 716      0
                 tx-underrun:                0      0              0             0      0
                tx-collision:                0      0              0             0      0
    tx-excessive-collision:                  0      0              0             0      0
     tx-multiple-collision:                  0      0              0             0      0
       tx-single-collision:                  0      0              0             0      0
     tx-excessive-deferred:                  0      0              0             0      0
                tx-deferred:                 0      0              0             0      0
           tx-late-collision:                0      0              0             0      0
```

MikroTik wired interface compatibility

MikroTik SFP/SFP+/SFP28/QSFP+/QSFP28 compatibility

This article shows the compatibility of MikroTik devices with SFP, SFP+, SFP28, QSFP+, and QSFP28 transceivers. It features detailed compatibility tables that provide valuable insights into which transceivers are suitable for use with MikroTik devices. Additionally, some practical configuration examples are provided using the RouterOS CLI to set different data transmission rates. For more detailed descriptions of properties, please refer to the Ethernet user manual.

> MikroTik devices and SFP, SFP+, SFP28, QSFP+, and QSFP28 modules do not have any restrictions for other vendor equipment.
>
> While MikroTik cannot ensure full compatibility with modules from all manufacturers, as long as the other vendor modules and devices comply with transceiver multi-source agreement (MSA) they should be compatible with MikroTik.

1G SFP

Model	S-RJ01	S-85DLC05D	S-31DLC20D	S-3553LC20D	S-55DLC80D	S-4554LC80D	SFP CWDM	SFP 1m/3m DAC	S+AO0005 AOC	SFP28 1m/3m DAC
CCR1072-1G-8S+	+	+	+	+	+	+	+	+	+	+
CCR1036-12G-4S	+	+	+	+	+	+	+	+	+	+
CCR1036-8G-2S+	+	+	+	+	+	+	+	+	+	+
CCR1016-12S-1S+	+ 1	+ 1	+ 1	+ 1	+ 1	+ 1	+	+	+	+
CCR1009-7G-1C	+	+	+	+	+	+	+	+	+	+
CCR1009-8G-1S-1S+	+	+	+	+	+	+	+	+	+	+
CCR1009-7G-1C-1S+	+	+	+	+	+	+	+	+	+	+
CCR2004-1G-2XS-PCIe	-	-	-	-	-	-	-	-	-	-
CCR2004-1G-12S+2XS	+	+	+	+	+	+	+	+	+	+
CCR2004-16G-2S+	+	+	+	+	+	+	+	+	+	+
CCR2116-12G-4S+	+	+	+	+	+	+	+	+	+	+
CCR2216-1G-12XS-2XQ	+	+	+	+	+	+	+	+	+	+
CRS125-24G-1S	+	+	+	+	+	+	+	+	+	+
CRS305-1G-4S+	+	+	+	+	+	+	+	+	+	+
CRS309-1G-8S+	+	+	+	+	+	+	+	+	+	+
CRS312-4C+8XG	-	+	+	+	+	+	+	+	+	+
CRS318-1Fi-15Fr-2S	+	+	+	+	+	+	+	+	+	+
CRS318-16P-2S+	+	+	+	+	+	+	+	+	+	+
CRS326-4C+20G+2Q+	+	+	+	+	+	+	+	+	+	+

CRS326-24S+2Q+	+	+	+	+	+	+	+	+	+	+
CRS354-48G/P-4S+2Q+	+	+	+	+	+	+	+	+	+	+
CRS518-16XS-2XQ	+	+	+	+	+	+	+	+	+	+
CRS510-8XS-2XQ	+	+	+	+	+	+	+	+	+	+
CSS/CRS326-24G-2S+	+	+	+	+	+	+	+	+	+	+
CRS317-1G-16S+	+	+	+	+	+	+	+	+	+	+
CRS328-4C-20S-4S+	+	+	+	+	+	+	+	+	+	+
CRS328-24P-4S+	+	+	+	+	+	+	+	+	+	+
CRS226-24G-2S+	-	+ 2	+ 2	+ 2	+ 2	+ 2	+	+	+	+
CRS212-1G-10S-1S+	+ 1	+ 1	+ 1	+ 1	+ 1	+ 1	+	+	+	+
CRS210-8G-2S+	-	+ 2	+ 2	+ 2	+ 2	+ 2	+	+	+	+
CRS112-8G/P-4S	+	+	+	+	+	+	+	+	+	+
CRS109-8G-1S	+	+	+	+	+	+	+	+	+	+
CRS106-1C-5S/FiberBox	+	+	+	+	+	+	+	+	+	+
RB5009	+	+	+	+	+	+	+	+	+	+
RB4011	+	+	+	+	+	+	+	+	+	+
RB3011	+	+	+	+	+	+	+	+	+	+
RB2011	+	+	+	+	+	+	+	+	+	+
L009	+	+	+	+	+	+	+	+ 14	+ 14	+ 14
RB260/CSS106	+	+	+	+	+	+	+	+	+	+
RB922/921	+	+	+	+	+	+	+	+	+	+
RB953GS	+	+	+	+	+	+	+	+	+	+
hAP AC	+	+	+	+	+	+	+	+	+	+
hEX PoE/PowerBox Pro	+	+	+	+	+	+	+	+	+	+
hEX S	+	+	+	+	+	+	+	+	+	+
RBFTC11	+	+ 8	+ 8	+ 8	+ 8	+ 8	+ 8	+ 8	+ 8	+ 8
LHG XL 52 ac	-	+	+	+	+	+	+	+	+	+
RBD22/D23 mANTBox 52 15s/NetMetal ac²	-	+	+	+	+	+	+	+	+	+
L22/mANTBox ax 15s	+	+ 15	+ 15	+ 15	+ 15	+ 15	+ 15	+ 14	+ 14	+ 14
CSS610	+	+	+	+	+	+	+	+	+	+
CRS310-1G-5S-4S+/netFiber 9	+	+	+	+	+	+	+	+	+	+
CRS310-8G+2S+IN	+	+	+	+	+	+	+	+	+	+

10G SFP+/25G SFP28

Model	S+RJ 10	S+85DLC0 3D	S+31DLC1 0D	S+2332LC1 0D	SFP+ CWDM	SFP+ 1m/3m D AC	S+AO0005 A OC	Q+BC0003-S+	XQ+BC0003-XS+	SFP28 1m/3m D AC	SFP28 XS+31LC 10D	SFP28 XS+2733LC 15D
CCR1072-1G-8S+	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR1036-12G-4S	-	+	+	+	+	+	+	-	-	+	+	+
CCR1036-8G-2S+	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR1016-12S-1S+	+ 10	+	+	+	+	+	+	+ 1,5	+ 1,5	+	+	+
CCR1009-7G-1C	-	+	+	+	+	+	+	-	-	+	+	+
CCR1009-8G-1S-1S+	+ 10	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR1009-7G-1C-1S+	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR2004-1G-2XS-PCIe	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR2004-1G-12S+2XS	+ 11	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR2004-16G-2S+	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR2116-12G-4S+	+	+	+	+	+	+	+	+ 5	+ 5	+	+	+
CCR2216-1G-12XS-2XQ	+	+	+	+	+	+	+	+	+	+	+	+
CRS125-24G-1S	-	+	+	+	+	+	+	-	-	+	+	+

CRS305-1G-4S+	+ [7]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS309-1G-8S+	+ [4]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS312-4C+8XG	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS318-1Fi-15Fr-2S	-	+	+	+	+	+	+	-	-	+	+	+
CRS318-16P-2S+	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS326-4C+20G+2Q+	+	+	+	+	+	+	+	+	+	+	+	+
CRS326-24S+2Q+	+ [6]	+	+	+	+	+	+	+	+	+	+	+
CRS354-48G/P-4S+2Q+	+	+	+	+	+	+	+	+	+	+	+	+
CRS518-16XS-2XQ	+	+	+	+	+	+	+	+	+	+	+	+
CRS510-8XS-2XQ	+	+	+	+	+	+	+	+	+	+	+	+
CSS/CRS326-24G-2S+	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+ [9]	+ [9]
CRS317-1G-16S+	+ [3]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS328-4C-20S-4S+	+ [10]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS328-24P-4S+	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+ [9]	+ [9]
CRS226-24G-2S+	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS212-1G-10S-1S+	+ [10]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS210-8G-2S+	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS112-8G/P-4S	-	+	+	+	+	+	+	-	-	+	+	+
CRS109-8G-1S	-	+	+	+	+	+	+	-	-	+	+	+
CRS106-1C-5S/FiberBox	-	+	+	+	+	+	+	-	-	+	+	+
RB5009	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
RB4011	+	+	+	+	+	+	+	+	+	+	+	+
RB3011	-	+	+	+	+	+	+	-	-	+	+	+
RB2011	-	+	+	+	+	+	+	-	-	+	+	+
L009	-	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]
RB260/CSS106	-	+	+	+	+	+	+	-	-	+	+	+
RB922/921	-	+	+	+	+	+	+	-	-	+	+	+
RB953GS	-	+	+	+	+	+	+	-	-	+	+	+
hAP AC	-	+	+	+	+	+	+	-	-	+	+	+
hEX PoE/PowerBox Pro	-	+	+	+	+	+	+	-	-	+	+	+
hEX S	-	+	+	+	+	+	+	-	-	+	+	+
RBFTC11	-	+ [8]	+ [8]	+ [8]	+ [8]	+ [8]	+ [8]	-	-	+ [8]	+ [8]	+ [8]
LHG XL 52 ac	-	+	+	+	+	+	+	-	-	+	+	+
RBD22/D23 mANTBox 52 15s /NetMetal ac²	-	+	+	+	+	+	+	-	-	+	+	+
L22/mANTBox ax 15s	-	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]	+ [14]
CSS610	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS310-1G-5S-4S+ /netFiber 9	+ [10]	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+
CRS310-8G+2S+IN	+	+	+	+	+	+	+	+ [5]	+ [5]	+	+	+

40G QSFP+

Model	QSFP+ Q+DA0001	Q+85MP01D	Q+BC0003-S+
CRS326-4C+20G+2Q+	+	+	+
CRS326-24S+2Q+	+	+	+
CRS354-48G/P-4S+2Q+	+	+	+
CRS504-4XQ-IN	+	+	+
CRS504-4XQ-OUT	+ [13]	+	+ [13]
CRS518-16XS-2XQ	+	+	+
CRS510-8XS-2XQ	+	+	+
CCR2216-1G-12XS-2XQ	+	+	+

100G QSFP28

Model	XQ+31LC10D	XQ+31LC02D	XQ+85MP01D	XQ+DA0001	XQ+DA0003	XQ+BC0003-XS+	XQ+CM0000-XS+
CRS326-4C+20G+2Q+	-	-	+	+	+	+	+
CRS326-24S+2Q+	-	-	+	+	+	+	+
CRS354-48G/P-4S+2Q+	-	-	+	+	+	+	+
CRS504-4XQ-IN	+	+	+	+	+	+	+
CRS504-4XQ-OUT	+	+	+	+13	+13	+13	+
CRS518-16XS-2XQ	+	+	+	+	+	+	+
CRS510-8XS-2XQ	+	+	+	+	+	+	+
CCR2216-1G-12XS-2XQ	+	+	+	+	+	+	+

Legend		
Color codes:	Not supported	Check notes below

Notes:

1. **CCR1016-12S-1S+**, **CRS212-1G-10S-1S+** the SFP+1 interface does not work on any other link speed than 10G (does not support 1.25 G fiber optic transceivers)
2. **CRS226-24G-2S+**, **CRS210-8G-2S+** - the SFP+1 interface also supports SFP 1.25G fiber optic transceivers, SFP+2 works only with 10G transceivers/links.
3. **CSS/CRS317-1G-16S+** - power controller supports up to 10 simultaneous S+RJ10 modules.
4. **CSS/CRS309-1G-8S+** - We do not recommend using S+RJ10 in passive cooling devices without additional cooling, as they have relatively high power consumption and in turn high operating temperature.
5. **Q+BC0003-S+**, **XQ+BC0003-XS+** - SFP+ connector support in the SFP+/SFP28 cages
6. **CSS/CRS326-24S+2Q+** - supports up to 12 simultaneous S+RJ10 modules
7. **CSS/CRS305-1G-4S+** - supports up to 2 simultaneous S+RJ10 modules.
8. **RBFTC11** - works connected to other device 1G SFP port.
9. **XS+31LC10D**, **XS+2733LC15D** - full support has been added to CSS/CRS326-24G-2S+ and CRS328-24P-4S+ switches manufactured from October 2021.
10. **S+RJ10** - support in the SFP+ cages.
11. **CCR2004-1G-12S+2XS** - supports up to 6 simultaneous S+RJ10 modules.
12. **CCR2004-1G-2XS-PCIe** - supports only the same speed in both SFP28 ports (2 x 25G or 2 x 10G modes).
13. **CRS504-4XQ-OUT** - reduce IP code from IP66 to IP54.
14. **L009, L22/mANTBox ax 15s** - supports up to 2.5G rate (works with forced speed setting).
15. **L22/mANTBox ax 15s** - works with forced speed setting

S-RJ01

Table that states in what link rates if mounted in specific MikroTik devices S-RJ01 module will be able to work. Use these modules only with auto-negotiation enabled, forced link speeds are not supported. They will negotiate to correct duplex and highest possible rate.

Model	1000	100	10
RB5009	+	+	+
RB4011	+	+	+
RB3011	+	+	+
RB922	+	+	+
RB921	+	+	+
hAP ac	+	+	+
hEX PoE	+	+	+

hEX S	+	-	-
RB953	+/+	-/+	-/+
RB2011	+	-	-
RB260/CSS106	+	-	-
RBFTC11	+	-	-
LHG XL 52 ac	-	-	-
RBD22/D23 mANTBox 52 15s/NetMetal ac²	-	-	-
CRS106	+	+	+
CRS112	+	+	+
CRS125/CRS109	+	+	+
CRS212	+	+	+
CRS226/CRS210	-	-	-
CSS/CRS305-1G-4S+	+	+	+
CSS/CRS309-1G-8S+	+	+	+
CRS318-1Fi-15Fr-2S	+	+	+
CRS318-16P-2S+	+	+	+
CSS/CRS326-24G-2S+	+	+	+
CRS354	+	+	+
CSS/CRS328-4C-20S-4S+	+	+	+
CSS/CRS328-24P-4S+	+	+	+
CSS/CRS317-1G-16S+	+	+	+
CRS326-4C+20G+2Q+	+	+	+
CRS326-24S+2Q+	+	+	+
CSS610	+	+	+
CRS310-1G-5S-4S+/netFiber 9	+	+	+
CRS310-8G+2S+IN	+	+	+
CCR1009	+/+	+/+	+/+
CCR1016-12S-1S+	+	+	+
CCR1036-12G-4S	+	+	+
CCR1036-8G-2S+	+	+	+
CCR1072-1G-8S+	+	+	+

Notes:

- Rate works fine: +
- Rate does not work: -
- RB953: SFP1/SFP2
- CCR1009: SFP+/SFP

10 Gigabit Ethernet

S+RJ10

Use these modules only in 10G SFP+ ports with auto-negotiation enabled, forced link speeds and configurable link speed advertisements are not supported. They will negotiate to correct duplex and highest possible rate. For proper S+RJ10 module installation and recommended use case scenarios, please read S+RJ10 General Guidance.

Speed	Cable type	S+RJ10 to Ethernet port
10BASE-T	Cat5e/6	100m
100BASE-T	Cat5e/6	100m
1000BASE-T	Cat5e/6	100m
2.5GBASE-T	Cat5e/6 UTP	100m
2.5GBASE-T	Cat5e/6 STP	100m
5GBASE-T	Cat5e/6	100m
10GBASE-T	Cat6/7	30m

Negotiated speed highly depends on quality and length of the cables that are used.

S+RJ10 to S+RJ10 always will negotiate to highest possible rate.

The latest revision of S+RJ10 contains "/r2" by the end of serial number. It comes with following improvements:

- Jumbo frames up to 10218 Bytes at 2.5G, 5G and 10G speeds;
- Actual link speed reporting;
- DDM monitoring (Supply Voltage, Module temperature).

Link Speed	Max MTU
10Gbps	10218
5Gbps	10218
2.5Gbps	10218
1000Mbps	1504
100Mbps	1504
10Mbps	1504

CRS312-4C+8XG

10GE ports maximum supported cable length.

Speed	Cable type	10 Gigabit Ethernet ports
10BASE-T	Cat5e/6	100m
100BASE-T	Cat5e/6	100m
1000BASE-T	Cat5e/6	100m
2.5GBASE-T	Cat5e/6	100m
5GBASE-T	Cat5e/6	100m

10GBASE-T	Cat6/7	30m

Negotiated speed highly depends on quality and length of the cables that are used.

10GE ports doesn't support Half duplex mode with forced link speeds.

SFP interface compatibility with 100M optical transceivers

SFP interface on the listed devices is compatible with fast ethernet fiber links.

Compatible devices (interface):

- CCR1009-7G-1C (combo1)
- CCR1009-7G-1C-1S+ (combo1)
- CRS106-1C-5S (combo1)
- CRS328-4C-20S-4S+ (combo1 - combo4 and SFP1 - SFP20)
- LHG XL 52 ac
- RBD22/D23/mANTBox 52 15s/NetMetal ac²

SFP+ interface compatibility with 1G optical transceivers

For MikroTik devices with SFP+ interface that support both 10G and 1G link rate, following settings must be set on both linked devices for required interfaces. These settings only relate when optical SFP transceivers are used. In order to get them working in 1G link rate, use the following configuration:

```
# Since RouterOS v7.12
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=1G-baseX

# Older RouterOS
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=1Gbps full-duplex=yes
```

- auto-negotiation disabled
- port speed 1G
- full-duplex

Devices which SFP+ ports support 1G links:

- CCR2004-1G-12S+2XS - All SFP+ interfaces can be used in 1G mode if required.
- CCR2004-16G-2S+ - All SFP+ interfaces can be used in 1G mode if required.
- CCR1072-1G-8S+ - All SFP+ interfaces can be used in 1G mode if required.
- CCR1036-8G-2S+ - All SFP+ interfaces can be used in 1G mode if required.
- CCR1009-8G-1S-1S+ - All SFP+ interfaces can be used in 1G mode if required.
- CCR1009-7G-1C-1S+ - All SFP+ interfaces can be used in 1G mode if required.
- CSS326-24G-2S+RM - All SFP+ interfaces can be used in 1G mode if required.
- CRS3xx series switches - All SFP+ interfaces can be used in 1G mode if required.
- RB5009 series - SFP+1 interface can be used in 1G mode if required.
- RB4011 series - SFP+1 interface can be used in 1G mode if required.
- CRS226-24G-2S+ - Only SFP+1 supports 1G link speed, SFP+2 is for 10G links only.
- CRS210-8G-2S+ - Only SFP+1 supports 1G link speed, SFP+2 is for 10G links only.
- CSS610 series switches - All SFP+ interfaces can be used in 1G mode if required.

Devices which SFP+ interfaces can be used only for 10G links:

- CCR1016-12S-1S+
- CRS212-1G-10S-1S+

SFP+ interface compatibility with 10G/25G optical transceivers

MikroTik devices with SFP+ ports can establish 10G links using 10G/25G optical fiber transceivers, however additional SFP Rate Select setting must be configured to avoid data corruption during transmission. The following settings are required on the SFP+ interface:

```
# Since RouterOS v7.12
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=10G-baseSR-LR sfp-rate-select=low

# Older RouterOS
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=10Gbps full-duplex=yes sfp-rate-select=low
```

This requirement applies to MikroTik 10G/25G modules:

- XS+31LC10D
- XS+2733LC15D

SFP+/SFP28 interface compatibility with 2.5G transceivers

The 2.5G link rate support is implemented since RouterOS v7.3. MikroTik devices with SFP+ and SFP28 interfaces that support 2.5G link rate require following settings to be set on both linked device interfaces.

```
# Since RouterOS v7.12
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=2.5G-baseX

# Older RouterOS
/interface ethernet set sfp-sfpplus1 auto-negotiation=no speed=2.5Gbps full-duplex=yes
```

- auto-negotiation disabled
- port speed 2.5G
- full-duplex

Devices which support 2.5G links in SFP/SFP+/SFP28 ports:

- CRS3xx series switches - All SFP+ interfaces can be used in 2.5G mode if required.
- CCR2004-1G-12S+2XS - All SFP+ and SFP28 interfaces can be used in 2.5G mode if required.
- CCR2116-12G-4S+ - All SFP+ interfaces can be used in 2.5G mode if required.
- CRS518-16XS-2XQ - All SFP28 interfaces can be used in 2.5G mode if required.
- CCR2216-1G-12XS-2XQ - All SFP28 interfaces can be used in 2.5G mode if required.
- RB5009 series - SFP+ interface can be used in 2.5G mode if required.
- L009 series - **SFP** interface can be used in 2.5G mode if required.

QSFP+/QSFP28 interface supported link rates

In RouterOS, every QSFP+ and QSFP28 physical interface is divided into four subinterfaces. These subinterfaces correspond to the four transmission lines necessary for the operation of QSFP+ or QSFP28. The first digit after "qsfpplus" or "qsfp28-" indicates the numbering of the QSFP+ or QSFP28 physical interface. The second digit, ranging from 1 to 4, indicates each of the individual lines. Here are some examples of QSFP+ and QSFP28 interface printouts for better understanding:

```
/interface ethernet print
Flags: R - RUNNING
Columns: NAME, MTU, MAC-ADDRESS, ARP, SWITCH
 #   NAME            MTU  MAC-ADDRESS        ARP      SWITCH
 1   qsfpplus1-1    1500  48:8F:5A:B6:09:8C  enabled  switch1
 2   qsfpplus1-2    1500  48:8F:5A:B6:09:8D  enabled  switch1
 3   qsfpplus1-3    1500  48:8F:5A:B6:09:8E  enabled  switch1
 4   qsfpplus1-4    1500  48:8F:5A:B6:09:8F  enabled  switch1

/interface ethernet print
Flags: R - RUNNING
Columns: NAME, MTU, MAC-ADDRESS, ARP, SWITCH
 #   NAME         MTU  MAC-ADDRESS        ARP      SWITCH
 1   qsfp28-1-1  1500  DC:2C:6E:9E:11:14  enabled  switch1
 2   qsfp28-1-2  1500  DC:2C:6E:9E:11:15  enabled  switch1
 3   qsfp28-1-3  1500  DC:2C:6E:9E:11:16  enabled  switch1
 4   qsfp28-1-4  1500  DC:2C:6E:9E:11:17  enabled  switch1
```

The configuration and monitoring of each of these subinterfaces can vary based on factors such as auto-negotiation, advertise, speed configuration and transceiver type (e.g. break-out cable or single fiber). Further paragraphs describes in more detail the QSFP+ and QSFP28 supported rates and correct configuration.

> Disabling or enabling any of the four QSFP+/QSFP28 subinterfaces causes the entire port group to undergo reconfiguration, leading to a restart of all four lines.

QSFP+ interfaces of MikroTik CRS3xx series devices support following link speeds.

- 1x 40G
- 4x 10G
- 4x 1G

40G links can be established either with autonegotiation or forced 40G speed.
4x 10G and 4x 1G links can be established only with forced speeds and disabled autonegotiation.
In a single link mode only the first QSFP+ subinterface needs to be configured, although rest of the subinterfaces should remain enabled.

Example of forced 1x 40G speed with disabled autonegotiation. Starting from RouterOS version 7.12, in addition to choosing the right transmission rate, it's important to specify the correct link mode. For example, you might use CR4 for Direct Attach Copper (DAC) or SR4-LR4 for an optical module:

```
# Since RouterOS v7.12
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=40G-baseCR4
/interface ethernet set qsfpplus2-1 auto-negotiation=no speed=40G-baseSR4-LR4

# Older RouterOS
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=40Gbps full-duplex=yes
```

In four link mode all four QSFP+ subinterfaces support configuration of different speeds.

Example of forced 4x 10G speed with disabled autonegotiation:

```
# Since RouterOS v7.12
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-2 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-3 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-4 auto-negotiation=no speed=10G-baseCR

# Older RouterOS
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-2 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-3 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-4 auto-negotiation=no speed=10Gbps full-duplex=yes
```

QSFP28 interfaces of MikroTik CRS5xx series and CCR2216 devices support following link speeds.

- 1x 100G
- 2x 50G (since RouterOS v7.12)
- 1x 40G
- 4x 25G
- 4x 10G
- 4x 1G

Not supported: 1x 50G, 2x 40G

100G links can be established either with autonegotiation or forced 100G speed.
2x 50G, 1x 40G, 4x 25G, 4x 10G and 4x 1G links can be established only with forced speeds and disabled autonegotiation.
In a single link mode only the first QSFP28 subinterface needs to be configured, although rest of the subinterfaces should remain enabled.

Example of forced 1x 40G speed with disabled autonegotiation. Starting from RouterOS version 7.12, in addition to choosing the right transmission rate, it's important to specify the correct link mode. For example, you might use CR4 for Direct Attach Copper (DAC) or SR4-LR4 for an optical module:

```
# Since RouterOS v7.12
/interface ethernet set qsfp28-1-1 auto-negotiation=no speed=40G-baseCR4
/interface ethernet set qsfp28-2-1 auto-negotiation=no speed=40G-baseSR4-LR4

# Older RouterOS
/interface ethernet set qsfp28-1-1 auto-negotiation=no speed=40Gbps full-duplex=yes
```

In four link mode all four QSFP28 subinterfaces support configuration of different speeds.

Example of forced 4x 25G speed with disabled autonegotiation:

```
# Since RouterOS v7.12
/interface ethernet set qsfp28-1-1 auto-negotiation=no speed=25G-baseCR
/interface ethernet set qsfp28-1-2 auto-negotiation=no speed=25G-baseCR
/interface ethernet set qsfp28-1-3 auto-negotiation=no speed=25G-baseCR
/interface ethernet set qsfp28-1-4 auto-negotiation=no speed=25G-baseCR

# Older RouterOS
/interface ethernet set qsfp28-1-1 auto-negotiation=no speed=25Gbps full-duplex=yes
/interface ethernet set qsfp28-1-2 auto-negotiation=no speed=25Gbps full-duplex=yes
/interface ethernet set qsfp28-1-3 auto-negotiation=no speed=25Gbps full-duplex=yes
/interface ethernet set qsfp28-1-4 auto-negotiation=no speed=25Gbps full-duplex=yes
```

QSFP+ interface compatibility with QSFP+ to SFP+ breakout cables

MikroTik devices can establish links between QSFP+ and SFP+ ports using breakout cable when following settings are configured on the QSFP+ interface:

```
# Since RouterOS v7.12
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-2 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-3 auto-negotiation=no speed=10G-baseCR
/interface ethernet set qsfpplus1-4 auto-negotiation=no speed=10G-baseCR

# Older RouterOS
/interface ethernet set qsfpplus1-1 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-2 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-3 auto-negotiation=no speed=10Gbps full-duplex=yes
/interface ethernet set qsfpplus1-4 auto-negotiation=no speed=10Gbps full-duplex=yes
```

- auto-negotiation disabled
- port speed 10G
- full-duplex

On the SFP+ side, auto-negotiation remains enabled.

PWR Line

Summary

The PWR-Line series devices allow Ethernet-like connectivity between supported devices over regular power lines. When plugged into the same electrical circuit, the PWR-Line devices will establish connectivity by using the HomePlug AV standard.

Properties

arp (*disabled | enabled | proxy-arp | reply-only*; Default: **enabled**)	Address Resolution Protocol mode: • disabled - the interface will not use ARP • enabled - the interface will use ARP • proxy-arp - the interface will use the ARP proxy feature • reply-only - the interface will only reply to requests originating from matching IP address/MAC address combinations which are entered as static entries in the "/ip arp" table. No dynamic entries will be automatically stored in the ARP table. Therefore for communications to be successful, a valid static entry must already exist.
bandwidth (*integer /integer*; Default: **unlimited/unlimited**)	Sets max rx/tx bandwidth in kbps that will be handled by an interface. TX limit is supported on all Atheros switch-chip ports. RX limit is supported only on Atheros8327/QCA8337 switch-chip ports.
comment (*string*; Default:)	Descriptive name of an item
l2mtu (*integer [0..65536]*; Default:)	Layer2 Maximum transmission unit. MTU in RouterOS
mac-address (*MAC*; Default:)	Media Access Control number of an interface.
mtu (*integer [0..65536]*; Default: **1500**)	Layer3 Maximum transmission unit
name (*string*; Default:)	Name of an interface
orig-mac-address (*MAC*; Default:)	
rx-flow-control (*on | off | auto*; Default: **off**)	When set to on, the port will process received pause frames and suspend transmission if required. **auto** is the same as **on** except when auto-negotiation=yes flow control status is resolved by taking into account what the other end advertises. The feature is supported on AR724x, AR9xxx, and QCA9xxx CPU ports, all CCR ports, and all Atheros switch chip ports.
tx-flow-control (*on | off | auto*; Default: **off**)	When set to on, the port will send pause frames when specific buffer usage thresholds are met. **auto** is the same as **on** except when auto-negotiation=yes flow control status is resolved by taking into account what the other end advertises. The feature is supported on AR724x, AR9xxx, and QCA9xxx CPU ports, all CCR ports, and all Atheros switch chip ports.

Menu specific commands

Property	Description
configure ()	The command configures the attached PWR-Line device's network-key, network-password, plc-cco-selection-mode.
join ()	Initiates the pairing sequence which will look for other PWR-Line devices in the same electrical circuit that is also in the pairing mode. This mode lasts for 60 seconds.
leave ()	Initiates the leaving sequence which essentially randomizes the device's network-key.
monitor ()	Outputs PWR-Line-related statuses in real-time.

upgrade-firmware ()	Upgrades the PWR-Line device with specified firmware-file and pib-file files.

Configuration example

For two or more devices to be able to connect with each other, they must share the same network-key value. The currently configured network-key can be seen using the monitor command as plc-actual-network-key.

```
[admin@MikroTik] > /interface pwr-line monitor pwr-line1
name: pwr-line1
connection-to-plc: ok
tx-flow-control: no
rx-flow-control: no
plc-actual-network-key: c973947c200e1540b0f84b571d92bebe
plc-hw-platform: QCA7420
plc-sw-platform: MAC
plc-fw-version: 1.4.0(24-20180515-CS)
plc-line-freq: 50Hz
plc-zero-crossing: detected
plc-mac: B8:69:F4:C4:34:68
```

Method 1

There are two ways to set the same network-key on different devices. You can either use the network-key parameter which is a hashed version of network-password parameter. Or use the network-password parameter and let the router apply the hash on a human-readable string.

For example:

```
/interface pwr-line configure pwr-line1 network-password=mynetwork
```

is the same as:

```
/interface pwr-line configure pwr-line1 network-key=cb01fcc6167bf3d1edb1433c2ebde4b3
```

You must set the same key or password on all devices you wish to communicate with each other.

Method 2

It is possible to use the join and leave commands and make the PWR-Line devices automatically synchronize the network-key value. It is advised to use the leave command before using the join command to make sure a new network-key is randomly generated and the device is not part of any old network.

```
/interface pwr-line leave pwr-line1
```

Then we can issue the join command. When doing so, the pairing sequence is enabled for 60 seconds, meaning you have to enable pairing mode on another device within 60 seconds for them to successfully pair.

```
/interface pwr-line join pwr-line1
```

Method 3

It is also possible to set a specified role for the PWR-Line device (master or slave) with the plc-cco-selection-mode parameter.

Property	Description

plc-cco-selection-mode (*auto* | *always* | *never*; Default: **auto**)	Sets PWR-Line device mode: • auto - PWR-Line will automatically decide what role to take depending on the situation upon joining a PWR-Line network. • always - PWR-Line will always be forced to act as "central-coordinator" or master device. • never - PWR-Line will always be forced to act as a slave device.

Example:

```
/interface pwr-line configure pwr-line1 plc-cco-selection-mode=auto
```

```
/interface pwr-line configure pwr-line1 plc-cco-selection-mode=always
```

```
/interface pwr-line configure pwr-line1 plc-cco-selection-mode=never
```

Sync Button usage

- Hold 0.5 – 3 seconds to turn on sync mode. For 120 seconds will try to communicate with another PWR-LINE device. A blinking orange LED light indicates that it is in search mode. You have to also do the same on the other PWR-LINE device, so they can synchronize. Press the button again to cancel the search. You can also manually set the security keys in RouterOS settings.
- Hold 5 – 8 seconds to generate a new security key. This is needed to remove a PWR-LINE device from an existing PWR-LINE network.
- Hold 10 – 15 seconds to reset all PWR-LINE related settings.

Supported Hardware

The device is fully compatible with our PWR-LINE AP and the newest revisions of products that have a MicroUSB port, such as hAP lite, hAP lite tower, hAP mini, mAP, and mAP lite have a pwr-line interface. A simple software upgrade to v6.44+ enables this feature (supported by the mentioned devices with serial numbers that end with /9xx). PWR-LINE functionality is also supported by some previously manufactured units - if you have a unit with a serial number that ends with /8xx, upgrade to 6.44+ and see if the pwr-line interface shows up).

Wireless

In This Section:

This section will describe the configuration of 802.11 wireless protocols and best use examples.

Wireless capabilities of a router can greatly enhance the usability of your home or office network or provide a solution for industrial structures. Choosing the right device for setup can be a puzzle to inexperienced users. This guide intends to explain different parameters and suggest a thought process to not get lost in the vast selection of MikroTik routers.

RouterOS package type

Note that since RouterOS v7.13 some MikroTik devices can choose between two types of Wireless NPK package, depending on the required features and the device type. More details can be found in the respective documentation sections.

Old 802.11ac ARM CPU devices*

Feature	Needed packages	Notes
New drivers (WPA3, Fast Roaming)	routeros + wifi-qcom-ac	
Legacy drivers (Nstreme, Nv2)	routeros + wireless	
New Capsman and own real interfaces	routeros + wifi-qcom-ac	*Built-in cards work with new drivers*
New Capsman only controller	routeros	*Built-in cards are not used at all*
Old Capsman	routeros + wireless	*Actually old = dual. Built-in cards will work with legacy drivers*
Running both capsmans at the same time	routeros + wireless	*Built-in cards can only work with legacy drivers*

***wifi-qcom-ac**: Audience, Audience LTE kit, Chateau (all variants of D53), hAP ac^2, hAP ac^3, cAP ac, cAP XL ac, LDF 5 ac, LHG XL 5 ac, LHG XL 52 ac, NetMetal ac^2, mANTBox 52 15s, wAP ac (RBwAPG-5HacD2HnD), SXTsq 5 ac*

New 802.11ax devices

Feature	Needed packages	Notes
New drivers (WPA3, Fast Roaming)	routeros + wifi-qcom	
Legacy drivers (Nstreme, Nv2)	-	*Not possible*
New Capsman and own real interfaces	routeros + wifi-qcom	*Built-in cards work with new drivers*
New Capsman only controller	routeros	*Built-in cards are not used at all*
Old Capsman	routeros + wireless	*Actually old = dual. Loses built-in cards*
Running both capsmans at the same time	routeros + wireless	*Loses built-in cards*

Frequencies

MikroTik provides routers with interfaces in 3 frequency bands - 2.4GHz, 5GHz, and 60GHz. Each frequency band has its own advantages and use cases.

2.4GHz

Nowadays considered legacy because of overuse, it is still the most widely supported band. If you have a wireless client like phone, laptop or another device, it will most probably support this band. Even IoT devices often support 2.4GHz band. Because of the lower frequency, the 2.4GHz band can better overcome obstacles, so sequentially it has a bigger range than a 5GHz device, but it also usually has smaller throughput (internet speed). Also, it can severely suffer from interference (noise) from other 2.4GHz wireless devices, because almost every home access point supports 2.4GHz band and it performs well through walls and over large distances also, there are fewer frequencies to choose from (3 non-overlapping). If you have many close neighbors (apartments, shared office building) chances are 2.4GHz band will be saturated and performance will be lower. This band can also be used for industrial links.

5GHz

Usually, new phones and laptops also support 5GHz band. If your client device and router support 802.11ac (sometimes referred to as just "ac") it will be faster than the 2.4GHz band. 5GHz band has more frequencies to choose from, but also usually has a lower range than 2.4GHz band. If you have new client devices, your network will benefit from an ac router. The 5GHz band is also often used for industrial links, because of the big frequency range.

60GHz

Currently, there are very few client devices (phones, laptops) that support the 60GHz band. However, it offers cutting edge solutions for industrial links. For example, if you have 2 points that must be connected at distances up to 1500 meters you will get a 1 Gbps duplex link. For example, one of the MikroTik products is called the *Wireless Wire* because it provides the same speed as 1Gbps wired connection, but you will need a clear line of sight to establish the link.

Use case

RouterOS software on MikroTik devices provides broad and coherent configuration possibilities. RouterOS software allows you to use MikroTik devices in many ways, for example, if needed, a "home access point" device can be easily reconfigured to act as a client or form a point to point link, if needed. The "home AP" is simply the default configuration, but it can be changed to whatever configuration you wish. That being said, it is best to use hardware for its intended purpose.

Home AP for phones and laptops

Before you determine the most optimal choice for your setup, you should answer questions like how many clients you want to connect, what range you should cover and what speeds you want to get.

Client count

More connected clients mean higher latency and smaller throughput. We recommend 20-50 clients per interface to reach the peak performance, depending on conditions the number of clients can go up to 100 and still work stable. If clients will need high throughput or data traffic is time sensitive it is advised to plan fewer clients per access point. Often it is beneficial to choose simpler access points but place them denser.

Range

The range of wireless connections depends on many conditions. Some of those are antenna gain, transmit power of router and client device, interference from other devices, obstacles (walls, metal objects), router placement. An important factor to note is that all involved devices affect the achievable distance, meaning that no matter how strong and sensitive your AP is, a small phone will be limited by its own transmit power and sensitivity. One device is unable to cover large areas if the client devices are mobile phones. Usually, only a few hundred meters can be achieved and more AP devices are required to cover bigger areas.

- **Antenna gain** is measured in dBi and determines how narrow the beam is. The radiation pattern of 0 dBi (practically impossible) is of the shape of a sphere, 1.5 dBi - 5 dBi radiates to all directions almost equally but has some dents and sides where the signal will be stronger. > 9 dBi has an obvious directional radiation pattern. Antennas with higher antenna gain if properly positioned will reach further in the necessary direction.
- **Transmit power** measured in dBm or mW determines signal strength that is coming out of a wireless interface. Mobile devices usually have small transmit power to save battery power. Even if, for example, phone reports an OK received signal strength, the router may receive a weak signal from the phone.

- **Interference** from other devices increases the noise floor and it gets harder for the router to distinguish signal from noise, therefore, the signal must be stronger and client closer to the access point. Access points in the same frequency occupy the same air time decreasing throughput and increased latency.
- Some **objects** decrease (attenuate) signal strength while others reflect the signal. Usually, in buildings, you have to keep in mind walls and their thickness, floor, and ceiling, metallic objects, glass, and wood also attenuate the signal.
- **Placement** of access point also affects range. The access point shouldn't be covered by metallic objects or surfaces so the signal would have space to spread.

Speed

If speed is important, then you should choose a 5GHz wireless router with 802.11 ac support.

For other wireless antennas to connect (CPE to AP)

Often it is necessary to connect two or more points, like, connect buildings on campus or connect client homes to network, or establish a long link. MikroTik provides solutions in these situations too. In order to choose, you must know the distance, whether you have to connect two points (point to point - PtP) or multiple points (point to multipoint - PtMP) and what speeds you need.

Distance

Because in these situations we are dealing with directional antennas and big distances, you must keep in mind that alignment and line of sight are crucial. For small distances up to 1500 meters, we advise using 60 GHz devices which will provide stability and great speed. Also, in small distances, 2.4 GHz or 5 GHz devices with small antenna gain will do just fine, although, you won't get such speeds as with 60 GHz devices. For longer links antenna gain and transmit power should be taken into consideration. Higher signal strength will allow higher data rates which mean higher throughput. Choose your frequency wisely to escape interference with other wireless links. Keep in mind that in cities even above the roof there often is interference from neighboring 2.4 GHz and 5 GHz links.

PtP or PtMP

PtP and PtMP links differ in some areas. For PtMP you most probably will want an antenna with a wider beam also called - sector antenna. Such antenna covers a wider angle but also has less gain, therefore, less distance. Also in PtMP access point must have at least level 4 RouterOS software license. Clients or devices that are connected to the access point and devices in PtP link can have license level 3 and narrower beamwidth.

Speed

For higher speeds in short links, you must choose 60 GHz devices, in longer distances - 5GHz ac devices.

Note

For controlling large networks of access points you can use AP Controller (CAPsMAN) (Controlled access point manager). All RouterOS devices can act as CAPsMAN servers, however, it is advised to use appropriate devices for the task, with higher CPU power and more RAM. All RouterOS devices with 2GHz and/or 5GHz interface and software level at least 4 can be CAP (Controlled access point) clients and connect to a CAPsMAN server. For controlling hundreds of access points, we advise using our CCR series devices. For controlling thousands of access point you might want to try using CHR or an x86 machine.

WiFi

Overview

The 'WiFi' configuration menu, introduced in **RouterOS 7.13**, is a RouterOS menu for managing Wi-Fi 5 wave2 and newer WiFi interfaces.

Devices with compatible radios also require either the 'wifi-qcom-ac' driver package (for 802.11ac chipsets) or the 'wifi-qcom' driver package for 802.11ax and newer chipsets.

The configuration menu used to be called 'wifiwave2' in RouterOS versions before 7.13, where it was a part of the 'wifiwave2' software package.

WiFi Terminology

Before we move on let's familiarize ourselves with terms important for understanding the operation of the menu. These terms will be used throughout the article.

- **Profile** - refers to the configuration preset created under one of this WiFi sub-menus: **aaa**, **channel**, **security**, **datapath**, or **interworking**.
- **Configuration profile** - configuration preset defined under /interface/wifi/configuration, it can reference various profiles.
- **Station** - wireless client.

Basic Configuration

Basic password-protected AP

```
/interface/wifi
set wifi1 disabled=no configuration.country=Latvia configuration.ssid=MikroTik security.authentication-
types=wpa2-psk,wpa3-psk security.passphrase=8-63_characters
```

Open AP with OWE transition mode

Opportunistic wireless encryption (OWE) allows the creation of wireless networks that do not require the knowledge of a password to connect, but still offer the benefits of traffic encryption and management frame protection. It is an improvement on regular open access points.

However, since a network cannot be simultaneously encrypted and unencrypted, 2 separate interface configurations are required to offer connectivity to older devices that do not support OWE and offer the benefits of OWE to devices that do.

This configuration is referred to as OWE transition mode.

```
/interface/wifi
add master-interface=wifi1 name=wifi1_owe configuration.ssid=MikroTik_OWE security.authentication-types=owe
security.owe-transition-interface=wifi1 configuration.hide-ssid=yes
set wifi1 configuration.country=Latvia configuration.ssid=MikroTik security.authentication-types="" security.
owe-transition-interface=wifi1_owe
enable wifi1,wifi1_owe
```

Client devices that support OWE will prefer the OWE interface. If you don't see any devices in your registration table that are associated with the regular open AP, you may want to move on from running a transition mode setup to a single OWE-encrypted interface.

Resetting configuration

WiFi interface configurations can be reset by using the 'reset' command.

```
/interface/wifi reset wifi1
```

Configuration profiles

One of the new WiFi additions is configuration profiles, you can create various presets, that can be assigned to interfaces as needed. Configuration settings for WiFi are grouped in **profiles** according to the parameter sections found at the end of this page - **aaa**, **channel**, **configuration**, **datapath**, **interworking**, and **security**, and can then be assigned to interfaces. **Configuration profiles** can include other profiles as well as separate parameters from other categories.

This optional flexibility is meant to allow each user to arrange their configuration in a way that makes the most sense for them, but it also means that each parameter may have different values assigned to it in different sections of the configuration.

The following priority determines, which value is used:

1. Value in interface settings
2. Value in a profile assigned to the interface
3. Value in configuration profile assigned to interface
4. Value in a profile assigned to the configuration profile (which in turn is assigned to the interface).

If you are at any point unsure of which parameter value will be used for an interface, consult the **actual-configuration** menu. For an example of configuration profile usage, see the following example.

Example for dual-band home AP

```
# Creating a security profile, which will be common for both interfaces
/interface wifi security
add name=common-auth authentication-types=wpa2-psk,wpa3-psk passphrase="diceware makes good passwords"
wps=disable
# Creating a common configuration profile and linking the security profile to it
/interface wifi configuration
add name=common-conf ssid=MikroTik country=Latvia security=common-auth
# Creating separate channel configurations for each band
/interface wifi channel
add name=ch-2ghz frequency=2412,2432,2472 width=20mhz
add name=ch-5ghz frequency=5180,5260,5500 width=20/40/80mhz
# Assigning to each interface the common profile as well as band-specific channel profile
/interface wifi
set wifi1 channel=ch-2ghz configuration=common-conf disabled=no
set wifi2 channel=ch-5ghz configuration=common-conf disabled=no

/interface/wifi/actual-configuration print
 0 name="wifi1" mac-address=74:4D:28:94:22:9A arp-timeout=auto radio-mac=74:4D:28:94:22:9A
   configuration.ssid="MikroTik" .country=Latvia
   security.authentication-types=wpa2-psk,wpa3-psk .passphrase="diceware makes good passwords" .wps=disable
   channel.frequency=2412,2432,2472 .width=20mhz

 1 name="wifi2" mac-address=74:4D:28:94:22:9B arp-timeout=auto radio-mac=74:4D:28:94:22:9B
   configuration.ssid="MikroTik" .country=Latvia
   security.authentication-types=wpa2-psk,wpa3-psk .passphrase="diceware makes good passwords" .wps=disable
   channel.frequency=5180,5260,5500 .width=20/40/80mhz
```

Access List

The access list provides multiple ways of filtering and managing wireless connections.

RouterOS will check each new connection to see if its parameters match the parameters specified in any access list rule.

The rules are checked in the order they appear in the list. Only management actions specified in the first matching rule are applied to each connection.

Connections, which have been accepted by an access list rule, will be periodically checked, to see if they remain within the permitted **time** and **signal-range**. If they do not, they will be terminated.

Take care when writing access list rules which reject clients. After being repeatedly rejected by an AP, a client device may start avoiding it.

The access list has two kinds of parameters - filtering, and action. Filtering properties are only used for matching clients, to whom the access list rule should be applied to. Action parameters can change connection parameters for that specific client and potentially overriding its default connection parameters with ones specified in the access list rule.

MAC address authentication

Implemented through the **query-radius** action, MAC address authentication is a way to implement a centralized whitelist of client MAC addresses using a RADIUS server.

When a client device tries to associate with an AP, which is configured to perform MAC address authentication, the AP will send an access-request message to a RADIUS server with the device's MAC address as the user name and an empty password. If the RADIUS server answers with access-accept to such a request, the AP proceeds with whatever regular authentication procedure (passphrase or EAP authentication) is configured for the interface.

Access rule examples

Only accept connections to guest network from nearby devices during business hours

```
/interface/wifi/access-list/print detail
Flags: X - disabled
 0   signal-range=-60..0 allow-signal-out-of-range=5m ssid-regexp="MikroTik Guest" time=7h-19h,mon,tue,wed,thu,
fri action=accept

 1   ssid-regexp="MikroTik Guest" action=reject
```

Reject connections from locally-administered ('anonymous'/'randomized') MAC addresses

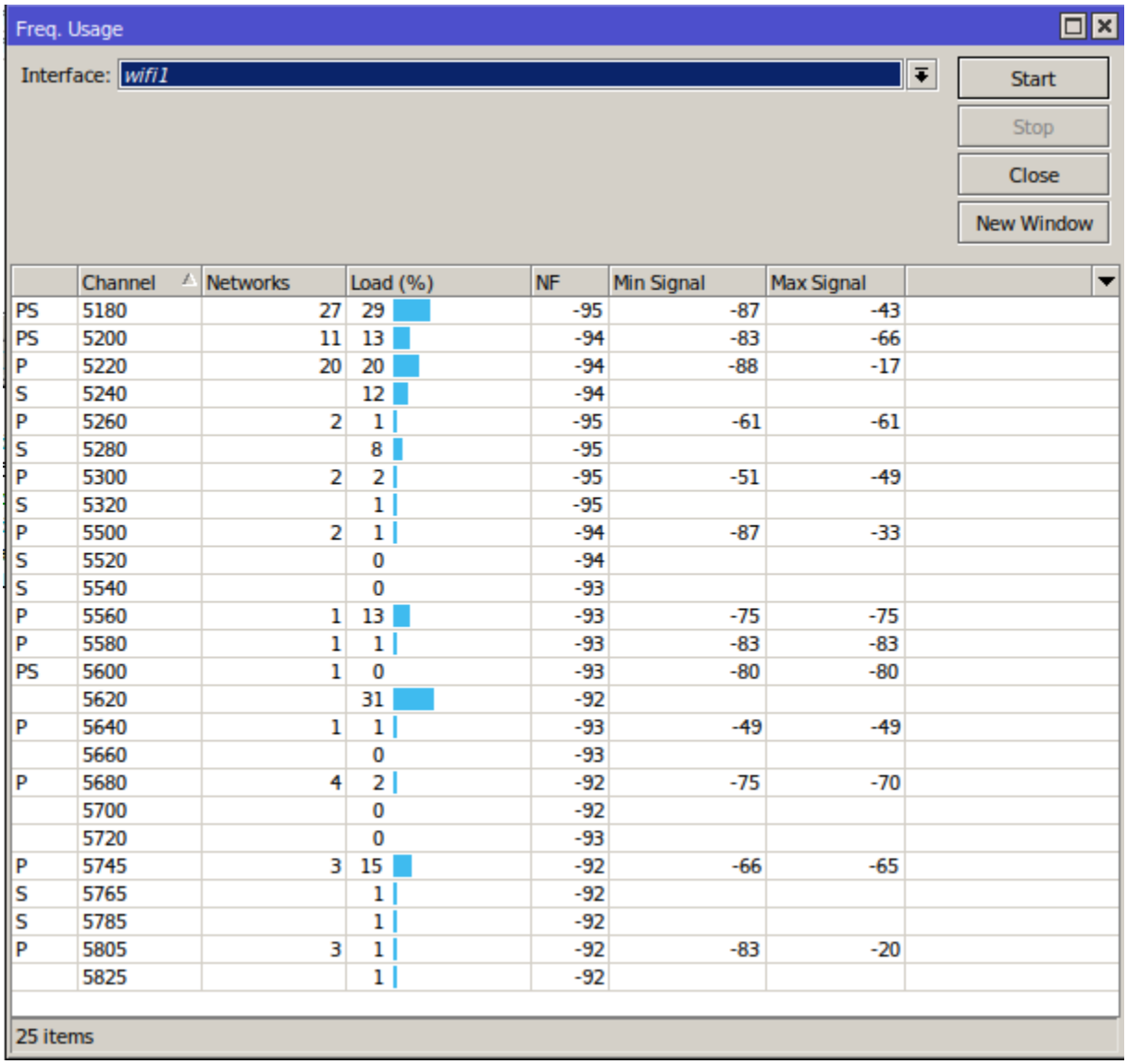
```
/interface/wifi/access-list/print detail
Flags: X - disabled
 0   mac-address=02:00:00:00:00:00 mac-address-mask=02:00:00:00:00:00 action=reject
```

Assigning a different passphrase for a specific client can be useful, if you need to provide wireless access to a client, but don't want to share your wireless password, or don't want to create a separate SSID. When the matching client connects to this network, instead of using the password defined in the interface configuration, the access list will make that client use a different password. Just make sure that the specific client doesn't get matched by a more generic access list rule first.

```
/interface wifi access-list
add action=accept disabled=no mac-address=22:F9:70:E5:D2:8E interface=wifi1 passphrase=StrongPassword
```

Frequency scan

The '/interface/wifi/frequency-scan wifi1' command provides information about RF conditions on available channels that can be obtained by running the frequency-scan command. Used to approximate the spectrum usage, it can be useful to find less crowded frequencies.



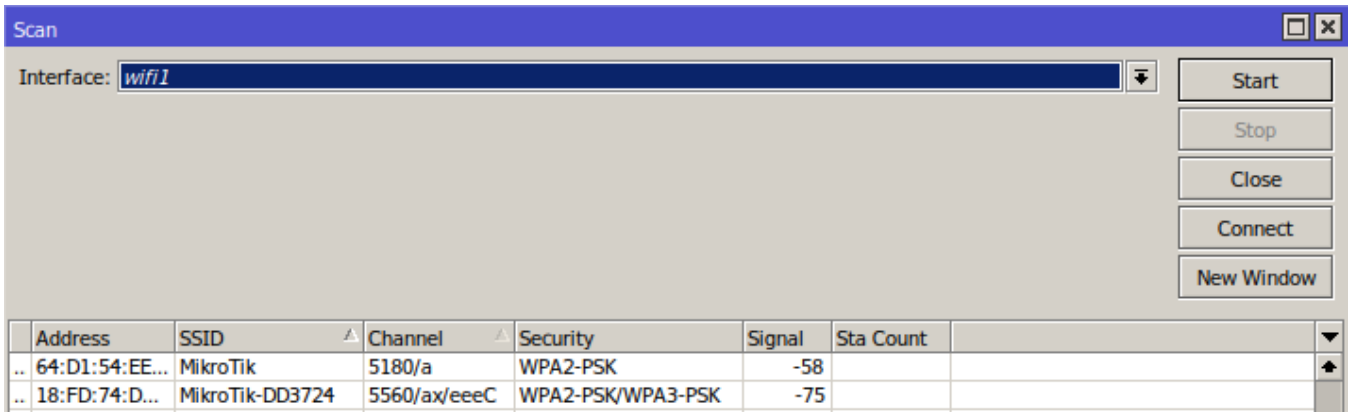
	Channel	Networks	Load (%)	NF	Min Signal	Max Signal
PS	5180	27	29	-95	-87	-43
PS	5200	11	13	-94	-83	-66
P	5220	20	20	-94	-88	-17
S	5240		12	-94		
P	5260	2	1	-95	-61	-61
S	5280		8	-95		
P	5300	2	2	-95	-51	-49
S	5320		1	-95		
P	5500	2	1	-94	-87	-33
S	5520		0	-94		
S	5540		0	-93		
P	5560	1	13	-93	-75	-75
P	5580	1	1	-93	-83	-83
PS	5600	1	0	-93	-80	-80
	5620		31	-92		
P	5640	1	1	-93	-49	-49
	5660		0	-93		
P	5680	4	2	-92	-75	-70
	5700		0	-92		
	5720		0	-93		
P	5745	3	15	-92	-66	-65
S	5765		1	-92		
S	5785		1	-92		
P	5805	3	1	-92	-83	-20
	5825		1	-92		

Running a frequency scan will disconnect all connected clients, or if the interface is in station mode, it will disconnect from AP.

Scan command

The '/interface wifi scan' command will scan for access points and print out information about any APs it detects. It doesn't show the frequency usage, per channel, but it will reveal all access points that are transmitting. You can use the "connect" button, to initiate a connection to a specific AP.

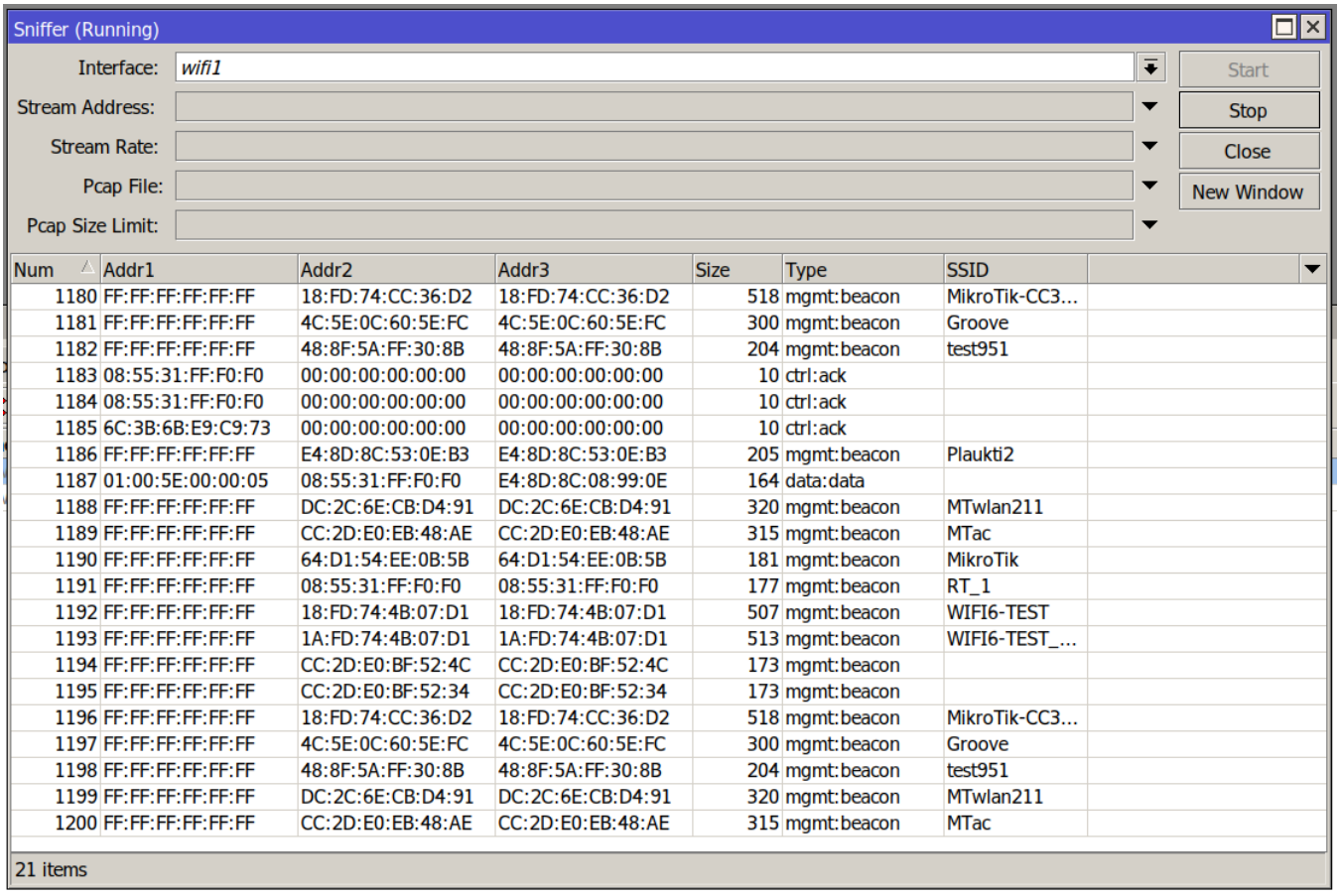
The scan command takes all the same parameters as the frequency-scan command.



Sniffer

The sniffer command enables monitor mode on a wireless interface. This turns the interface into a passive receiver for all WiFi transmissions.
The command continuously prints out information on received packets and can save them locally to a pcap file or stream them using the TZSP protocol.

The sniffer will operate on whichever channel is configured for the chosen interface.



WPS

WPS client

The wps-client command enables obtaining authentication information from a WPS-enabled AP.

```
/interface/wifi/wps-client wifi1
```

WPS server

An AP can be made to accept WPS authentication by a client device for 2 minutes by running the following command.

```
/interface/wifi wps-push-button wifi1
```

Radios

Information about the capabilities of each radio can be gained by running the `/interface/wifi/radio print detail` command. It can be useful to see what bands are supported by the interface and what channels can be selected. The country profile that is applied to the interface will influence the results.

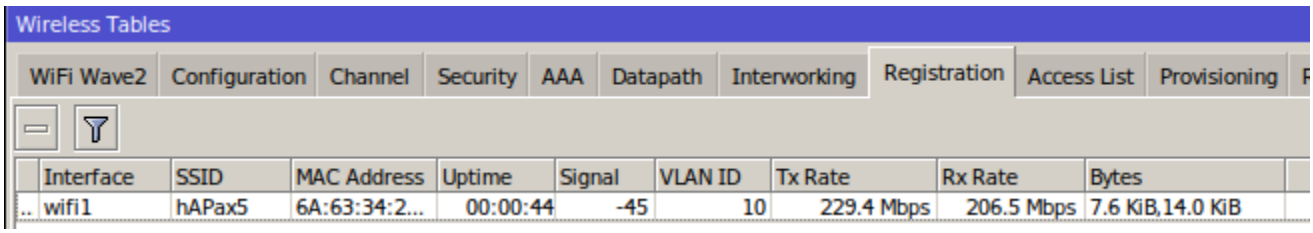
```
interface/wifi/radio/print detail
Flags: L - local
 0 L radio-mac=48:A9:8A:0B:F7:4A phy-id=0 tx-chains=0,1 rx-chains=0,1
     bands=5ghz-a:20mhz,5ghz-n:20mhz,20/40mhz,5ghz-ac:20mhz,20/40mhz,20/40/80mhz,5ghz-ax:20mhz,
      20/40mhz,20/40/80mhz
     ciphers=tkip,ccmp,gcmp,ccmp-256,gcmp-256,cmac,gmac,cmac-256,gmac-256 countries=all
     5g-channels=5180,5200,5220,5240,5260,5280,5300,5320,5500,5520,5540,5560,5580,5600,5620,5640,5660,
            5680,5700,5720,5745,5765,5785,5805,5825
     max-vlans=128 max-interfaces=16 max-station-interfaces=3 max-peers=120 hw-type="QCA6018"
     hw-caps=sniffer interface=wifi1 current-country=Latvia
     current-channels=5180/a,5180/n,5180/n/Ce,5180/ac,5180/ac/Ce,5180/ac/Ceee,5180/ax,5180/ax/Ce,
                 5180/ax/Ceee,5200/a,5200/n,5200/n/eC,5200/ac,5200/ac/eC,5200/ac/eCee,5200/ax...
                 ...5680/n/eC,5680/ac,5680/ac/eC,5680/ax,5680/ax/eC,5700/a,5700/n,5700/ac,5700/ax
     current-gopclasses=115,116,128,117,118,119,120,121,122,123 current-max-reg-power=30
```

While Radio information gives us information about supported channel width, it is also possible to deduce this information from the product page, to do so you need to check the following parameters: **number of chains**, **max data rate**. Once you know these parameters, you need to check the modulation and coding scheme (MCS) table, for example, here: https://mcsindex.com/.

If we take hAP ax^2, as an example, we can see that number of chains is 2, and the max data rate is 1200 - 1201 in the MCS table. In the MCS table we need to find entry for 2 spatial streams - chains, and the respective data rate, which in this case shows us that 80MHz is the maximum supported channel width.

Registration table

'/interface/wifi/registration-table/' displays a list of connected wireless clients and detailed information about them.



De-authentication

Wireless peers can be manually de-authenticated (forcing re-association) by removing them from the registration table.

```
/interface/wifi/registration-table remove [find where mac-address=02:01:02:03:04:05]
```

WiFi CAPsMAN

WiFi CAPsMAN allows applying wireless settings to multiple MikroTik WiFi AP devices from a central configuration interface.

More specifically, the Controlled Access Point system Manager (CAPsMAN) allows the centralization of wireless network management. When using the CAPsMAN feature, the network will consist of a number of 'Controlled Access Points' (CAP) that provide wireless connectivity and a 'system Manager' (CAPsMAN) that manages the configuration of the APs, it also takes care of client authentication.

WiFi CAPsMAN only passes wireless configuration to the CAP, all forwarding decisions are left to the CAP itself - there is no CAPsMAN forwarding mode.

Requirements:

- Any RouterOS device, that supports the WiFi package, can be a controlled wireless access point (CAP) as long as it has at least a Level 4 RouterOS license.
- WiFi CAPsMAN server can be installed on any RouterOS device that supports the WiFi package, even if the device itself does not have a wireless interface
- Unlimited CAPs (access points) supported by CAPsMAN

> WiFi CAPsMAN can only control WiFi interfaces, and WiFi CAPs can join only WiFi CAPsMAN, similarly, regular CAPsMAN only supports non-WiFi caps.
>
> The CAPs don't send traffic usage information to CAPsMAN.

CAPsMAN - CAP simple configuration example:

CAPsMAN in WiFi uses the same menu as a regular WiFi interface, meaning when you pass configuration to CAPs, you have to use the same configuration, security, channel configuration, etc. as you would for regular WiFi interfaces.

> You can configure sub-configuration menus, directly under "/interface/wifi/configuration" or reference previously created profiles in the main configuration profile

CAPsMAN:

```
#create a security profile
/interface wifi security
add authentication-types=wpa3-psk name=sec1 passphrase=HaveAg00dDay

#create configuraiton profiles to use for provisioning
/interface wifi configuration
add country=Latvia name=5ghz security=sec1 ssid=CAPsMAN_5
add name=2ghz security=sec1 ssid=CAPsMAN2
add country=Latvia name=5ghz_v security=sec1 ssid=CAPsMAN5_v

#configure provisioning rules, configure band matching as needed
/interface wifi provisioning
add action=create-dynamic-enabled master-configuration=5ghz slave-configurations=5ghz_v supported-bands=\
    5ghz-n
add action=create-dynamic-enabled master-configuration=2ghz supported-bands=2ghz-n

#enable CAPsMAN service
/interface wifi capsman
set ca-certificate=auto enabled=yes
```

CAP:

```
#enable CAP service, in this case CAPsMAN is on same LAN, but you can also specify "caps-man-addresses=x.x.x.x"
here
/interface/wifi/cap set enabled=yes

#set configuration.manager= on the WiFi interface that should act as CAP
/interface/wifi/set wifi1,wifi2 configuration.manager=capsman-or-local
```

If the CAP is hAP ax^2 or hAP ax^3, it is strongly recommended to enable RSTP in the bridge configuration, on the CAP

configuration.manager should only be set on the CAP device itself, don't pass it to the CAP or configuration profile that you provision.

The interface that should act as CAP needs additional configuration under "interface/wifi/set wifiX configuration.manager="

CAPsMAN - CAP VLAN configuration example:

In this example, we will assign VLAN10 to our main SSID, and will add VLAN20 for the guest network, ether5 from CAPsMAN is connected to CAP.

CAPs using "wifi-qcom" package can get "vlan-id" via Datapath from CAPsMAN, CAPs using "wifi-qcom-ac" package will need to use the configuration provided at the end of this example.

CAPsMAN:

```
/interface bridge
add name=br vlan-filtering=yes
/interface vlan
add interface=br name=MAIN vlan-id=10
add interface=br name=GUEST vlan-id=20
/interface wifi datapath
add bridge=br name=MAIN vlan-id=10
add bridge=br name=GUEST vlan-id=20
/interface wifi security
add authentication-types=wpa2-psk,wpa3-psk ft=yes ft-over-ds=yes name=Security_MAIN passphrase=HaveAg00dDay
add authentication-types=wpa2-psk,wpa3-psk ft=yes ft-over-ds=yes name=Security_GUEST passphrase=HaveAg00dDay
/interface wifi configuration
add datapath=MAIN name=MAIN security=Security_MAIN ssid=MAIN_Network
add datapath=GUEST name=GUEST security=Security_GUEST ssid=GUEST_Network
/ip pool
add name=dhcp_pool0 ranges=192.168.1.2-192.168.1.254
add name=dhcp_pool1 ranges=192.168.10.2-192.168.10.254
add name=dhcp_pool2 ranges=192.168.20.2-192.168.20.254
/ip dhcp-server
add address-pool=dhcp_pool0 disabled=yes interface=br name=dhcp1
add address-pool=dhcp_pool1 interface=MAIN name=dhcp2
add address-pool=dhcp_pool2 interface=GUEST name=dhcp3
/interface bridge port
add bridge=br interface=ether5
add bridge=br interface=ether4
add bridge=br interface=ether3
add bridge=br interface=ether2
/interface bridge vlan
add bridge=br tagged=br,ether5,ether4,ether3,ether2 vlan-ids=20
add bridge=br tagged=br,ether5,ether4,ether3,ether2 vlan-ids=10
/interface wifi capsman
set enabled=yes interfaces=br
/interface wifi provisioning
add action=create-dynamic-enabled master-configuration=MAIN slave-configurations=GUEST supported-bands=5ghz-ax
add action=create-dynamic-enabled master-configuration=MAIN slave-configurations=GUEST supported-bands=2ghz-ax
/ip address
add address=192.168.1.1/24 interface=br network=192.168.1.0
add address=192.168.10.1/24 interface=MAIN network=192.168.10.0
add address=192.168.20.1/24 interface=GUEST network=192.168.20.0
/ip dhcp-server network
add address=192.168.1.0/24 gateway=192.168.1.1
add address=192.168.10.0/24 gateway=192.168.10.1
add address=192.168.20.0/24 gateway=192.168.20.1
/system identity
set name=cAP_Controller
```

CAP using "wifi-qcom" package:

```
/interface bridge
add name=bridgeLocal
/interface wifi datapath
add bridge=bridgeLocal comment=defconf disabled=no name=capdp
/interface wifi
set [ find default-name=wifi1 ] configuration.manager=capsman datapath=capdp disabled=no
set [ find default-name=wifi2 ] configuration.manager=capsman datapath=capdp disabled=no
/interface bridge port
add bridge=bridgeLocal comment=defconf interface=ether1
add bridge=bridgeLocal comment=defconf interface=ether2
add bridge=bridgeLocal comment=defconf interface=ether3
add bridge=bridgeLocal comment=defconf interface=ether4
add bridge=bridgeLocal comment=defconf interface=ether5
/interface wifi cap
set discovery-interfaces=bridgeLocal enabled=yes slaves-datapath=capdp
/ip dhcp-client
add interface=bridgeLocal disabled=no
```

CAP using "wifi-qcom-ac" package:

```
/interface bridge
add name=bridgeLocal vlan-filtering=yes
/interface wifi
set [ find default-name=wifi1 ] configuration.manager=capsman disabled=no
set [ find default-name=wifi2 ] configuration.manager=capsman disabled=no
add disabled=no  master-interface=wifi1 name=wifi21
add disabled=no  master-interface=wifi2 name=wifi22
/interface bridge port
add bridge=bridgeLocal comment=defconf interface=ether1
add bridge=bridgeLocal comment=defconf interface=ether2
add bridge=bridgeLocal comment=defconf interface=ether3
add bridge=bridgeLocal comment=defconf interface=ether4
add bridge=bridgeLocal comment=defconf interface=ether5
add bridge=bridgeLocal interface=wifi1 pvid=10
add bridge=bridgeLocal interface=wifi21 pvid=20
add bridge=bridgeLocal interface=wifi2 pvid=10
add bridge=bridgeLocal interface=wifi22 pvid=20
/interface bridge vlan
add bridge=bridgeLocal tagged=ether1 untagged=wifi1,wifi2 vlan-ids=10
add bridge=bridgeLocal tagged=ether1 untagged=wifi21,wifi22 vlan-ids=20
/interface wifi cap
set discovery-interfaces=bridgeLocal enabled=yes slaves-static=yes
```

Additionally, the configuration below has to be added to the **CAPsMAN configuration**:

```
/interface wifi datapath
add bridge=br name=DP_AC
/interface wifi configuration
add datapath=DP_AC name=MAIN_AC security=Security_MAIN ssid=MAIN_Network
add datapath=DP_AC name=GUEST_AC security=Security_GUEST ssid=GUEST_Network
/interface wifi provisioning
add action=create-dynamic-enabled master-configuration=MAIN_AC slave-configurations=GUEST_AC supported-
bands=5ghz-ac
add action=create-dynamic-enabled master-configuration=MAIN_AC slave-configurations=GUEST_AC supported-
bands=2ghz-n
```

Passing datapaths "MAIN/GUEST" from the start of the example to "wifi-qcom-ac" CAP would be misconfiguration, make sure to use datapath without "vlan-id" specified to such devices.

Advanced examples

Enterprise wireless security with User Manager v5

Replacing 'wireless' package

Some MikroTik Wi-Fi 5 APs, which ship with their interfaces managed by the 'wireless' menu, can install the additional 'wifi-qcom-ac' package to make their interfaces compatible with the 'wifi' menu instead.

To do this, it is necessary to uninstall the 'wireless' package, then install 'wifi-qcom-**ac**'.

Compatibility

The wifi-qcom-**ac** package includes alternative drivers for IPQ4018/4019 and QCA9984 radios that make them compatible with the WiFi configuration menu. For possible, wifi-qcom-ac/wifi-qcom/wireless, package combinations, please see the package types section here.

As a rule of thumb, the package is compatible with 802.11ac products, which have an ARM CPU. It is NOT compatible with any of our 802.11ac products which have a MIPS CPU.

Compatibility	Devices
Compatible	Audience, Audience LTE kit, Chateau (all variants of D53), hAP ac^2, hAP ac^3, cAP ac, cAP XL ac, LDF 5 ac, LHG XL 5 ac, LHG XL 52 ac, NetMetal ac^2, mANTBox 52 15s, wAP ac (RBwAPG-5HacD2HnD), SXTsq 5 ac
Incompatible	RB4011iGS+5HacQ2HnD-IN (no support for the 2.4GHz interface), Cube 60Pro ac (no support for 60GHz interface), wAP ac (RBwAPG-5HacT2HnD) and **all other devices with a MIPSBE CPU**

Benefits

- WPA3 authentication and OWE (opportunistic wireless encryption)
- 802.11w standard management frame protection
- 802.11r/k/v
- MU-MIMO and beamforming
- 400Mb/s maximum data rate in the 2.4GHz band for IPQ4019 interfaces

These benefits apply both to the wifi-qcom and wifi-qcom-ac packages.

Lost features

The following notable features are lost when running 802.11ac products with drivers that are compatible with the 'wifi' management interface

- Nstreme and Nv2 wireless protocols
- VLAN configuration in the wireless settings (Per-interface VLANs can be configured in bridge settings)
- Compatibility with station-bridging as implemented in the 'wireless' package, station-bridge only works between the same type of drivers. Wifi to Wifi, and Wireless to Wireless.

Property Reference

AAA properties

Properties in this category configure an access point's interaction with AAA (RADIUS) servers.

Certain parameters in the table below take *format-string* as their value. In a *format-string*, certain characters are interpreted in the following way:

Character	Interpretation

a	Hexadecimal character making up the MAC address of the client device in lowercase
A	Hexadecimal character making up the MAC address of the client device in upper case
i	Hexadecimal character making up the MAC address of the AP's interface in lowercase
I (capital 'i')	Hexadecimal character making up the MAC address of the AP's interface in upper case
N	The entire name of the AP's interface (e.g. 'wifi1')
S	The entire SSID

All other characters are used without interpreting them in any way. For examples, see default values.

Property	Description
called-format (*format-string*)	Format for the value of the Called-Station-Id RADIUS attribute, in AP's messages to RADIUS servers. Default: II-II-II-II-II-II:S
calling-format (*format-string*)	Format for the value of the Calling-Station-Id RADIUS attribute, in AP's messages to RADIUS servers. Default: AA-AA-AA-AA-AA-AA
interim-update (*time interval)*	Interval at which to send interim updates about traffic accounting to the RADIUS server. Default: 5m
mac-caching (*time interval* | *'disabled'*)	Length of time to cache RADIUS server replies, when MAC address authentication is enabled. This resolves issues with client device authentication timing out due to (comparatively high latency of RADIUS server replies. Default value: disabled.
name (*string*)	A unique name for the AAA profile. No default value.
nas-identifier (*string*)	Value of the NAS-Identifier attribute, in AP's messages to RADIUS servers. Defaults to the host name of the device (/system/identity).
password-format (*format-string*)	Format for value to use in calculating the value of the User-Password attribute in AP's messages to RADIUS servers when performing MAC address authentication. Default value: "" (an empty string).
username-format (*format-string*)	Format for the value of the User-Name attribute in APs messages to RADIUS servers when performing MAC address authentication. Default value : AA:AA:AA:AA:AA:AA

Channel properties

Properties in this category specify the desired radio channel.

Property	Description
band (*2ghz-g* | *2ghz-n* | *2ghz-ax* | *5ghz-a* | *5ghz-ac* | *5ghz-an* | *5ghz-ax*)	Frequency band and wireless standard that will be used by the AP. Defaults to newest supported standard. **Note that band support is limited by radio capabilities.**

frequency (*list of integers or integer ranges*)	For an interface in AP mode, specifies frequencies (in MHz) to consider when picking control channel center frequency. For an interface in station mode, specifies frequencies on which to scan for APs. Leave unset (default) to consider all frequencies supported by the radio and permitted by the applicable regulatory profille. The parameter can contain 1 or more comma-separated values of integers or, optionally, ranges of integers denoted using the syntax RangeBeginning-RangeEnd:RangeStep Examples of valid channel.frequency values: • 2412 • 2412,2432,2472 • 5180-5240:20,5500-5580:20
secondary-frequency (*list of integers* | 'disabled')	Frequency (in MHz) to use for the center of the secondary part of a split 80+80MHz channel. Only official 80MHz channels (5210, 5290, 5530, 5610, 5690, 5775) are supported. Leave unset (default) for automatic selection of secondary channel frequency.
skip-dfs-channels (*10min-cac* | *all* | *disabled*)	Whether to avoid using channels, on which channel availability check (listening for presence of radar signals) is required. • *10min-cac* - interface will avoid using channels, on which 10 minute long CAC is required • *all* - interface will avoid using all channels, on which CAC is required • *disabled* (default) - interface may select any supported channel, regardless of CAC requirements
width (*20mhz* | *20/40mhz* | *20/40mhz-Ce* | *20/40mhz-eC* | *20/40/80mhz* | *20/40/80+80mhz* | *20/40/80/160mhz*)	Width of radio channel. Defaults to widest channel supported by the radio hardware.
reselect-interval (time interval)	Specifies when the interface should rescan channel availability and select the most appropriate one to use. Specifying intervall will allow the system to select this interval dynamically and randomly. This helps to avoid a situation when many APs at the same time scan network, select the same channel and prefer to use it at the same time.

Configuration properties

This section includes properties relating to the operation of the interface and the associated radio.

Property	Description
antenna-gain (*integer 0..30*)	Overrides the default antenna gain. The *master* interface of each radio sets the antenna gain for every interface which uses the same radio. This setting cannot override the antenna gain to be lower than the minimum antenna gain of a radio. No default value.
beacon-interval (*time interval 100ms..1s*)	Interval between beacon frames of an AP. Default: 100ms. ⓘ The 802.11 standard defines beacon interval in terms of *time units* (1 TU = 1.024 ms). The actual interval between beacons will be 1 TU for every 1 ms configured. ⚠ Every AP running on the same radio (i.e. a master AP and all its 'virtual'/'slave' APs) must use the same beacon interval.

chains (*list of integer 0..7*)	Radio chains to use for receiving signals. Defaults to all chains available to the corresponding radio hardware.
country (*name of a country*)	Determines, which regulatory domain restrictions are applied to an interface. Defaults to "Latvia". ⚠ It is important to set this value correctly to comply with local regulations and ensure interoperability with other devices.
distance ()	Maximum link distance in kilometers, needs to be set for long-range outdoor links. The value should reflect the distance to the AP or station that is furthest from the device. Unconfigured value allows usage of 3KM links. ⚠ `distance` is not used by the wifi-qcom-ac package. Setting `distance` above the actual needed value can have detrimental effects on throughput and latency.
dtim-period (*integer 1..255*)	Period at which to transmit multicast traffic, when there are client devices in power save mode connected to the AP. Expressed as a multiple of the beacon interval. Higher values enable client devices to save more energy, but increase network latency. Default: 1
hide-ssid (*no* | *yes*)	• *yes* - AP does not include its SSID in beacon frames, and does not reply to probe requests that have broadcast SSID. • *no* - AP includes its SSID in the beacon frames, and replies to probe requests that have broadcast SSID. Default: no
manager (*capsman* | *capsman-or-local* | *local***)**	capsman - the interface will act as CAP only, this option should **not** be passed via provisioning rules to the CAP capsman-or-local - the interface will get configuration via CAPsMAN or use its own, if /interface/wifi/cap is not enabled. local - interface won't contact CAPsMAN in order to get configuration. Default: local
mode (*ap* | *station*)	Interface operation mode • *ap* (default) - interface operates as an access point • *station* - interface acts as a client device, scanning for access points advertising the configured SSID • station-bridge - interface acts as a client device and enables support for a 4-address frame format, so that the interface can be used as a bridge port ⓘ The station-bridge mode, as implemented for 'wifi' intefaces, is incompatible with APs running the older 'wireless' package and vice versa.
multicast-enhance (*enabled* | *disabled*)	With the multicast-enhance feature enabled, an AP will convert every multicast-addressed IP or IPv6 packet into multiple unicast-addressed frames for each connected station. This may improve link throughput and reliability since, unlike multicast frames, unicasts are acknowledged by stations and transmitted using a higher data rate. Default: disabled

qos-classifier (*dscp-high-3-bits* | *priority*)	• dscp-high-3-bits - interface will transmit data packets using a WMM priority equal to the value of the 3 most significant bits of the IP DSCP field • priority - interface will transmit data packets using a WMM priority equal to that set by IP firewall or bridge filter Default: priority 802.11ac wireless chipsets do not support the dscp-high-3-bits classifier mode. For 802.11ac interfaces, please see DSCP from priority.
ssid (*string*)	The name of the wireless network, aka the (E)SSID. No default value.
tx-chains (*list of integer 0..7*)	Radio chains to use for transmitting signals. Defaults to all chains available to the corresponding radio hardware.
tx-power (*integer 0..40*)	A limit on the transmit power (in dBm) of the interface. Can not be used to set power above limits imposed by the regulatory profile. Unset by default.

Datapath properties

Parameters relating to forwarding packets to and from wireless client devices.

Property	Description
bridge (*bridge interface*)	Bridge interface to add interface to, as a bridge port. Virtual ('slave') interfaces are by default added to the same bridge, if any, as the corresponding master interface. Master interfaces are not by default added to any bridge.
bridge-cost (*integer*)	Bridge port cost to use when adding as bridge port. Default: 10
bridge-horizon (*none* | *integer)*	Bridge horizon to use when adding as bridge port Default: none.
client-isolation (*no* | *yes*)	Determines whether client devices connecting to this interface are (by default) isolated from others or not. This policy can be overridden on a per-client basis using access list rules, so a an AP can have a mixture of isolated and non-isolated clients. Traffic from an isolated client will not be forwarded to other clients and unicast traffic from a non-isolated client will not be forwarded to an isolated one. Default: no
interface-list (*interface list*)	List to which add the interface as a member. No default value.
vlan-id (*none* | *integer* 1..4095)	Default VLAN ID to assign to client devices connecting to this interface (only relevant to interfaces in AP mode). When a client is assigned a VLAN ID, traffic coming from the client is automatically tagged with the ID and only packets tagged with with this ID are forwarded to the client. Default: none 802.11ac chipsets do not support this type of VLAN tagging , but they can be configured as VLAN access ports in bridge settings.

Security Properties

Parameters relating to authentication.

Property	Description

authentication-types (*list of wpa-psk, wpa2-psk, wpa-eap, wpa2-eap, wpa3-psk, owe, wpa3-eap, wpa3-eap-192*)	Authentication types to enable on the interface. The default value is an empty list (no authentication, an open network). Configuring a passphrase adds to the default list the *wpa2-psk* authentication method (if the interface is an AP) or both *wpa-psk* and *wpa2-psk* (if the interface is a station). Configuring an *eap-username* and an *eap-password* adds to the default list *wpa-eap and wpa2-eap* authentication methods.
connect-group (*string*)	APs within the same connect group do not allow more than 1 client device with the same MAC address. This is to prevent malicious authorized users from intercepting traffic intended to other users ('MacStealer' attack) or performing a denial of service attack by spoofing the MAC address of a victim. Handling of new connections with duplicate MAC addresses depends on the connect-priority of AP interfaces involved. By default, all APs are assigned the same connect-group.

connect-priority (accept-priority/hold-priority (*integers*))	These parameters determine, how a connection is handled if the MAC address of the client device is the same as that of another active connection to another AP.
	If (accept-priority of AP2) **<** (hold-priority of AP1), a connection to AP2 wil cause the client to be dropped from AP1.
	If (accept-priority of AP2) **=** (hold-priority of AP1), a connection to AP2 will be allowed only if the MAC address can no longer be reached via AP1.
	If (accept-priority of AP2) **>** (hold-priority of AP1), a connection to AP2 will not be accepted.
	
	If omitted, hold-priority is the same as accept-priority.
	By default, APs, which perform user authentication, have higher priority (lower integer value), than open APs.

dh-groups (*list of 19, 20, 21*)	Identifiers of elliptic curve cryptography groups to use in SAE (WPA3) authentication.
disable-pmkid (*no* | *yes*)	For interfaces in AP mode, disables inclusion of a PMKID in EAPOL frames. Disabling PMKID can cause compatibility issues with client devices that make use of it. • *yes* - Do not include PMKID in EAPOL frames. • *no* (default) - include PMKID in EAPOL frames.
eap-accounting (*no* | *yes*)	Send accounting information to RADIUS server for EAP-authenticated peers. Default: no.
⚠ Properties related to EAP, are only relevant to interfaces in station mode. APs delegate (passthrough) EAP authentication to the RADIUS server.	
eap-anonymous-identity (*string*)	Optional anonymous identity for EAP outer authentication. No default value.
eap-certificate-mode (*dont-verify-certificate* | *no-certificates* | *verify-certificate* | *verify-certificate-with-crl*)	Policy for handling the TLS certificate of the RADIUS server. • verify-certificate - require server to have a valid certificate. Check that it is signed by a trusted certificate authority. • dont-verify-certificate (default) - Do not perform any checks on the certificate. • no-certificates - Attempt to establish the TLS tunnel by performing anonymous Diffie-Hellman key exchange. To be used if the RADIUS server has no certificate at all. • verify-certificate-with-crl - Same as *verify-certificate,* but also checks if the certificate is valid by checking the Certificate Revocation List.
eap-methods (*list of peap, tls, ttls*)	EAP methods to consider for authentication. Defaults to all supported methods.
eap-password (*string*)	Password to use, when the chosen EAP method requires one. No default value.
eap-tls-certificate (*certificate*)	Name or id of a certificate in the device's certificate store to use, when the chosen EAP authentication method requires one. No default value.
eap-username (*string*)	Username to use when the chosen EAP method requires one. No default value.

Take care when configuring encryption ciphers.

All client devices MUST support the group encryption cipher used by the AP to connect, and some client devices (notably, Intel® 8260) will also fail to connect if the list of unicast ciphers includes any they don't support.

encryption (*list of ccmp, ccmp-256, gcmp, gcmp-256, tkip*)	A list of ciphers to support for encrypting unicast traffic. Defaults to *ccmp*.

Properties related to 802.11r fast BSS transition only apply to interfaces in AP mode. WiFi interfaces in station mode do not support 802.11r.

For a client device to successfully roam between 2 APs, the APs need to be managed by the same instance of RouterOS. For information on how to centrally manage multiple APs, see CAPsMAN

ft (*no* | *yes*)	Whether to enable 802.11r fast BSS transitions (roaming). Default: no.
ft-mobility-domain (*integer 0..65535*)	The fast BSS transition mobility domain ID. Default: 44484 (0xADC4).
ft-nas-identifier (string of *2..96 hex characters*)	Fast BSS transition PMK-R0 key holder identifier. Default: MAC address of the interface.
ft-over-ds (*no* | *yes*)	Whether to enable fast BSS transitions over DS (distributed system). Default: no.
ft-preserve-vlanid (*no* | *yes*)	• no - when a client connects to this AP via 802.11r fast BSS transition, it is assigned a VLAN ID according to the access and/or interface settings • yes (default) - when a client connects to this AP via 802.11r fast BSS transition, it retains the VLAN ID, which it was assigned during initial authentication The default behavior is essential when relying on a RADIUS server to assign VLAN IDs to users, since a RADIUS server is only used for initial authentication.
ft-r0-key-lifetime (*time interval 1s.. 6w3d12h15m*)	Lifetime of the fast BSS transition PMK-R0 encryption key. Default: 600000s (~7 days)
ft-reassociation-deadline (*time interval 0..70s*)	Fast BSS transition reassociation deadline. Default: 20s.
group-encryption (*ccmp* | *ccmp-256* | *gcmp* | *gcmp-256* | *tkip*)	Cipher to use for encrypting multicast traffic. Defaults to *ccmp*.
group-key-update (*time interval*)	Interval at which the group temporal key (key for encrypting broadcast traffic) is renewed. Defaults to 24 hours.
management-encryption (*cmac* | *cmac-256* | *gmac* | *gmac-256*)	Cipher to use for encrypting protected management frames. Defaults to *cmac*.
management-protection (*allowed* | *disabled* | *required*)	Whether to use 802.11w management frame protection. **Incompatible with management frame protection in standard wireless package**. The default value depends on the value of the selected authentication type. WPA2 allows the use of management protection, WPA3 requires it.
owe-transition-interface (*interface*)	Name or internal id of an interface whose MAC address and SSID to advertise as the matching AP when running in OWE transition mode. Required for setting up open APs that offer OWE, but also work with older devices that don't support the standard. See configuration example below.
passphrase (*string of up to 63 characters*)	The passphrase to use for PSK authentication types. Defaults to an empty string - "". WPA-PSK and WPA2-PSK authentication requires a minimum of 8 chars, while WPA3-PSK does not have a minimum passphrase length.

sae-anti-clogging-threshold (*'disabled'* | *integer*)	Due to SAE (WPA3) associations being CPU resource intensive, overwhelming an AP with bogus authentication requests makes for a feasible denial-of-service attack. This parameter provides a way to mitigate such attacks by specifying a threshold of in-progress SAE authentications, at which the AP will start requesting that client devices include a cookie bound to their MAC address in their authentication requests. It will then only process authentication requests that contain valid cookies. Default: 5.
sae-max-failure-rate (*'disabled'* | *integer*)	Rate of failed SAE (WPA3) associations per minute, at which the AP will stop processing new association requests. Default: 40.
sae-pwe (*both* | *hash-to-element* | *hunting-and-pecking*)	Methods to support for deriving SAE password element. Default: both.
wps (*disabled* | *push-button*)	• *push-button* (default) - AP will accept WPS authentication for 2 minutes after 'wps-push-button' command is called. Physical WPS button functionality not yet implemented. • *disabled* - AP will not accept WPS authentication

Steering properties

Properties in this category govern mechanisms for advertising potential roaming candidates to client devices.

Property	Description
neighbor-group (*string*)	When sending neighbor reports and BSS transition management requests, an AP will list all other APs within its neighbor group as potential roaming candidates. By default, a dynamic neighbor group is created for each set of APs with the same SSID and authentication settings. APs operating in the 5GHz band are indicated to be preferable to ones operating in the 2.4GHz band.
rrm (*no* | *yes*)	Enables sending of 802.11k neighbor reports. Default: yes
wnm (*no* | *yes*)	Enables sending of solicited 802.11v BSS transition management requests. Default: yes

Miscellaneous properties

Property	Description
arp (*disabled* | *enabled* | *local-proxy-arp* | *proxy-arp* | *reply-only*)	Address Resolution Protocol mode: • *disabled* - the interface will not use ARP • *enabled* - the interface will use ARP (default) • *local-proxy-arp* - the router performs proxy ARP on the interface and sends replies to the same interface • *proxy-arp* - the router performs proxy ARP on the interface and sends replies to other interfaces • *reply-only* - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the ARP table. No dynamic entries will be automatically stored in the ARP table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*time interval* | *'auto'*)	Determines how long a dynamically added ARP table entry is considered valid since the last packet was received from the respective IP address. Value *auto* equals to the value of *arp-timeout* in */ip settings*, which defaults to 30s.
disable-running-check (*no* | *yes*)	• *yes* - interface's *running* property will be true whenever the interface is not disabled • *no* (default) - interface's *running* property will only be true when it has established a link to another device
disabled (*no* | *yes*) (X)	Hardware interfaces are disabled by default. Virtual interfaces are not.

mac-address (*MAC*)	MAC address (BSSID) to use for an interface. Hardware interfaces default to the MAC address of the associated radio interface. Default MAC addresses for virtual interfaces are generated by 1. Taking the MAC address of the associated master interface 2. Setting the second-least-significant bit of the first octet to 1, resulting in a locally administered MAC address 3. If needed, increment the last octet of the address to ensure it doesn't overlap with the address of another interface on the device
mtu (*integer [32..2290]*; Default: **1500**)	Layer 3 Maximum transmission unit.
l2mtu (*integer [32..2290]*; Default: **2290**)	Layer 2 Maximum transmission unit.
master-interface (*interface*)	Multiple interface configurations can be run simultaneously on every wireless radio. Only one of them determines the radio's state (whether it is enabled, what frequency it's using, etc). This 'master' interface, is *bound* to a radio with the corresponding *radio-mac*. To create additional ('virtual') interface configurations on a radio, they need to be *bound* to the corresponding master interface. No default value.
name (*string*)	A name for the interface. Defaults to *wifiN*, where *N* is the lowest integer that has not yet been used for naming an interface.

Read-only properties

Property	Description
bound (*boolean*) (B)	True for *master* interfaces that are currently available for WiFi manager. True for a virtual interface (configurations linked to a master interface) when both the interface itself and its master interface are not disabled and the *master* interface has a bound flag.
default-name (*string*)	The default name for an interface.
inactive (*boolean*) (I)	False for interfaces in AP mode when they've selected a channel for operation (i.e. configuration has been successfully applied). False for interfaces in station mode when they've connected to an AP (i.e. configuration has been successfully applied, and an AP with matching settings has been found). True otherwise.
master (*boolean*) (M)	True for physical interfaces on the router itself or detected CAP if running as CAPsMAN. False for virtual interfaces.
radio-mac (*MAC*)	The MAC address of the associated radio.
running (*boolean*) (R)	True, when an interface has established a link to another device. If *disable-running-check* is set to 'yes', true whenever the interface is not disabled.

Access List

Filtering parameters	
Parameter	**Description**

interface (*interface* | *interface-list* | *'any'*)	Match if connection takes place on the specified interface or interface belonging to a specified list. Default: any.
mac-address (*MAC address*)	Match if the client device has the specified MAC address. No default value.
mac-address-mask (*MAC address*)	Modifies the **mac-address** parameter to match if it is equal to the result of performing bit-wise AND operation on the client MAC address and the given address mask. Default: FF:FF:FF:FF:FF:FF (i.e. client's MAC address must match value of **mac-address** exactly)
signal-range (*min..max*)	Match if the strength of the received signal from the client device is within the given range. Default: '-120..120'
ssid-regexp (*regex*)	Match if the given regular expression matches the SSID.
time (*start-end,days*)	Match during the specified time of day and (optionally) days of week. Default: 0s-1d

Action parameters	
Parameter	**Description**
allow-signal-out-of-range (*time period* | 'always')	The length of time which a connected peer's signal strength is allowed to be outside the range required by the **signal-range** parameter, before it is disconnected. If the value is set to 'always', peer signal strength is only checked during association. Default: 0s.
action (*accept* | *reject* | *query-radius*)	Whether to authorize a connection • *accept* - connection is allowed • *reject* - connection is not allowed • *query-radius* - connection is allowed if MAC address authentication of the client's MAC address succeeds Default: *accept*
client-isolation (*no* | *yes*)	Whether to isolate the client from others connected to the same AP. No default value.
passphrase (*string*)	Override the default passphrase with given value. No default value.
radius-accounting (*no* | *yes*)	Override the default RADIUS accounting policy with given value. No default value.
vlan-id (*none* | *integer 1..4095*)	Assign the given VLAN ID to matched clients. No default value.

Frequency scan

Information about RF conditions on available channels can be obtained by running the frequency-scan command.

Command parameters	
Parameter	**Description**
duration (*time interval)*	Length of time to perform the scan for before exiting. Useful for non-interactive use. Not set by default.
freeze-frame-interval (*time interval)*	Time interval at which to update command output. Default: 1s.
frequency (*list of frequencies /ranges)*	Frequencies to perform the scan on. See channel.frequency parameter syntax above for more detail. Defaults to all supported frequencies.
number (*string)*	Either the name or internal id of the interface to perform the scan with. Required. Not set by default.
rounds (*integer)*	Number of times to go through list of scannable frequencies before exiting. Useful for non-interactive use. Not set by default.
save-file (string)	Name of file to save output to. Not set by default.

Output parameters	
Parameter	**Description**
channel (*integer)*	Frequency (in MHz) of the channel scanned.
networks (*integer)*	Number of access points detected on the channel.
load (integer*)*	Percentage of time the channel was busy during the scan.
nf (integer)	Noise floor (in dBm) of the channel.
max-signal (*integer*)	Maximum signal strength (in dBm) of APs detected in the channel.
min-signal (*integer*)	Minimum signal strength (in dBm) of APs detected in the channel.
primary (*boolean*) (P)	Channel is in use as the primary (control) channel by an AP.
secondary (boolean) (S)	Channel is in use as a secondary (extension) channel by an AP.

Flat-snoop

The '/interface wifi flat-snoop' is a tool for surveying APs and stations. Monitors frequency usage, and displays which devices occupy each frequency. Provides more detailed infromation regarding nearby APs than scan, and offers easy overview of frequency usage by station/AP count.

Output parameters	
Parameter	**Description**
duration (*time interval*)	Length of time to perform the scan before exiting. Useful for non-interactive use. Not set by default.
filter-type (*bsss* |*frequency* |*stas*)	bsss - list of active APs and their parameters. frequency - list of station and AP count per scanned frequency stas - a detailed list of stations on each scanned frequency If filter-type is unspecified all types will be returned.
freeze-frame-interval (*time interval)*	Time interval at which to update command output. Default: 1s.

Scan command

The '/interface wifi scan' command will scan for access points and print out information about any APs it detects.

The scan command takes all the same parameters as the frequency-scan command.

Output parameters	
Parameter	**Description**
active (*boolean*) (A)	This signifies that beacons from the AP have been received in the last 30 seconds.
address (*MAC*)	The MAC address (BSSID) of the AP.
channel (*string*)	The control channel frequency used by the AP, its supported wireless standards and control/extension channel layout.
security (*string*)	Authentication methods supported by the AP.
signal (*integer*)	The signal strength of the AP's beacons (in dBm).
ssid (*string*)	The extended service set identifier of the AP.

sta-count (*integer*)	The number of client devices associated with the AP. Only available if the AP includes this information in its beacons.

Sniffer

Command parameters	
Parameter	**Description**
duration (*time interval*)	Automatically interrupt the sniffer after the specified time has passed. No default value.
filter (*string*)	A string that specifies a filter to apply to captured frames. Only frames matched by the filter expression will be displayed, saved or streamed. This works similarly to filter strings in libpcap, for example. The filter can match • Address fields (addr1, addr2, addr3) • Wireless frame type and subtype, including shortcuts such as 'beacon' (type == 0 && subtype == 8) • Flags (to-ds, from-ds, retry, power, protected) A string can include the following operators: • == (exact match) • != (does not equal) • && (logical AND) • || (logical OR) • () (for grouping filter expressions)
number (*interface*)	Interface to use for sniffing.
pcap-file (*string)*	Save captured frames to a file with the given name. No default value (captured frames are not saved to a file by default).
pcap-size-limit (*integer*)	File size limit (in bytes) when storing captured frames locally. When this limit has been reached, no new frames are added to the capture file. No default value.
stream-address (IP address)	Stream captured packets via the TZSP protocol to the given address. No default value (captured packets are not streamed anywhere by default).
stream-rate (*integer*)	Limit the rate (in packets per second) at which captured frames are streamed via TZSP.

WPS

interface/wifi/wps-client wifi

Command parameters	
Parameter	**Description**
duration (*time interval*)	Length of time after which the command will time out if no AP is found. Unlimited by default.
interval (*time interval*)	Time interval at which to update command output. Default: 1s.
mac-address (*MAC*)	Only attempt connecting to AP with the specified MAC (BSSID). Not set by default.
number (*string*)	Name or internal id of the interface with which to attempt a connection. Not set by default.
ssid (*string*)	Only attempt to connect to APs with the specified SSID. Not set by default.

Radios

Information about the capabilities of each radio can be gained by running the `/interface/wifi/radio print detail` command.

Property	Description
2g-channels (*list of integers*)	Frequencies supported in the 2.4GHz band.
5g-channels (*list of integers*)	Frequencies supported in the 5GHz band.
bands (*list of strings*)	Supported frequency bands, wireless standards, and channel widths.
ciphers (*list of strings*)	Supported encryption ciphers.
countries (*list of strings*)	Regulatory domains supported by the interface.
hw-caps (*list of strings*)	Additional supported features (e.g. sniffer, qos-classifier-dscp).
hw-type (*string*)	Radio hardware model number.
max-interfaces (*integer*)	Maximum number of logical interfaces.
max-peers (*integer*)	Maximum number of associated peers (connected stations).
max-station-interfaces (*integer*)	Maximum number of logical interfaces in station mode.
max-vlans (*integer*)	Maximum number of different per-user VLANs.
min-antenna-gain (*integer*)	Minimum antenna gain permitted for the interface.
phy-id (*string*)	A unique identifier.
radio-mac (*MAC*)	MAC address of the radio interface. Can be used to match radios to interface configurations.
rx-chains (*list of integers*)	IDs for radio chains available for receiving radio signals.
tx-chains (*list of integers*)	IDs for radio chains available for transmitting radio signals.

Registration table

The registration table contains read-only information about associated wireless devices.

Parameter	Description
authorized (*boolean*) (A)	True when the peer has successfully authenticated.
bytes (*list of integers*)	Number of bytes in packets transmitted to a peer and received from it.
interface (*string*)	Name of the interface, which was used to associate with the peer.
mac-address (*MAC*)	The MAC address of the peer.
packets (*list of integers*)	Number of packets transmitted to a peer and received from it.
rx-rate *(string)*	Bitrate of received transmissions from peer.
signal (*integer)*	Strength of signal received from the peer (in dBm).
tx-rate (*string)*	Bitrate used for transmitting to the peer.
uptime (*time interval)*	Time since association.

CAPsMAN Global Configuration

Menu: /interface/wifi/capsman

Property	Description
ca-certificate *(auto | certificate name)*	Device CA certificate, CAPsMAN server requires a certificate, certificate on CAP is optional.

certificate (*auto* | *certificate name* | *none*; Default: **none**)	Device certificate
enabled *(no | yes*)	Disable or enable CAPsMAN functionality
package-path (*string* |; Default:)	Folder location for the RouterOS packages. For example, use "/upgrade" to specify the upgrade folder from the files section. If an empty string is set, CAPsMAN can use built-in RouterOS packages, note that in this case only CAPs with the same architecture as CAPsMAN will be upgraded.
require-peer-certificate (*yes* | *no*; Default: **no**)	Require all connecting CAPs to have a valid certificate
upgrade-policy (*none* | *require-same-version* | *suggest-same-upgrade*; Default: **none**)	Upgrade policy options • none - do not perform upgrade • require-same-version - CAPsMAN suggests to upgrade the CAP RouterOS version and, if it fails it will not provision the CAP. (Manual provision is still possible) • suggest-same-version - CAPsMAN suggests to upgrade the CAP RouterOS version and if it fails it will still be provisioned
interfaces *(all | interface name | none; Default:* ***all****)*	Interfaces on which CAPsMAN will listen for CAP connections

CAPsMAN Provisioning

Provisioning rules for matching radios are configured in **/interface/wifi/provisioning/** menu:

Property	Description
action (*create-disabled* | *create-enabled* | *create-dynamic-enabled* | *none*; Default: **none**)	Action to take if rule matches are specified by the following settings: • **create-disabled** - create disabled static interfaces for radio. I.e., the interfaces will be bound to the radio, but the radio will not be operational until the interface is manually enabled; • **create-enabled** - create enabled static interfaces. I.e., the interfaces will be bound to the radio and the radio will be operational; • **create-dynamic-enabled** - create enabled dynamic interfaces. I.e., the interfaces will be bound to the radio, and the radio will be operational; • **none** - do nothing, leaves radio in the non-provisioned state;
comment (*string*; Default:)	Short description of the Provisioning rule
common-name-regexp (*string*; Default:)	Regular expression to match radios by common name. Each CAP's common name identifier can be found under "/interface/wifi/radio" as value "REMOTE-CAP-NAME"
supported-bands (*2ghz-ax* | *2ghz-g* | *2ghz-n* | *5ghz-a* | *5ghz-ac* | *5ghz-ax* | *5ghz-n*; Default:)	Match radios by supported wireless modes.
identity-regexp (*string*; Default:)	Regular expression to match radios by router identity
address-ranges (*IpAddressRange[, IpAddressRanges] max 100x*; Default: **""**)	Match CAPs with IPs within the configured address range. Will only work for CAPs that joined CAPsMAN using IP, not MAC address.
master-configuration (*string*; Default:)	If **action** specifies to create interfaces, then a new master interface with its configuration set to this configuration profile will be created
name-format (*string*)	Base string to use when constructing names of provisioned interfaces. Each new interface will be created by taking the base string and appending a number to the end of it, a number will only be appended if the string is not unique. If included in the string, the character sequence **%I** will be replaced by the system identity of the cAP, **%C** will be replaced with the cAP's TLS certificate's Common Name, **%R**, or **%r** for lowercase, will be replaced with the CAP's radio MAC Default: "cap-wifi"

radio-mac (*MAC address*)	MAC address of radio to be matched. No default value.
slave-configurations (*string*; Default:)	If the **action** specifies to create interfaces, then a new slave interface for each configuration profile in this list is created.
disabled (*yes* | *no*; Default: **no**)	Specifies if the provision rule is disabled.

CAP configuration

Menu: /interface/wifi/cap

Property	Description
caps-man-addresses *(list of IP addresses; Default:* ***empty***)	List of Manager IP addresses that CAP will attempt to contact during discovery
caps-man-names ()	An ordered list of CAPs Manager names that the CAP will connect to, if empty - CAP does not check Manager name
discovery-interfaces (*list of interfaces*;)	List of interfaces over which CAP should attempt to discover Manager
lock-to-caps-man (*no* | *yes;* Default: **no**)	Sets, if CAP should lock to the first CAPsMAN it connects to
slaves-static ()	Creates Static Virtual Interfaces, allows the possibility to assign IP configuration to those interfaces. MAC address is used to remember each static-interface when applying the configuration from the CAPsMAN.
caps-man-certificate-common-names ()	List of Manager certificate CommonNames that CAP will connect to, if empty - CAP does not check Manager certificate CommonName
certificate ()	Certificate to use for authenticating
enabled (*yes* | *no*; Default: **no**)	Disable or enable the CAP feature
current-caps-man-address ()	Shows currently used CAPsMAN address (available since 7.15)
current-caps-man-identity ()	Shows currently used CAPsMAN identity (available since 7.15)
slaves-datapath ()	

Wireless Interface

Overview

Package: wireless

RouterOS wireless complies with IEEE 802.11 standards, it provides complete support for 802.11a, 802.11b, 802.11g, 802.11n and 802.11ac as long as additional features like WPA, WEP, AES encryption, Wireless Distribution System (WDS), Dynamic Frequency selection (DFS), Virtual Access Point, Nstreme and NV2 proprietary protocols and many more. Wireless features compatibility table for different wireless protocols.

Wireless can operate in several modes: client (station), access point, wireless bridge etc. Client/station also can operate in different modes, a complete list of supported modes can be found here.

General interface properties

Sub-menu: `/interface wireless`

Property	Description
adaptive-noise-immunity (*ap-and-client-mode* | *client-mode* | *none*; Default: **none**)	This property is only effective for cards based on Atheros chipset.
allow-sharedkey (*yes* | *no*; Default: **no**)	Allow WEP Shared Key clients to connect. Note that no authentication is done for these clients (WEP Shared keys are not compared to anything) - they are just accepted at once (if access list allows that)
ampdu-priorities (*list of integer [0..7]*; Default: **0**)	Frame priorities for which AMPDU sending (aggregating frames and sending using block acknowledgment) should get negotiated and used. Using AMPDUs will increase throughput, but may increase latency, therefore, may not be desirable for real-time traffic (voice, video). Due to this, by default AMPDUs are enabled only for best-effort traffic.
amsdu-limit (*integer [0..8192]*; Default: **8192**)	Max AMSDU that device is allowed to prepare when negotiated. AMSDU aggregation may significantly increase throughput especially for small frames, but may increase latency in case of packet loss due to retransmission of aggregated frame. Sending and receiving AMSDUs will also increase CPU usage.
amsdu-threshold (*integer [0..8192]*; Default: **8192**)	Max frame size to allow including in AMSDU.
antenna-gain (*integer [0..4294967295]*; Default: **0**)	Antenna gain in dBi, used to calculate maximum transmit power according to **co untry** regulations.
antenna-mode (*ant-a* | *ant-b* | *rxa-txb* | *txa-rxb*; Default:)	Select antenna to use for transmitting and for receiving - *ant-a* - use only 'a' antenna - *ant-b* - use only 'b' antenna - *txa-rxb* - use antenna 'a' for transmitting, antenna 'b' for receiving - *rxa-txb* - use antenna 'b' for transmitting, antenna 'a' for receiving
area (*string*; Default:)	Identifies group of wireless networks. This value is announced by AP, and can be matched in connect-list by **area-prefix**. This is a proprietary extension.
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	`Read more >>`
arp-timeout (*auto* | *integer*; Default: **auto**)	ARP timeout is time how long ARP record is kept in ARP table after no packets are received from IP. Value **auto** equals to the value of **arp-timeout** in **/ip settings** , default is 30s

band (*2ghz-b | 2ghz-b/g | 2ghz-b/g/n | 2ghz-onlyg | 2ghz-onlyn | 5ghz-a | 5ghz-a/n | 5ghz-onlyn | 5ghz-a/n/ac | 5ghz-onlyac | 5ghz-n/ac*; Default:)	Defines set of used data rates, channel frequencies and widths.
basic-rates-a/g (*12Mbps | 18Mbps | 24Mbps | 36Mbps | 48Mbps | 54Mbps | 6Mbps | 9Mbps*; Default: **6Mbps**)	Similar to the **basic-rates-b** property, but used for 5ghz, 5ghz-10mhz, 5ghz-5mhz, 5ghz-turbo, 2.4ghz-b/g, 2.4ghz-onlyg, 2ghz-10mhz, 2ghz-5mhz and 2.4 ghz-g-turbo bands.
basic-rates-b (*11Mbps | 1Mbps | 2Mbps | 5.5Mbps*; Default: **1Mbps**)	List of basic rates, used for 2.4ghz-b, 2.4ghz-b/g and 2.4ghz-onlyg bands. Client will connect to AP only if it supports all basic rates announced by the AP. AP will establish WDS link only if it supports all basic rates of the other AP. This property has effect only in AP modes, and when value of **rate-set** is configured.
bridge-mode (*disabled | enabled*; Default: **enabled**)	Allows to use station-bridge mode. `Read more >>`
burst-time (*integer | disabled*; Default: **disabled**)	Time in microseconds which will be used to send data without stopping. Note that no other wireless cards in that network will be able to transmit data during burst-time microseconds. This setting is available only for AR5000, AR5001X, and AR5001X+ chipset based cards.
channel-width (*20/40/80/160mhz-Ceeeeeee | 20/40/80/160mhz-XXXXXXXX | 20/40/80/160mhz-eCeeeeee | 20/40/80/160mhz-eeCeeeee | 20/40/80/160mhz-eeeCeeee | 20/40/80/160mhz-eeeeCeee | 20/40/80/160mhz-eeeeeCee | 20/40/80/160mhz-eeeeeeCe | 20/40/80/160mhz-eeeeeeeC | 20/40/80mhz-Ceee | 20/40/80mhz-eCee | 20/40/80mhz-eeCe | 20/40/80mhz-eeeC | 20/40/80mhz-XXXX | 20/40mhz-Ce | 20/40mhz-eC | 20/40mhz-XX | 40mhz-turbo | 20mhz | 10mhz | 5mhz*; Default: **20mhz**)	Use of extension channels (e.g. Ce, eC etc) allows additional 20MHz extension channels and if it should be located below or above the control (main) channel. Extension channel allows 802.11n devices to use up to 40MHz (802.11ac up to 160MHz) of spectrum in total thus increasing max throughput. Channel widths with XX and XXXX extensions automatically scan for a less crowded control channel frequency based on the number of concurrent devices running in every frequency and chooses the “C” - Control channel frequency automatically.
comment (*string*; Default:)	Short description of the interface
compression (*yes | no*; Default: **no**)	Setting this property to *yes* will allow the use of the hardware compression. Wireless interface must have support for hardware compression. Connections with devices that do not use compression will still work.
country (*name of the country | no_country_set*; Default: **etsi**)	Limits available bands, frequencies and maximum transmit power for each frequency. Also specifies default value of **scan-list**. Value *no_country_set* is an FCC compliant set of channels.
default-ap-tx-limit (*integer [0..4294967295]*; Default: **0**)	This is the value of **ap-tx-limit** for clients that do not match any entry in the access-list. 0 means no limit.
default-authentication (*yes | no*; Default: **yes**)	For AP mode, this is the value of **authentication** for clients that do not match any entry in the access-list. For station mode, this is the value of **connect** for APs that do not match any entry in the connect-list
default-client-tx-limit (*integer [0..4294967295]*; Default: **0**)	This is the value of **client-tx-limit** for clients that do not match any entry in the access-list. 0 means no limit
default-forwarding (*yes | no*; Default: **yes**)	This is the value of **forwarding** for clients that do not match any entry in the access-list
disable-running-check (*yes | no*; Default: **no**)	When set to **yes** interface will always have running flag. If value is set to **no'**, the router determines whether the card is up and running - for AP one or more clients have to be registered to it, for station, it should be connected to an AP.
disabled (*yes | no*; Default: **yes**)	Whether interface is disabled
disconnect-timeout (*time [0s..15s]*; Default: **3s**)	This interval is measured from third sending failure on the lowest data rate. At this point 3 * (**hw-retries** + 1) frame transmits on the lowest data rate had failed. During **disconnect-timeout** packet transmission will be retried with **on-fail-retry-time** interval. If no frame can be transmitted successfully during **disconnect-timeout**, the connection is closed, and this event is logged as "extensive data loss". Successful frame transmission resets this timer.

distance (*integer* | *dynamic* | *indoors*; Default: **dynamic**)	How long to wait for confirmation of unicast frames (**ACKs**) before considering transmission unsuccessful, or in short **ACK-Timeout**. Distance value has these behaviors: • *Dynamic* - causes AP to detect and use the smallest timeout that works with all connected clients. • *Indoor* - uses the default ACK timeout value that the hardware chip manufacturer has set. • *Number* - uses the input value in formula: ACK-timeout = ((**distance** * 1000) + 299) / 300 us; Acknowledgments are not used in Nstreme/NV2 protocols.
frame-lifetime (*integer [0..4294967295]*; Default: **0**)	Discard frames that have been queued for sending longer than **frame-lifetime**. By default, when value of this property is *0*, frames are discarded only after connection is closed.
frequency (*integer [0..4294967295]*; Default:)	Channel frequency value in MHz on which AP will operate. Allowed values depend on the selected band, and are restricted by **country** setting and wireless card capabilities. This setting has **no effect** if interface is in any of **station** modes, or in *wds-slave* mode, or if DFS is active. *Note*: If using mode "superchannel", any frequency supported by the card will be accepted, but on the RouterOS client, any non-standard frequency must be configured in the scan-list, otherwise it will not be scanning in non-standard range. In Winbox, scanlist frequencies are in *bold*, any other frequency means the clients will need scan-list configured.
frequency-mode (*manual-txpower* | *regulatory-domain* | *superchannel*; Default: **regulatory_domain**)	Three frequency modes are available: • *regulatory-domain* - Limit available channels and maximum transmit power for each channel according to the value of **country** • *manual-txpower* - Same as above, but do not limit maximum transmit power. • *superchannel* - Conformance Testing Mode. Allow all channels supported by the card. List of available channels for each band can be seen in **/interface wireless info allowed-channels**. This mode allows you to test wireless channels outside the default scan-list and/or regulatory domain. This mode should only be used in controlled environments, or if you have special permission to use it in your region. Before v4.3 this was called Custom Frequency Upgrade, or Superchannel. Since RouterOS v4.3 this mode is available without special key upgrades to all installations.
frequency-offset (*integer [-2147483648..2147483647]*; Default: **0**)	Allows to specify offset if the used wireless card operates at a different frequency than is shown in RouterOS, in case a frequency converter is used in the card. So if your card works at 4000MHz but RouterOS shows 5000MHz, set offset to 1000MHz and it will be displayed correctly. The value is in MHz and can be positive or negative.
guard-interval (*any* | *long*; Default: **any**)	Whether to allow use of short guard interval (refer to 802.11n MCS specification to see how this may affect throughput). "any" will use either short or long, depending on data rate, "long" will use long.
hide-ssid (*yes* | *no*; Default: **no**)	• *yes* - AP does not include SSID in the beacon frames, and does not reply to probe requests that have broadcast SSID. • *no* - AP includes SSID in the beacon frames, and replies to probe requests that have broadcast SSID. This property has an effect only in AP mode. Setting it to *yes* can remove this network from the list of wireless networks that are shown by some client software. Changing this setting does not improve the security of the wireless network, because SSID is included in other frames sent by the AP.

ht-basic-mcs (*list of (mcs-0 | mcs-1 | mcs-2 | mcs-3 | mcs-4 | mcs-5 | mcs-6 | mcs-7 | mcs-8 | mcs-9 | mcs-10 | mcs-11 | mcs-12 | mcs-13 | mcs-14 | mcs-15 | mcs-16 | mcs-17 | mcs-18 | mcs-19 | mcs-20 | mcs-21 | mcs-22 | mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11n for MCS specification.
ht-supported-mcs (*list of (mcs-0 | mcs-1 | mcs-2 | mcs-3 | mcs-4 | mcs-5 | mcs-6 | mcs-7 | mcs-8 | mcs-9 | mcs-10 | mcs-11 | mcs-12 | mcs-13 | mcs-14 | mcs-15 | mcs-16 | mcs-17 | mcs-18 | mcs-19 | mcs-20 | mcs-21 | mcs-22 | mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7; mcs-8; mcs-9; mcs-10; mcs-11; mcs-12; mcs-13; mcs-14; mcs-15; mcs-16; mcs-17; mcs-18; mcs-19; mcs-20; mcs-21; mcs-22; mcs-23**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11n for MCS specification.
hw-fragmentation-threshold (*integer[256..3000] | disabled*; Default: **0**)	Specifies maximum fragment size in bytes when transmitted over the wireless medium. 802.11 standard packet (MSDU in 802.11 terminologies) fragmentation allows packets to be fragmented before transmitting over a wireless medium to increase the probability of successful transmission (only fragments that did not transmit correctly are retransmitted). Note that transmission of a fragmented packet is less efficient than transmitting unfragmented packet because of protocol overhead and increased resource usage at both - transmitting and receiving party.
hw-protection-mode (*cts-to-self | none | rts-cts*; Default: **none**)	Frame protection support property `read more >>`
hw-protection-threshold (*integer [0..65535]*; Default: **0**)	Frame protection support property `read more >>`
hw-retries (*integer [0..15]*; Default: **7**)	Number of times sending frame is retried without considering it a transmission failure. Data-rate is decreased upon failure and the frame is sent again. Three sequential failures on the lowest supported rate suspend transmission to this destination for the duration of **on-fail-retry-time**. After that, the frame is sent again. The frame is being retransmitted until transmission success, or until the client is disconnected after **disconnect-timeout**. The frame can be discarded during this time if **frame-lifetime** is exceeded.
installation (*any | indoor | outdoor*; Default: **any**)	Adjusts scan-list to use indoor, outdoor or all frequencies for the country that is set.
interworking-profile (*enabled | disabled*; Default: **disabled**)	
keepalive-frames (*enabled | disabled*; Default: **enabled**)	Applies only if wireless interface is in mode=**ap-bridge**. If a client has not communicated for around 20 seconds, AP sends a "keepalive-frame". **Note**, disabling the feature can lead to "ghost" clients in registration-table.
l2mtu (*integer [0..65536]*; Default: **1600**)	
mac-address (*MAC*; Default:)	
master-interface (*string*; Default:)	Name of wireless interface that has *virtual-ap* capability. Virtual AP interface will only work if master interface is in *ap-bridge*, *bridge*, *station* or *wds-slave* mode. This property is only for virtual AP interfaces.
max-station-count (*integer [1..2007]*; Default: **2007**)	Maximum number of associated clients. WDS links also count toward this limit.

mode (*station* | *station-wds* | *ap-bridge* | *bridge* | *alignment-only* | *nstreme-dual-slave* | *wds-slave* | *station-pseudobridge* | *station-pseudobridge-clone* | *station-bridge*; Default: **station**)	Selection between different station and access point (AP) modes. Station modes: • *station* - Basic station mode. Find and connect to acceptable AP. • *station-wds* - Same as *station*, but create WDS link with AP, using proprietary extension. AP configuration has to allow WDS links with this device. Note that this mode does not use entries in wds. • *station-pseudobridge* - Same as *station*, but additionally perform MAC address translation of all traffic. Allows interface to be bridged. • *station-pseudobridge-clone* - Same as *station-pseudobridge*, but use **station-bridge-clone-mac** address to connect to AP. • station-bridge - Provides support for transparent protocol-independent L2 bridging on the station device. RouterOS AP accepts clients in station-bridge mode when enabled using bridge-mode parameter. In this mode, the AP maintains a forwarding table with information on which MAC addresses are reachable over which station device. Only works with RouterOS APs. With station-bridge mode, it is not possible to connect to CAPsMAN controlled CAP. AP modes: • *ap-bridge* - Basic access point mode. • *bridge* - Same as *ap-bridge*, but limited to one associated client. • *wds-slave* - Same as *ap-bridge*, but scan for AP with the same **ssid** and establishes WDS link. If this link is lost or cannot be established, then continue scanning. If **dfs-mode** is *radar-detect*, then APs with enabled **hide-ssid** will not be found during scanning. Special modes: • *alignment-only* - Put the interface in a continuous transmit mode that is used for aiming the remote antenna. • *nstreme-dual-slave* - allow this interface to be used in nstreme-dual setup. MAC address translation in pseudobridge modes works by inspecting packets and building a table of corresponding IP and MAC addresses. All packets are sent to AP with the MAC address used by pseudobridge, and MAC addresses of received packets are restored from the address translation table. There is a single entry in the address translation table for all non-IP packets, hence more than one host in the bridged network cannot reliably use non-IP protocols. Note: Currently IPv6 doesn't work over Pseudobridge
mtu (*integer [0..65536]*; Default: **1500**)	
multicast-buffering (*disabled* | *enabled*; Default: **enabled**)	For a client that has power saving, buffer multicast packets until next beacon time. A client should wake up to receive a beacon, by receiving beacon it sees that there are multicast packets pending, and it should wait for multicast packets to be sent.
multicast-helper (*default* | *disabled* | *full*; Default: **default**)	When set to **full**, multicast packets will be sent with a unicast destination MAC address, resolving multicast problem on the wireless link. This option should be enabled only on the access point, clients should be configured in **station-bridge** mode. Available starting from v5.15. • disabled - disables the helper and sends multicast packets with multicast destination MAC addresses • dhcp - dhcp packet mac addresses are changed to unicast mac addresses prior to sending them out • full - all multicast packet mac address are changed to unicast mac addresses prior to sending them out • default - default choice that currently is set to *dhcp*. Value can be changed in future releases.
name (*string*; Default:)	name of the interface

noise-floor-threshold (*default* | *integer [-128..127]*; Default: **default**)	For advanced use only, as it can badly affect the performance of the interface. It is possible to manually set noise floor threshold value. By default, it is dynamically calculated. This property also affects received signal strength. This property is only effective on non-AC chips.
nv2-cell-radius (*integer [10..200]*; Default: **30**)	Setting affects the size of contention time slot that AP allocates for clients to initiate connection and also size of time slots used for estimating distance to client. When setting is too small, clients that are farther away may have trouble connecting and/or disconnect with "ranging timeout" error. Although during normal operation the effect of this setting should be negligible, in order to maintain maximum performance, it is advised to not increase this setting if not necessary, so AP is not reserving time that is actually never used, but instead allocates it for actual data transfer. • on AP: distance to farthest client in km • on station: no effect
nv2-noise-floor-offset (*default* | *integer [0..20]*; Default: **default**)	
nv2-preshared-key (*string*; Default:)	
nv2-qos (*default* | *frame-priority*; Default: **default**)	Sets the packet priority mechanism, firstly data from high priority queue is sent, then lower queue priority data until 0 queue priority is reached. When link is full with high priority queue data, lower priority data is not sent. Use it very carefully, setting works on AP • **frame-priority** - manual setting that can be tuned with Mangle rules. • **default** - default setting where small packets receive priority for best latency
nv2-queue-count (*integer [2..8]*; Default: **2**)	
nv2-security (*disabled* | *enabled*; Default: **disabled**)	
on-fail-retry-time (*time [100ms..1s]*; Default: **100ms**)	After third sending failure on the lowest data rate, wait for specified time interval before retrying.
periodic-calibration (*default* | *disabled* | *enabled*; Default: **default**)	Setting *default* enables periodic calibration if info **default-periodic-calibration** property is *enabled*. Value of that property depends on the type of wireless card. This property is only effective for cards based on Atheros chipset.
periodic-calibration-interval (*integer [1..10000]*; Default: **60**)	This property is only effective for cards based on Atheros chipset.

preamble-mode (*both* | *long* | *short*; Default: **both**)	Short preamble mode is an option of 802.11b standard that reduces per-frame overhead.
	• On AP:
	  ○ *long* - Do not use short preamble.
	  ○ *short* - Announce short preamble capability. Do not accept connections from clients that do not have this capability.
	  ○ *both* - Announce short preamble capability.
	• On station:
	  ○ *long* - do not use short preamble.
	  ○ *short* - do not connect to AP if it does not support short preamble.
	  ○ *both* - Use short preamble if AP supports it.

prism-cardtype (*100mW* | *200mW* | *30mW*; Default:)	Specify type of the installed Prism wireless card.
proprietary-extensions (*post-2.9.25* | *pre-2.9.25*; Default: **post-2.9.25**)	RouterOS includes proprietary information in an information element of management frames. This parameter controls how this information is included. • *pre-2.9.25* - This is older method. It can interoperate with newer versions of RouterOS. This method is incompatible with some clients, for example, Centrino based ones. • *post-2.9.25* - This uses standardized way of including vendor specific information, that is compatible with newer wireless clients.

radio-name (*string*; Default: **MAC address of an interface**)	Descriptive name of the device, that is shown in registration table entries on the remote devices. This is a proprietary extension.
rate-selection (*advanced* | *legacy*; Default: **advanced**)	Starting from v5.9 default value is advanced since legacy mode was inefficient.
rate-set (*configured* | *default*; Default: **default**)	Two options are available: • *default* - default basic and supported rate sets are used. Values from **basic-rates** and **supported-rates** parameters have no effect. • *configured* - use values from **basic-rates**, **supported-rates**, **basic-mcs**, **mcs**. Read more >>.
rx-chains (*list of integer [0..3]*; Default: **0**)	Which antennas to use for receive. In current MikroTik routers, both RX and TX chain must be enabled, for the chain to be enabled.
scan-list (*Comma separated list of frequencies and frequency ranges | default. Since v6.35 (wireless-rep) type also support range:step option*; Default: **default**)	The *default* value is all channels from selected band that are supported by card and allowed by the **country** and **frequency-mode** settings (this list can be seen in info). For default scan list in *5ghz* band channels are taken with 20MHz step, in *5ghz-turbo* band - with 40MHz step, for all other bands - with 5MHz step. If **scan-list** is specified manually, then all matching channels are taken. (Example: **scan-list**=*default,5200-5245,2412-2427* - This will use the default value of scan list for current band, and add to it supported frequencies from 5200-5245 or 2412-2427 range.) Since RouterOS v6.0 with Winbox or Webfig, for inputting of multiple frequencies, add each frequency or range of frequencies into separate multiple scan-lists. Using a comma to separate frequencies is no longer supported in Winbox/Webfig since v6.0. Since RouterOS v6.35 (wireless-rep) scan-list support step feature where it is possible to manually specify the scan step. Example: **scan-list**=*5500-5600:20* will generate such scan-list values *5500,5520,5540,5560,5580,5600*
security-profile (*string*; Default: **default**)	Name of profile from security-profiles
secondary-channel (*integer*; Default: **""**)	Specifies secondary channel, required to enable 80+80MHz transmission. To disable 80+80MHz functionality, set secondary-channel to "" or unset the value via CLI/GUI.
ssid (*string (0..32 chars)*; Default: **value of system/identity**)	SSID (service set identifier) is a name that identifies wireless network.
skip-dfs-channels (*string* | *10min-cac* | *all* | *disabled*; Default: **disabled**)	These values are used to skip all DFS channels or specifically skip DFS CAC channels in range 5600-5650MHz which detection could go up to 10min.
station-bridge-clone-mac (*MAC*; Default:)	This property has effect only in the *station-pseudobridge-clone* mode. Use this MAC address when connection to AP. If this value is *00:00:00:00:00:00* , station will initially use MAC address of the wireless interface. As soon as packet with MAC address of another device needs to be transmitted, station will reconnect to AP using that address.
station-roaming (*disabled* | *enabled*; Default: **disabled**)	Station Roaming feature is available only for 802.11 wireless protocol and only for station modes. Read more >>
supported-rates-a/g (*list of rates [12Mbps | 18Mbps | 24Mbps | 36Mbps | 48Mbps | 54Mbps | 6Mbps | 9Mbps]*; Default: **6Mbps; 9Mbps; 12Mbps; 18Mbps; 24Mbps; 36Mbps; 48Mbps; 54Mbps**)	List of supported rates, used for all bands except *2ghz-b*.
supported-rates-b (*list of rates [11Mbps | 1Mbps | 2Mbps | 5.5Mbps]*; Default: **1Mbps; 2Mbps; 5.5Mbps; 11Mbps**)	List of supported rates, used for *2ghz-b*, *2ghz-b/g* and *2ghz-b/g/n* bands. Two devices will communicate only using rates that are supported by both devices. This property has effect only when value of **rate-set** is *configured*.
tdma-period-size (*integer [1..10]*; Default: **2**)	Specifies TDMA period in milliseconds. It could help on the longer distance links, it could slightly increase bandwidth, while latency is increased too.
tx-chains (*list of integer [0..3]*; Default: **0**)	Which antennas to use for transmitting. In current MikroTik routers, both RX and TX chain must be enabled, for the chain to be enabled.

tx-power (*integer [-30..40]*; Default:)	For 802.11ac wireless interface it's total power but for 802.11a/b/g/n it's power per chain.
tx-power-mode (*default, card-rates, all-rates-fixed, manual-table*; Default: **default**)	sets up tx-power mode for wireless card • default - use values stored in the card • all-rates-fixed - use same transmit power for all data rates. Can damage the card if transmit power is set above rated value of the card for used rate. • manual-table - define transmit power for each rate separately. Can damage the card if transmit power is set above rated value of the card for used rate. • card-rates - use transmit power calculated for each rate based on value of **tx-power** parameter. Legacy mode only compatible with currently discontinued products.
update-stats-interval (; Default:)	How often to request update of signals strength and ccq values from clients. Access to registration-table also triggers update of these values. This is proprietary extension.
vht-basic-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **MCS 0-7**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - client must support MCS-0 to MCS-7 • *MCS 0-8* - client must support MCS-0 to MCS-8 • *MCS 0-9* - client must support MCS-0 to MCS-9
vht-supported-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **MCS 0-9**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - devices will advertise as supported MCS-0 to MCS-7 • *MCS 0-8* - devices will advertise as supported MCS-0 to MCS-8 • *MCS 0-9* - devices will advertise as supported MCS-0 to MCS-9
wds-cost-range (*start [-end] integer[1..200000000]*; Default: **50-150**)	Bridge port cost of WDS links are automatically adjusted, depending on measured link throughput. Port cost is recalculated and adjusted every 5 seconds if it has changed by more than 10%, or if more than 20 seconds have passed since the last adjustment. Setting this property to *0* disables automatic cost adjustment. Automatic adjustment does not work for WDS links that are manually configured as a bridge port.
wds-default-bridge (*string* | *none*; Default: **none**)	When WDS link is established and status of the wds interface becomes *running*, it will be added as a bridge port to the bridge interface specified by this property. When WDS link is lost, wds interface is removed from the bridge. If wds interface is already included in a bridge setup when WDS link becomes active, it will not be added to bridge specified by , and will (needs editing)
wds-default-cost (*integer [1..200000000]*; Default: **100**)	Initial bridge port cost of the WDS links.
wds-ignore-ssid (*yes* | *no*; Default: **no**)	By default, WDS link between two APs can be created only when they work on the same frequency and have the same SSID value. If this property is set to *yes*, then SSID of the remote AP will not be checked. This property has no effect on connections from clients in *station-wds* mode. It also does not work if **wds-mode** is *static-mesh* or *dynamic-mesh*.

wds-mode (*disabled* / *dynamic* / *dynamic-mesh* / *static* / *static-mesh*; Default: **disabled**)	Controls how WDS links with other devices (APs and clients in *station-wds* mode) are established. • *disabled* does not allow WDS links. • *static* only allows WDS links that are manually configured in WDS • *dynamic* also allows WDS links with devices that are not configured in WDS, by creating required entries dynamically. Such dynamic WDS entries are removed automatically after the connection with the other AP is lost. -mesh modes use different (better) method for establishing link between AP, that is not compatible with APs in non-mesh mode. This method avoids one-sided WDS links that are created only by one of the two APs. Such links cannot pass any data.When AP or station is establishing WDS connection with another AP, it uses connect-list to check whether this connection is allowed. If station in station-wds mode is establishing connection with AP, AP uses access-list to check whether this connection is allowed.If mode is station-wds, then this property has no effect.
wireless-protocol (*802.11* / *any* / *nstreme* / *nv2* / *nv2-nstreme* / *nv2-nstreme-802.11* / *unspecified*; Default: **any**)	Specifies protocol used on wireless interface; • *unspecified* - protocol mode used on previous RouterOS versions (v3.x, v4.x). Nstreme is enabled by old enable-nstreme setting, Nv2 configuration is not possible. • *any* : on AP - regular 802.11 Access Point or Nstreme Access Point; on station - selects Access Point without specific sequence, it could be changed by connect-list rules. • *nstreme* - enables Nstreme protocol (the same as old enable-nstreme setting). • *nv2* - enables Nv2 protocol. • *nv2 nstreme* : on AP - uses first wireless-protocol setting, always Nv2; on station - searches for Nv2 Access Point, then for Nstreme Access Point. • *nv2 nstreme 802.11* - on AP - uses first wireless-protocol setting, always Nv2; on station - searches for Nv2 Access Point, then for Nstreme Access Point, then for regular 802.11 Access Point. **Warning!** Nv2 doesn't have support for Virtual AP
wmm-support (*disabled* / *enabled* / *required*; Default: **disabled**)	Specifies whether to enable WMM. Only applies to bands B and G. Other bands will have it enabled regardless of this setting
wps-mode (*disabled* / *push-button* / *push-button-virtual-only*; Default: **depending on the device model**)	`Read more >>`

802.11n wireless chipsets represent power per chain and the 802.11ac

wireless chipsets represent the total power, for reference see the table below:Transmit Power representation on 802.11n and 802.11ac

Wireless chipset signal level representation

Wireless chipset	Enabled Chains	Power per Chain	Total Power
802.11n	1	Equal to the selected Tx Power	Equal to the selected Tx Power
802.11n	2	Equal to the selected Tx Power	+3dBm
802.11n	3	Equal to the selected Tx Power	+5dBm
802.11ac	1	Equal to the selected Tx Power	Equal to the selected Tx Power
802.11ac	2	-3dBm	Equal to the selected Tx Power
802.11ac	3	-5dBm	Equal to the selected Tx Power
802.11ac	4	-6dBm	Equal to the selected Tx Power

Basic and MCS Rate table

Default basic and supported rates, depending on selected band

band	basic rates	basic-HT-mcs	basic-VHT-mcs	VHT-mcs	HT-mcs	supported rates
2.4ghz-b	1	-	-	-	-	1-11
2.4ghz-onlyg	6	-	-	-	-	1-11,6-54
2.4ghz-onlyn	6	0-7	-	-	0-23	1-11,6-54
2.4ghz-b/g	1-11	-	-	-	-	1-11,6-54
2.4ghz-b/g/n	1-11	none	-	-	0-23	1-11,6-54
2.4ghz-g/n	6	none	-	-	0-23	6-54
2.4ghz-g-turbo	6	-	-	-	-	6-54
5ghz-a	6	-	-	-	-	6-54
5ghz-a/n	6	none	-	-	0-23	6-54
5ghz-onlyn	6	0-7	-	-	0-23	6-54
5ghz-a/n/ac	6	none	none	0-9	0-23	6-54
5ghz-onlyac	6	none	0-7	0-9	0-23	6-54

Used settings when **rate-set=configured**

band	used settings
2.4ghz-b	basic-b, supported-b
2.4ghz-b/g, 2.4ghz-onlyg	basic-b, supported-b, basic-a/g, supported-a/g
2.4ghz-onlyn, 2.4ghz-b/g/n	basic-b, supported-b, basic-a/g, supported-a/g, ht-basic-mcs, ht-supported-mcs
2.4ghz-g/n	basic-a/g,supported-a/g,ht-basic-mcs,ht-supported-mcs
5ghz-a	basic-a/g,supported-a/g
5ghz-a/n, 5ghz-onlyn	basic-a/g,supported-a/g,ht-basic-mcs,ht-supported-mcs
5ghz-a/n/ac, 5ghz-onlyac	basic-a/g,supported-a/g,ht-basic-mcs,ht-supported-mcs,vht-basic-mcs,vht-supported-mcs

Settings independent from **rate-set**:

1. allowed mcs depending on number of chains:
    - 1 chain: 0-7
    - 2 chains: 0-15
    - 3 chains: 0-23
2. if standard channel width (20Mhz) is not used, then 2ghz modes (except 2.4ghz-b) are not using b rates (1-11)

Frame protection support (RTS/CTS)

802.11 standard provides means to protect the transmission against other device transmission by using RTS/CTS protocol. Frame protection helps to fight "hidden node" problem. There are several types of protection:

- RTS/CTS based protection - device willing to send frame at first sends RequestToSend frame and waits for ClearToSend frame from intended destination. By "seeing" RTS or CTS frame 802.11 compliant devices know that somebody is about to transmit and therefore do not initiate transmission themselves
- "CTS to self" based protection - device willing to send frame sends CTS frame "to itself". As in RTS/CTS protocol every 802.11 compliant device receiving this frame know not to transmit. "CTS to self" based protection has less overhead, but it must be taken into account that this only protects against devices receiving CTS frame (e.g. if there are 2 "hidden" stations, there is no use for them to use "CTS to self" protection, because they will not be able to receive CTS sent by other station - in this case stations must use RTS/CTS so that other station knows not to transmit by seeing CTS transmitted by AP).

Protection mode is controlled by **hw-protection-mode** setting of wireless interface. Possible values: **none** - for no protection (default), **rts-cts** for RTS/CTS based protection or **cts-to-self** for "CTS to self" based protection.

Frame size threshold at which protection should be used is controlled by **hw-protection-threshold** setting of wireless interface.

For example, to enable "CTS-to-self" based frame protection on AP for all frames, not depending on size, use command:

```
[admin@MikroTik] /interface wireless> set 0 hw-protection-mode=cts-to-self hw-protection-threshold=0
```

To enable RTS/CTS based protection on client use command:

```
[admin@MikroTik] /interface wireless> set 0 hw-protection-mode=rts-cts hw-protection-threshold=0
```

Nv2

MikroTik has developed a new wireless protocol based on TDMA technology (Time Division Multiple Access) - (Nstreme version 2). See the Nv2 documentation: NV2

TDMA is a channel access method for shared medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots. The users transmit in rapid succession, one after the other, each using his own time slot. This allows multiple stations to share the same transmission medium (e.g. radio frequency channel) while using only a part of its channel capacity.

The most important benefits of Nv2 are:

- Increased speed
- More client connections in PTM environments
- Lower latency
- No distance limitations
- No penalty for long distances

Nv2 protocol limit is 511 clients.

Warning: Nv2 doesn't have support for Virtual AP

Nv2 Troubleshooting

Increase throughput on long distance with **tdma-period-size**. In Every "period", the Access Point leaves part of the time unused for data transmission (which is equal to *round trip time* - the time in which the frame can be sent and received from the client), it is used to ensure that client could receive the last frame from Access Point, before sending its own packets to it. The longer the distance, the longer the period is unused.

For example, the distance between Access Point and client is 30km. Frame is sent in 100us one direction, respectively round-trip-time is ~200us. **tdma-period-size** default value is 2ms, it means 10% of the time is unused. When tdma-period-size is increased to 4ms, only 5% of time is unused. For 60km wireless link, round-trip-time is 400ms, unused time is 20% for default tdma-period-size 2ms, and 10% for 4ms. Bigger tdma-period-size value increases latency on the link.

Access List

Sub-menu: `/interface wireless access-list`

Access list is used by access point to restrict allowed connections from other devices, and to control connection parameters.

Access list rules are processed one by one until matching rule is found. Then the action in the matching rule is executed. If action specifies that client should be accepted, client is accepted, potentially overriding it's default connection parameters with ones specified in access list rule.

There are the following parameters for access list rules:

- client matching parameters:
  - address - MAC address of the client
  - interface - optional interface to compare with the interface to which client actually connects to
  - time - time of day and days when rule matches
  - signal-range - range in which client signal must fit for the rule to match
  - allow-signal-out-of-range - option which permits client's signal to be out of the range always or for some time interval
- connection parameters:
  - ap-tx-limit - tx speed limit in direction to client

- client-tx-limit - tx speed limit in direction to AP (applies to RouterOS clients only)
- private-passphrase - PSK passphrase to use for this client if some PSK authentication algorithm is used
- vlan-mode - VLAN tagging mode specifies if traffic coming from client should get tagged (and untagged when going to client).
- vlan-id - VLAN ID to use if doing VLAN tagging

Operation:

- Access list rules are checked sequentially.
- Disabled rules are always ignored.
- Only the first matching rule is applied.
- If there are no matching rules for the remote connection, then the default values from the wireless interface configuration are used.
- If remote device is matched by rule that has **authentication**=*no* value, the connection from that remote device is rejected.

Warning: If there is no entry in ACL about client which connects to AP (wireless,debug wlan2: A0:0B:BA:D7:4D:B2 not in local ACL, by default accept), then ACL for this client is ignored during all connection time.

For example, if client's signal during connection is -41 and we have ACL rule

```
/interface/wireless/access-list
add authentication=yes forwarding=yes interface=wlan2 signal-range=-55..0
```

Then the connection is matched to the ACL rule, but if signal drops below -55, client will not be disconnected.

Please note that if "default-authentication=yes" is set on the wireless interface, clients will be able to join even if there are no matching access-list entries. To make it work correctly it is required that client is matched by any of ACL rules.

If we modify ACL rules in the previous example to:

```
/interface/wireless/access-list
add interface=wlan2 signal-range=-55..0
add authentication=no forwarding=no interface=wlan2 signal-range=-120..-56
```

Then if signal drops to -56, client will be disconnected.

Properties

Property	Description
ap-tx-limit (*integer [0..4294967295]*; Default: **0**)	Limit rate of data transmission to this client. Value *0* means no limit. Value is in bits per second.
authentication (*yes* | *no*; Default: **yes**)	- *no* - Client association will always fail. - *yes* - Use authentication procedure that is specified in the **security-profile** of the interface.
client-tx-limit (*integer [0..4294967295]*; Default: **0**)	Ask client to limit rate of data transmission. Value *0* means no limit. This is a proprietary extension that is supported by RouterOS clients. Value is in bits per second.
comment (*string*; Default:)	Short description of an entry
disabled (*yes* | *no*; Default: **no**)	
forwarding (*yes* | *no*; Default: **yes**)	- *no* - Client cannot send frames to other station that are connected to same access point. - *yes* - Client can send frames to other stations on the same access point.
interface (*string* | *any* | *all*; Default: **any**)	Rules with **interface**=*any* are used for any wireless interface and the **interface**=*all* defines interface-list “all” name. To make rule that applies only to one wireless interface, specify that interface as a value of this property.

mac-address (*MAC*; Default: **00:00:00:00:00:00**)	Rule matches client with the specified MAC address. Value *00:00:00:00:00:00* matches always.
management-protection-key (*string*; Default: **""**)	
private-algo (*104bit-wep | 40bit-wep | aes-ccm | none | tkip*; Default: **none**)	Only for WEP modes.
private-key (*string*; Default: **""**)	Only for WEP modes.
private-pre-shared-key (*string*; Default: **""**)	Used in WPA PSK mode.
signal-range (*NUM..NUM - both NUM are numbers in the range -120..120*; Default: **-120..120**)	Rule matches if signal strength of the station is within the range. If signal strength of the station will go out of the range that is specified in the rule, access point will disconnect that station.
time (*TIME-TIME,sun,mon,tue,wed,thu,fri,sat - TIME is time interval 0..86400 seconds; all day names are optional; value can be unset*; Default:)	Rule will match only during specified time. Station will be disconnected after specified time ends. Both start and end time is expressed as time since midnight, 00:00. Rule will match only during specified days of the week.

Align

Sub-menu: `/interface wireless align`

Align tool is used to help in alignment devices running this tool.

Property	Description
active-mode (*yes | no*; Default: **yes**)	If in active mode, will send out frames for align.
audio-max (*integer [-2147483648..2147483647]*; Default: **-20**)	Maxumum signal strength for beeper
audio-min (*integer [-2147483648..2147483647]*; Default: **-100**)	Minimum signal strength for beeper
audio-monitor (*MAC*; Default: **00:00:00:00:00:00**)	Which MAC address to use for audio monitoring
filter-mac (*MAC*; Default: **00:00:00:00:00:00**)	Filtered out MAC address that will be shown in monitor screen.
frame-size (*integer [200..1500]*; Default: **300**)	Size of the frames used by monitor.
frames-per-second (*integer [1..100]*; Default: **25**)	Frame transmit interval
receive-all (*yes | no*; Default: **no**)	If set to "yes", monitor will find all available devices.
ssid-all (*yes | no*; Default: **no**)	Whether to show all SSIDs in the monitor or only one configured in wireless settings.

Menu Specific Commands

Property	Description
monitor (*interface name*)	Start align monitoring
test-audio (*integer [-2147483648..2147483647]*)	Test the beeper

Connect List

Sub-menu: `/interface wireless connect-list`

connect-list is used to assign priority and security settings to connections with remote access points, and to restrict allowed connections. connect-list is an ordered list of rules. Each rule in connect-list is attached to specific wireless interface, specified in the `interface`property of that rule (this is unlike access-list, where rules can apply to all interfaces). Rule can match MAC address of remote access point, it's signal strength and many other parameters.

Operation:

- connect-list rules are always checked sequentially, starting from the first.
- disabled rules are always ignored.
- Only the first matching rule is applied.
- If SSID or exact wireless protocol is provided in the wireless interface configuration Connect List SSIDs or wireless protocols not covered by wireless interface configuration are ignored.
- If connect-list does not have any rule that matches remote access point, then the default values from the wireless interface configuration are used.
- If access point is matched by rule that has **connect**=*no* value, connection with this access point will not be attempted.
- If access point is matched by rule that has **connect**=*yes* value, connection with this access point will be attempted.
    - In station mode, if several remote access points are matched by connect list rules with **connect**=*yes* value, connection will be attempted with access point that is matched by rule higher in the connect-list.
    - If no remote access points are matched by connect-list rules with **connect**=*yes* value, then value of **default-authentication** interface property determines whether station will attempt to connect to any access point. If **default-authentication**=*yes*, station will choose access point with best signal and compatible security.
- In access point mode, connect-list is checked before establishing WDS link with remote device. If access point is not matched by any rule in the connect list, then the value of **default-authentication** determines whether WDS link will be established.

Properties

Property	Description
3gpp (*string*; Default:)	
area-prefix (*string*; Default:)	Rule matches if area value of AP (a proprietary extension) begins with specified value.**area** value is a proprietary extension.
comment (*string*; Default:)	Short description of an entry
connect (*yes* | *no*; Default: **yes**)	Available options: - *yes* - Connect to access point that matches this rule. - *no* - Do not connect to any access point that matches this rule.
disabled (*yes* | *no*; Default: **no**)	
mac-address (*MAC*; Default: **00:00:00:00:00:00**)	Rule matches only AP with the specified MAC address. Value *00:00:00:00:00:00* matches always.
security-profile (*string* | *none*; Default: **none**)	Name of security profile that is used when connecting to matching access points, If value of this property is *none*, then security profile specified in the interface configuration will be used. In station mode, rule will match only access points that can support specified security profile. Value *none* will match access point that supports security profile that is specified in the interface configuration. In access point mode value of this property will not be used to match remote devices.
signal-range (*NUM..NUM - both NUM are numbers in the range -120..120*; Default: **-120..120**)	Rule matches if signal strength of the access point is within the range. If station establishes connection to access point that is matched by this rule, it will disconnect from that access point when signal strength goes out of the specified range.
ssid (*string*; Default: **""**)	Rule matches access points that have this SSID. Empty value matches any SSID. This property has effect only when station mode interface **ssid** is empty, or when access point mode interface has **wds-ignore-ssid**=*yes*
wireless-protocol (*802.11* | *any* | *nstreme* | *tdma*; Default: **any**)	
interface (*string*; Default:)	Each rule in connect list applies only to one wireless interface that is specified by this setting.

Usage

Restrict station connections only to specific access points

- Set value of **default-authentication** interface property to *no*.

/interface wireless set station-wlan default-authentication=no

- Create rules that matches allowed access points. These rules must have **connect**=*yes* and **interface** equal to the name of station wireless interface.

/interface wireless connect-list add interface=station-wlan connect=yes mac-address=00:11:22:33:00:01/interface wireless connect-list add interface=station-wlan connect=yes mac-address=00:11:22:33:00:02

Disallow connections to specific access points

- Set value of **default-authentication** interface property to *yes*.

/interface wireless set station-wlan default-authentication=yes

- Create **connect**=*no* rules that match those access points that station should not connect to. These rules must have **connect**=*no* and **interface** equal to the name of station wireless interface.

/interface wireless connect-list add interface=station-wlan connect=no mac-address=00:11:22:33:44:55

Select preferred access points

- Create rules that match preferred access points. These rules must have **connect**=*yes* and **interface** equal to the name of station wireless interface.
- Put rules that match preferred access points higher in the connect-list, in the order of preference.

Restrict WDS link establishment

- Place rules that match allowed access points at the top.
- Add deny-all rule at the end of connect list.

Info

Sub-menu: `/interface wireless info`

Property	Description
2ghz-10mhz-power-channels ()	
2ghz-11n-channels ()	
2ghz-5mhz-power-channels ()	
2ghz-b-channels ()	
2ghz-g-channels ()	
2ghz-g-turbo-channels ()	
5ghz-10mhz-power-channels ()	
5ghz-11n-channels ()	
5ghz-5mhz-power-channels ()	
5ghz-channels ()	
5ghz-turbo-channels ()	
allowed-channels	List of available channels for each band

capabilities ()	
country-info()	Takes country name as argument, shows available bands, frequencies and maximum transmit power for each frequency.
chip-info ()	
default-periodic-calibration ()	
firmware ()	
ht-chains ()	
interface-type ()	
name ()	
pci-info ()	
supported-bands ()	

Manual TX Power Table

Sub-menu: `/interface wireless manual-tx-power-table`

Property	Description
comment (*string*; Default:)	Short description of an entry
manual-tx-powers (*list of [Rate:TxPower];* Rate ::= 11Mbps | 12Mbps | 18Mbps | 1Mbps | 24Mbps | ... *TxPower ::= integer [-30..30]*; Default:)	
name (*string*)	Name of the wireless interface to which tx powers will be applied.

Wireless hardware table

Warning: You must follow to regulatory domain requirements in your country. If you are allowed to use other frequencies, note that Antenna Gain and Transmit Power may decrease depending on board and frequency. Devices are calibrated only for regulatory frequencies, use non standard frequencies at your own risk. The list only specifies frequencies accepted by the wireless chip, these frequencies might not always work due to antenna that is built into the product, device design, filters and other factors. USE STRICTLY AT YOUR OWN RISK

It is possible to deduce supported channel width from the product page, to do so you need to check the following parameters: **number of chains** and the **max data rate**. Once you know these parameters, you need to check the modulation and coding scheme (MCS) table, for example, here: https://mcsindex.com/.

If we take hAP ac, as an example, we can see that number of chains is 3, and the max data rate is 1300 in the MCS table. In the MCS table we need to find entry for 3 spatial streams - chains, and the respective data rate, which in this case shows us that 80MHz is the maximum supported channel width.

Integrated wireless interface frequency table

Board name	Wireless interfaces	Frequency range [MHz]	Supported channel widths [Mhz]
2011UAS-2HnD	1	2312-2732	20,40
751G-2HnD	1	2200-2700	20,40 and advanced channel support
751U-2HnD	1	2200-2700	20,40 and advanced channel support
911-2Hn	1	2312-2732	20,40
911-5HacD	1	4920-6100	20,40,80

911-5Hn	1	4920-6100	5,10,20,40
911-5HnD	1	4920-6100	20,40
911G-2HPnD	1	2312-2732	20,40
911G-5HPacDr2 /-NB /-QRT	1	4920-6100	5,10,20,40,80
911G-5HPnD /-QRT	1	4920-6100	5,10,20,40
912UAG-2HPnD /-OUT	1	2312-2732	20,40
912UAG-5HPnD /-OUT	1	4920-6100	5,10,20,40
912UAG-6HPnD /-OUT	1	5500-6500	and 20,40
921GS-5HPacD-15S /-19S	1	4920-6100	5^1,10^1,20,40,80
22UGS-5HPacD2HnD	2	4920-5925,2412-2482	20,40,80 and 20,40
921UAGS-5SHPacD-NM	1	4920-6100	20,40,80
921UAGS-5SHPacT-NM	1	4920-6100	20,40,80
922UAGS-5HPacD /-NM	1	4920-6100	20,40,80
922UAGS-5HPacT /-NM	1	4920-6100	20,40,80
941-2nD /-TC	1	2312-2732	20,40
951G-2HnD	1	2312-2732	20,40
951Ui-2HnD	1	2312-2732	20,40
951Ui-2nD	1	2312-2732	20,40
952Ui-5ac2nD /-TC	2	2312-2732,4920-6100	20,40 and 20,40,80
953GS-5HnT /-RP	1	4920-6100	5,10,20,40
962UiGS-5HacT2HnT	2	2312-2732,4920-6100	20,40 and 20,40,80
cAP2n	1	2312-2732	20,40
cAP2nD	1	2312-2732	20,40
cAPL-2nD	1	2312-2732	20,40
CRS109-8G-1S-2HnD-IN	1	2312-2732	20,40
CRS125-24G-1S-2HnD-IN	1	2312-2732	20,40
Disc-5nD	1	4920-6100	20,40
DynaDishG-5HacD	1	4920-6100	5^1,10^1,20,40,80
DynaDishG-6HnD	1	5500-6500	20,40
Groove52HPn	1	4920-6100,2312-2732	5,10,20,40 and 5,10,20,40
GrooveA-52HPn	1	4920-6100,2312-2732	5,10,20,40 and 5,10,20,40
GrooveG-52HPacn	1	4920-6100,2312-2732	20,40,80 and 20,40
GrooveGA-52HPacn	1	4920-6100,2312-2732	20,40,80 and 20,40
LDF-5nD	1	4920-6100	20,40
LHG-5nD	1	4920-6100	20,40
mAP2n	1	2312-2732	20,40
mAP2nD	1	2312-2732	20,40
mAPL-2nD	1	2312-2732	20,40
Metal2SHPn	1	2200-2700	20,40 and advanced channel support
Metal5SHPn	1	4800-6100	5,10,20,40 and advanced channel support
Metal9HPn	1	902-928	5,10,20
MetalG-52SHPacn	1	4920-6100,2312-2732	20,40,80 and 20,40

OmniTikG-5HacD	1	4920-6100	20,40,80
OmniTikPG-5HacD	1	4920-6100	20,40,80
OmniTIKU-5HnD	1	4800-6100	5,10,20,40
OmniTIKUPA-5HnD	1	4800-6100	5,10,20,40
QRTG-2SHPnD	1	2312-2732	20,40
SEXTANTG-5HPnD	1	4920-6100	20,40
SXT2nDr2	1	2312-2732	20,40
SXT5HacD2n	2	2312-2732,4920-6100	5^1,10^1,20,40 and 5^1,10^1,20,40,80
SXT5HPnDr2	1	4920-6100	20,40
SXT5nDr2	1	4920-6100	20,40
SXTG-2HnD	1	2200-2700	20,40
SXTG-2HnDr2	1	2300-2700	20,40
SXTG-5HPacD	1	4920-6100	5^1,10^1,20,40,80
SXTG-5HPacD-HG /-SA	1	4920-6100	5^1,10^1,20,40,80
SXTG-5HPnD-HGr2 /-SAr2	1	4920-6100	20,40
SXTG-6HPnD	1	5500-6500	20,40
SXTsq2nD	1	2312-2484	20,40
wAP2nD /-BE	1	2312-2732	20,40
wAPG-5HacT2HnD /-BE	2	2312-2732,4920-6100	20,40 and 20,40,80
R11e-2HnD	1	2312-2732	20,40
R11e-2HPnD	1	2312-2732	20,40
R11e-5HacD	1	4920-6100	20,40,80
R11e-5HacT	1	4920-6100	20,40,80
R11e-5HnD	1	4920-6100	20,40
R2SHPn	1	2200-2700	20,40 and advanced channel support
R52H	1	4920-6100,2192-2507	20 and 20
R52HnD	1	4800-6100,2200-2700	20,40 and 20,40
R52nM	1	4800-6100,2200-2700	20,40 and 20,40 and advanced channel support
R5SHPn	1	4800-6100	20,40 and advanced channel support

NOTES:

1. - Only in 802.11a/n standard

Overview

Advanced Channels feature provides extended opportunities in wireless interface configuration:

- scan-list that covers multiple bands and channel widths;
- non-standard channel center frequencies (specified with KHz granularity) for hardware that allows that;
- non-standard channel widths (specified with KHz granularity) for hardware that allows that.

Hardware support

Non standard center frequency and width channels can only be used with interfaces that support it.

Currently **only Atheros AR92xx** based chips support non-standard center frequencies and widths with the following ranges:

- center frequency range: 2200MHz-2500MHz with step 0.5MHz (500KHz), width range: 2.5MHz-30MHz width step 0.5MHz (500KHz);

- center frequency range: 4800MHz-6100MHz with step 0.5MHz (500KHz), width range: 2.5MHz-30MHz width step 0.5MHz (500KHz);

AR93xx doesn't support this feature

Configuring Advanced Channels

Advanced Channels are configured in **interface wireless channels** menu. This menu contains ordered list of user-defined channels that can be grouped by means of **list** property. Channels have the following properties:

- **name** - name by which this channel can be referred to. If **name** is not specified when adding channel, it will be automatically generated from channel frequency and width;
- **list** - name of list this channel is part of. Lists can be used to group channels;
- **frequency** - channel center frequency in MHz, allowing to specify fractional MHz part, e.g. **5181.5**;
- **width** - channel width in MHz, allowing to specify fractional MHz part, e.g. **14.5**;
- **band** - defines default set of data rates when using this channel;
- **extension-channel** - specifies placement of 11n extension channel.

Using Advanced Channels

In order to use Advanced Channels in wireless interface configuration, several interface settings accept channel names or list names as arguments. It is possible to configure interface with channel that interface does not support. In this case interface will not become operational. It is sole responsibility of administrator to configure channels in proper way.

frequency

To use particular Advanced Channel for wireless interface (applies to modes that make use of interface **frequency** setting) specify channel name in interface **frequency** setting. For example, to configure interface to operate with center frequency 5500MHz and channel width 14MHz, use the following commands:

```
[admin@MikroTik] /interface wireless> channels add name=MYCHAN frequency=5500 width=14 band=5ghz-onlyn
 list=MYLIST
[admin@MikroTik] /interface wireless> set wlan1 frequency=MYCHAN
```

scan-list

Interface **scan-list** is used in multiple modes that either gather information for list of channels (like interactive **scan** command) or selects channel to work on (like any of **station** modes or AP modes performing DFS). Interface **scan-list** can be configured with comma-separated list of the following items:

- **default** - default .11 channel list for given country and interface band and channel width;
- numeric frequency ranges in MHz;
- Advanced Channel, referred to by name;
- Advanced Channel list, referred to by list name.

For example, to configure interface to scan 5180MHz, 5200MHz and 5220MHz at first using channel width 20MHz and then using channel width 10MHz, the following commands can be issued:

```
[admin@MikroTik] /interface wireless> channels add frequency=5180 width=20 band=5ghz-a list=20MHz-list
[admin@MikroTik] /interface wireless> channels add frequency=5200 width=20 band=5ghz-a list=20MHz-list
[admin@MikroTik] /interface wireless> channels add frequency=5220 width=20 band=5ghz-a list=20MHz-list
[admin@MikroTik] /interface wireless> channels add frequency=5180 width=10 band=5ghz-a list=10MHz-list
[admin@MikroTik] /interface wireless> channels add frequency=5200 width=10 band=5ghz-a list=10MHz-list
[admin@MikroTik] /interface wireless> channels add frequency=5220 width=10 band=5ghz-a list=10MHz-list
[admin@MikroTik] /interface wireless> set wlan1 scan-list=20MHz-list,10MHz-list
```

Nstreme

Sub-menu: `/interface wireless nstreme`

This menu allows to switch a wireless card to the nstreme mode. In this case the card will work only with nstreme clients.

Property	Description

comment (*string*; Default:)	Short description of an entry
disable-csma (*yes* | *no*; Default: **no**)	Disable CSMA/CA when polling is used (better performance)
enable-nstreme (*yes* | *no* ; Default: **no**)	Whether to switch the card into the nstreme mode
enable-polling (*yes* | *no*; Default: **yes**)	Whether to use polling for clients
framer-limit (*integer [100..4000]*; Default: **3200**)	Maximal frame size
framer-policy (*best-fit* | *dynamic-size* | *exact-size* | *none*; Default: **none**)	The method how to combine frames. A number of frames may be combined into a bigger one to reduce the amount of protocol overhead (and thus increase speed). The card is not waiting for frames, but in case a number of packets are queued for transmitting, they can be combined. There are several methods of framing: • **none** - do nothing special, do not combine packets (framing is disabled) • **best-fit** - put as many packets as possible in one frame, until the framer-limit limit is met, but do not fragment packets • **exact-size** - put as many packets as possible in one frame, until the framer-limit limit is met, even if fragmentation will be needed (best performance) • **dynamic-size** - choose the best frame size dynamically
name (*string*)	Name of an interface, to which setting will be applied. Read only.

Note: The settings here (except for enabling nstreme) are relevant only on Access Point, they are ignored for client devices! The client automatically adapts to the AP settings.
WDS for Nstreme protocol requires using station-wds mode on one of the peers. Configurations with WDS between AP modes (bridge and ap-bridge) will not work.

Nstreme Dual

Sub-menu: `/interface wireless nstreme-dual`

Two radios in nstreme-dual-slave mode can be grouped together to make nstreme2 Point-to-Point connection. To put wireless interfaces into a nstreme2 group, you should set their mode to nstreme-dual-slave. Many parameters from /interface wireless menu are ignored, using the nstreme2, except:

- frequency-mode
- country
- antenna-gain
- tx-power
- tx-power-mode
- antenna-mode

Property	Description
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	Read more >>
comment (*string*; Default:)	Short description of an entry
disable-csma (*yes* | *no*; Default: **no**)	Disable CSMA/CA (better performance)
disable-running-check (*yes* | *no*; Default: **no**)	Whether the interface should always be treated as running even if there is no connection to a remote peer

disabled (*yes* | *no*; Default: **yes**)	
framer-limit (*integer [64..4000]*; Default: **2560**)	Maximal frame size
framer-policy (*best-fit* | *exact-size* | *none*; Default: **none**)	The method how to combine frames. A number of frames may be combined into one bigger one to reduce the amout of protocol overhead (and thus increase speed). The card are not waiting for frames, but in case a number packets are queued for transmitting, they can be combined. There are several methods of framing: • **none** - do nothing special, do not combine packets • **best-fit** - put as much packets as possible in one frame, until the framer-limit limit is met, but do not fragment packets • **exact-size** - put as much packets as possible in one frame, until the framer-limit limit is met, even if fragmentation will be needed (best performance)
ht-channel-width (*2040mhz* | *20mhz* | *40mhz*; Default: **20mhz**)	
ht-guard-interval (*both* | *long* | *short*; Default: **long**)	
ht-rates (*list of rates [1,2,3,4,5,6,7,8]*; Default: **1,2, 3,4,5,6,7,8**)	
ht-streams (*both* | *double* | *single*; Default: **single**)	
l2mtu (*integer [0..65536]*; Default:)	
mtu (*integer [0..65536]*; Default: **1500**)	
name (*string*; Default:)	Name of an entry
rates-a/g (*list of rates [6Mbps,9Mbps, 12Mbps, 18Mbps, 24Mbps, 36Mbps, 48Mbps, 54Mbps]*; Default: **6Mbps,9Mbps,12Mbps, 18Mbps, 24Mbps, 36Mbps, 48Mbps, 54Mbps**)	Rates to be supported in 802.11a or 802.11g standard
rates-b (*list of rates [1Mbps, 2Mbps, 5.5Mbps, 11Mbps]*; Default: **1Mbps, 2Mbps, 5.5Mbps, 11Mbps**)	Rates to be supported in 802.11b standard
remote-mac (*MAC*; Default: **00:00:00:00:00:00**)	Which MAC address to connect to (this would be the remote receiver card's MAC address)
rx-band (*2ghz-b* | *2ghz-g* | *2ghz-n* | *5ghz-a* | *5ghz-n*; Default:)	Operating band of the receiving radio
rx-channel-width (*10mhz*; Default: **20mhz**)	
rx-frequency (*integer [0..4294967295]*; Default:)	RX card operation frequency in Mhz.
rx-radio (*string*; Default:)	Name of the interface used for receive.
tx-band (*2ghz-b* | *2ghz-g* | *2ghz-n* | *5ghz-a* | *5ghz-n*; Default:)	Operating band of the transmitting radio
tx-channel-width (*10mhz*; Default: **20mhz**)	
tx-frequency (*integer [0..4294967295]*; Default:)	TX card operation frequency in Mhz.
tx-radio (*string*; Default:)	Name of the interface used for transmit.

Warning: WDS cannot be used on Nstreme-dual links.

Note: The difference between tx-freq and rx-freq should be about 200MHz (more is recommended) because of the interference that may occur!

Note: You can use different bands for rx and tx links. For example, transmit in 2ghz-g and receive data, using 2ghz-b band.

Registration Table

Sub-menu: `/interface wireless registration-table`

In the registration table, you can see various information about currently connected clients. It is used only for Access Points.

All properties are read-only.

Property	Description
802.1x-port-enabled (*yes* | *no*)	whether the data exchange is allowed with the peer (i.e., whether 802.1x authentication is completed, if needed)
ack-timeout (*integer*)	current value of ack-timeout
ap (*yes* | *no*)	Shows whether registered device is configured as access point.
ap-tx-limit (*integer*)	transmit rate limit on the AP, in bits per second
authentication-type ()	authentication method used for the peer
bridge (*yes* | *no*)	
bytes (*integer , integer*)	number of sent and received packet bytes
client-tx-limit (*integer*)	transmit rate limit on the AP, in bits per second
comment (*string*)	Description of an entry. comment is taken from appropriate Access List entry if specified.
compression (*yes* | *no*)	whether data compresson is used for this peer
distance (*integer*)	
encryption (*aes-ccm* | *tkip*)	unicast encryption algorithm used
evm-ch0 ()	
evm-ch1 ()	
evm-ch2 ()	

frame-bytes (*integer, integer*)	number of sent and received data bytes excluding header information
frames (*integer, integer*)	Number of frames that need to be sent over wireless link. This value can be compared to **hw-frames** to check wireless retransmits. Read more >>
framing-current-size (*integer*)	current size of combined frames
framing-limit (*integer*)	maximal size of combined frames
framing-mode ()	the method how to combine frames
group-encryption ()	group encryption algorithm used
hw-frame-bytes (*integer, integer*)	number of sent and received data bytes including header information
hw-frames (*integer, integer*)	Number of frames sent over wireless link by the driver. This value can be compared to **frames** to check wireless retransmits. Read more >>
interface (*string*)	Name of the wireless interface to which wireless client is associated
last-activity (*time*)	last interface data tx/rx activity
last-ip (*IP Address*)	IP address found in the last IP packet received from the registered client
mac-address (*MAC*)	MAC address of the registered client
management-protection (*yes | no*)	
nstreme (*yes | no*)	Shows whether Nstreme is enabled
p-throughput (*integer*)	estimated approximate throughput that is expected to the given peer, taking into account the effective transmit rate and hardware retries. Calculated once in 5 seconds
packed-bytes (*integer, integer*)	number of bytes packed into larger frames for transmitting/receiving (framing)

packed-frames (*integer, integer*)	number of frames packed into larger ones for transmitting/receiving (framing)
packets (*integer. integer*)	number of sent and received network layer packets
radio-name (*string*)	radio name of the peer
routeros-version (*string*)	RouterOS version of the registered client
rx-ccq ()	Client Connection Quality (CCQ) for receive. `Read more >>`
rx-rate (*integer*)	receive data rate
signal-strength (*integer*)	average strength of the client signal recevied by the AP
signal-strength-ch0 ()	
signal-strength-ch1 ()	
signal-strength-ch2 ()	
signal-to-noise ()	
strength-at-rates ()	signal strength level at different rates together with time how long were these rates used
tdma-retx ()	
tdma-rx-size ()	
tdma-timing-offset ()	tdma-timing-offset is proportional to **distance** and is approximately two times the propagation delay. AP measures this so that it can tell clients what offset to use for their transmissions - clients then subtract this offset from their target transmission time such that propagation delay is accounted for and transmission arrives at AP when expected. You may occasionally see small negative value (like few usecs) there for close range clients because of additional unaccounted delay that may be produced in transmitter or receiver hardware that varies from chipset to chipset.
tdma-tx-size (*integer*)	Value in bytes that specifies the size of data unit whose loss can be detected (data unit over which CRC is calculated) sent by device. In general - the bigger the better, because overhead is less. On the other hand, small value in this setting can not always be considered a signal that connection is poor - if device does not have enough pending data that would enable it to use bigger data units (e.g. if you are just pinging over link), this value will not go up.
tdma-windfull ()	
tx-ccq ()	Client Connection Quality (CCQ) for transmit. `Read more >>`
tx-evm-ch0 ()	

tx-evm-ch1 ()	
tx-evm-ch2 ()	
tx-frames-timed-out ()	
tx-rate ()	
tx-signal-strength ()	
tx-signal-strength-ch0 ()	
tx-signal-strength-ch1 ()	
tx-signal-strength-ch2 ()	
uptime (*time*)	time the client is associated with the access point
wds (*yes* | *no*)	whether the connected client is using wds or not
wmm-enabled (*yes* | *no*)	Shows whether WMM is enabled.

Security Profiles

Sub-menu: `/interface wireless security-profiles`

Security profiles are configured under the **/interface wireless security-profiles** path in the console, or in the "Security Profiles" tab of the "Wireless" window in the WinBox. Security profiles are referenced by the Wireless interface security-profile property and security-profile property of Connect Lists.

Basic properties

Property	Description
mode (*none* | *static-keys-optional* | *static-keys-required* | *dynamic-keys*; Default: **none**)	Encryption mode for the security profile. • none - Encryption is not used. Encrypted frames are not accepted. • static-keys-required - WEP mode. Do not accept and do not send unencrypted frames. Station in *static-keys-required* mode will not connect to an Access Point in *static-keys-optional* mode. • static-keys-optional - WEP mode. Support encryption and decryption, but allow also to receive and send unencrypted frames. Device will send unencrypted frames if encryption algorithm is specified as *none*. Station in *static-keys-optional* mode will not connect to an Access Point in *static-keys-required* mode. See also: static-sta-private-algo, static-transmit-key. • dynamic-keys - WPA mode.
name (*text*; Default:)	Name of the security profile

WPA properties

These properties have effect only when **mode** is set to *dynamic-keys*.

Property	Description
authentication-types (*wpa-psk* | *wpa2-psk* | *wpa-eap* | *wpa2-eap*; Default:)	Set of supported authentication types, multiple values can be selected. Access Point will advertise supported authentication types, and client will connect to Access Point only if it supports any of the advertised authentication types.
disable-pmkid (*no* | *yes*; Default: **no**)	Whether to include PMKID into the EAPOL frame sent out by the Access Point. Disabling PMKID can cause compatibility issues with devices that use the PMKID to connect to an Access Point. • yes - removes PMKID from EAPOL frames (improves security, reduces compatibility). • no - includes PMKID into EAPOL frames (reduces security, improves compatibility). This property only has effect on Access Points.
unicast-ciphers (*tkip* | *aes-ccm*; Default: **aes-ccm**)	Access Point advertises that it supports specified ciphers, multiple values can be selected. Client attempts connection only to Access Points that supports at least one of the specified ciphers. One of the ciphers will be used to encrypt unicast frames that are sent between Access Point and Station.
group-ciphers (*tkip* | *aes-ccm*; Default: **aes-ccm**)	Access Point advertises one of these ciphers, multiple values can be selected. Access Point uses it to encrypt all broadcast and multicast frames. Client attempts connection only to Access Points that use one of the specified group ciphers. • tkip - Temporal Key Integrity Protocol - encryption protocol, compatible with legacy WEP equipment, but enhanced to correct some of the WEP flaws. • aes-ccm - more secure WPA encryption protocol, based on the reliable AES (Advanced Encryption Standard). Networks free of WEP legacy should use only this cipher.
group-key-update (*time: 30s..1d*; Default: **5m**)	Controls how often Access Point updates the group key. This key is used to encrypt all broadcast and multicast frames. property only has effect for Access Points.
wpa-pre-shared-key (*text*; Default:)	WPA pre-shared key mode requires all devices in a BSS to have common secret key. Value of this key can be an arbitrary text. Commonly referred to as the network password for WPA mode. property only has effect when *wpa-psk* i s added to **authentication-types**.
wpa2-pre-shared-key (*text*; Default:)	WPA2 pre-shared key mode requires all devices in a BSS to have common secret key. Value of this key can be an arbitrary text. Commonly referred to as the network password for WPA2 mode. property only has effect when *wpa2-psk* is added to **authentication-types**.

Note: RouterOS also allows to override pre-shared key value for specific clients, using either the private-pre-shared-key property, or the Mikrotik-Wireless-Psk attribute in the RADIUS MAC authentication response. This is an extension.

WPA EAP properties

These properties have effect only when **authentication-types** contains *wpa-eap* or *wpa2-eap*, and **mode** is set to *dynamic-keys*.

Property	Description
eap-methods (*eap-tls* | *eap-ttls-mschapv2* | *passthrough* | *peap*; Default: **passthrough**)	Allowed types of authentication methods, multiple values can be selected. This property only has effect on Access Points. • eap-tls - Use built-in EAP TLS authentication. Both client and server certificates are supported. See description of **tls-mode** and **tls-certificate** properties. • eap-ttls-mschapv2 - Use EAP-TTLS with MS-CHAPv2 authentication. • passthrough - Access Point will relay authentication process to the RADIUS server. • peap - Use Protected EAP authentication.
supplicant-identity (*text*; Default: **Identity**)	EAP identity that is sent by client at the beginning of EAP authentication. This value is used as a value for User-Name attribute in RADIUS messages sent by RADIUS EAP accounting and RADIUS EAP pass-through authentication.

mschapv2-username (*text*; Default:)	Username to use for authentication when *eap-ttls-mschapv2* or *peap* authentication method is being used. This property only has effect on Stations.
mschapv2-password (*text*; Default:)	Password to use for authentication when *eap-ttls-mschapv2* or *peap* authentication method is being used. This property only has effect on Stations.
tls-mode (*verify-certificate* | *dont-verify-certificate* | *no-certificates* | *verify-certificate-with-crl*; Default: **no-certificates**)	This property has effect only when **eap-methods** contains *eap-tls*. • verify-certificate - Require remote device to have valid certificate. Check that it is signed by known certificate authority. No additional identity verification is done. Certificate may include information about time period during which it is valid. If router has incorrect time and date, it may reject valid certificate because router's clock is outside that period. See also the Certificates configuration. • dont-verify-certificate - Do not check certificate of the remote device. Access Point will not require client to provide certificate. • no-certificates - Do not use certificates. TLS session is established using 2048 bit anonymous Diffie-Hellman key exchange. • verify-certificate-with-crl - Same as *verify-certificate* but also checks if the certificate is valid by checking the Certificate Revocation List.
tls-certificate (*none* | *name*; Default: **none**)	Access Point always needs a certificate when configured when **tls-mode** is set to *verify-certificate*, or is set to *dont-verify-certificate*. Client needs a certificate only if Access Point is configured with **tls-mode** set to *verify-certificate*. In this case client needs a valid certificate that is signed by a CA known to the Access Point. This property only has effect when **tls-mode** is not set to *no-certificates* and **eap-methods** contains *eap-tls*.

Note: The order of allowed authentication methods in `eap-methods` is important, the same order is going to be used to send authentication method offers to the Station. Example: Access Point uses security-profile where **eap-methods** is set to *eap-tls,passthrough*; 1) Access Point offers EAP-TLS method to the client; 2) Client refuses; 3) Access Point starts relaying EAP communication to the radius server.

Note: When the AP is used for passthrough it is not required to add certificates on the AP itself, the AP device works as a transparent bridge and forwards the EAP-TLS association data from RADIUS server to the end client.

Note: When `tls-mode` is using either `verify-certificate` or `dont-verify-certificate`, then the remote device has to support one of the *RC4-MD5*, *RC4-SHA* or *DES-CBC3-SHA* TLS cipher suites. When using `no-certificates` mode, then the remote device must support "ADH-DES-CBC3-SHA" cipher suite.

RADIUS properties

Property	Description
radius-mac-authentication (*yes* | *no*; Default: **no**)	This property affects the way how Access Point processes clients that are not found in the Access List. • no - allow or reject client authentication based on the value of default-authentication property of the Wireless interface. • yes - Query RADIUS server using MAC address of client as user name. With this setting the value of default-authentication has no effect.
radius-mac-accounting (*yes* | *no*; Default: **no**)	
radius-eap-accounting (*yes* | *no*; Default: **no**)	
radius-called-format (*mac* | *mac:ssid* | *ssid*; Default: **mac:ssid**)	
interim-update (*time*; Default: **0**)	When RADIUS accounting is used, Access Point periodically sends accounting information updates to the RADIUS server. This property specifies default update interval that can be overridden by the RADIUS server using Acct-Interim-Interval at tribute.

radius-mac-format (*XX:XX:XX:XX:XX:XX | XXXX:XXXX:XXXX | XXXXXX:XXXXXX | XX-XX-XX-XX-XX-XX | XXXXXX-XXXXXX | XXXXXXXXXXXX | XX XX XX XX XX XX | lower case;* Default: **XX:XX:XX:XX:XX:XX**)	Controls how MAC address of the client is encoded by Access Point in the User-Name attribute of the MAC authentication and MAC accounting RADIUS requests.
radius-mac-mode (*as-username | as-username-and-password*; Default: **as-username**)	By default Access Point uses an empty password, when sending Access-Request during MAC authentication. When this property is set to *as-username-and-password*, Access Point will use the same value for User-Password attribute as for the User-Name attribute.
radius-mac-caching (*disabled | time*; Default: **disabled**)	If this value is set to time interval, the Access Point will cache RADIUS MAC authentication responses for specified time, and will not contact RADIUS server if matching cache entry already exists. Value *disabled* will disable cache, Access Point will always contact RADIUS server.

WEP properties

These properties have effect only when **mode** is set to *static-keys-required* or *static-keys-optional*.

Property	Description
static-key-0 | static-key-1 | static-key-2 | static-key-3 (*hex*; Default:)	Hexadecimal representation of the key. Length of key must be appropriate for selected algorithm. See the Statically configured WEP keys section.
static-algo-0 | static-algo-1 | static-algo-2 | static-algo-3 (*none | 40bit-wep | 104bit-wep | tkip | aes-ccm*; Default: **none**)	Encryption algorithm to use with the corresponding key.
static-transmit-key (*key-0 | key-1 | key-2 | key-3*; Default: **key-0**)	Access Point will use the specified key to encrypt frames for clients that do not use private key. Access Point will also use this key to encrypt broadcast and multicast frames. Client will use the specified key to encrypt frames if **static-sta-private-algo** is set to *none*. If corresponding **static-algo-N** property has value set to *none*, then frame will be sent unencrypted (when **mode** is set to *static-keys-optional*) or will not be sent at all (when **mode** is set to *static-keys-required*).
static-sta-private-key (*hex*; Default:)	Length of key must be appropriate for selected algorithm, see the Statically configured WEP keys section. This property is used only on Stations. Access Point uses corresponding key either from private-key property, or from Mikrotik-Wireless-Enc-Key attribute.
static-sta-private-algo (*none | 40bit-wep | 104bit-wep | tkip | aes-ccm*; Default: **none**)	Encryption algorithm to use with station private key. Value *none* disables use of the private key. This property is only used on Stations. Access Point has to get corresponding value either from private-algo property, or from Mikrotik-Wireless-Enc-Algo attribute. Station private key replaces key 0 for unicast frames. Station will not use private key to decrypt broadcast frames.

Management frame protection

Used for: *Deauthentication attack prevention, MAC address cloning issue.*

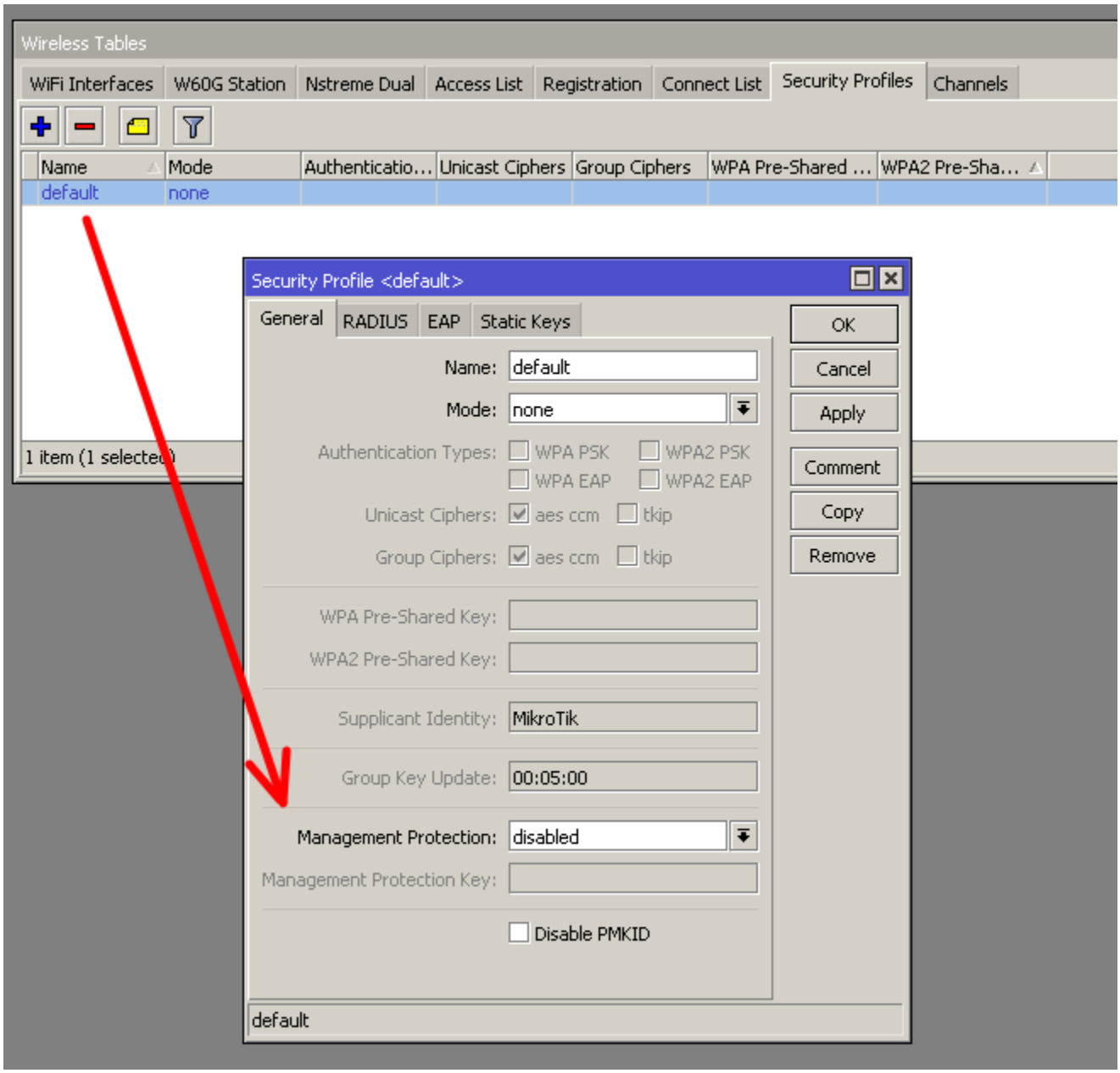
RouterOS implements proprietary management frame protection algorithm based on shared secret. Management frame protection means that RouterOS wireless device is able to verify source of management frame and confirm that particular frame is not malicious. This feature allows to withstand deauthentication and disassociation attacks on RouterOS based wireless devices.

Management protection mode is configured in security-profile with **management-protection** setting. Possible values are: **disabled** - management protection is disabled (default), **allowed** - use management protection if supported by remote party (for AP - allow both, non-management protection and management protection clients, for client - connect both to APs with and without management protection), **required** - establish association only with remote devices that support management protection (for AP - accept only clients that support management protection, for client - connect only to APs that support management protection).

Management protection shared secret is configured with security-profile **management-protection-key** setting.

When interface is in AP mode, default management protection key (configured in security-profile) can be overridden by key specified in access-list or RADIUS attribute.

```
[admin@mikrotik] /interface wireless security-profiles> print
 0 name="default" mode=none authentication-types="" unicast-ciphers=""
   group-ciphers="" wpa-pre-shared-key="" wpa2-pre-shared-key=""
   supplicant-identity="n-str-p46" eap-methods=passthrough
   tls-mode=no-certificates tls-certificate=none static-algo-0=none
   static-key-0="" static-algo-1=none static-key-1="" static-algo-2=none
   static-key-2="" static-algo-3=none static-key-3=""
   static-transmit-key=key-0 static-sta-private-algo=none
   static-sta-private-key="" radius-mac-authentication=no
   radius-mac-accounting=no radius-eap-accounting=no interim-update=0s
   radius-mac-format=XX:XX:XX:XX:XX:XX radius-mac-mode=as-username
   radius-mac-caching=disabled group-key-update=5m
   management-protection=disabled management-protection-key=""

[admin@mikrotik] /interface wireless security-profiles> set default management-protection=
                                                        allowed  disabled  required
```



Operation details

RADIUS MAC authentication

Note: RAIDUS MAC authentication is used by access point for clients that are not found in the access-list, similarly to the **default-authentication** property of the wireless interface. It controls whether client is allowed to proceed with authentication, or is rejected immediately.

When **radius-mac-authentication**=*yes*, access point queries RADIUS server by sending Access-Request with the following attributes:

- User-Name - Client MAC address. This is encoded as specified by the **radius-mac-format** setting. Default encoding is "XX:XX:XX:XX:XX:XX".
- Nas-Port-Id - **name** of wireless interface.
- User-Password - When **radius-mac-mode**=*as-username-and-password* this is set to the same value as User-Name. Otherwise this attribute is empty.
- Calling-Station-Id - Client MAC address, encoded as "XX-XX-XX-XX-XX-XX".
- Called-Station-Id - MAC address and SSID of the access point, encoded as "XX-XX-XX-XX-XX-XX:SSID" (minus separated pairs of MAC address digits, followed by colon, followed by SSID value).
- Acct-Session-Id - Added when **radius-mac-accounting**=*yes*.

When access point receives Access-Accept or Access-Reject response from the RADIUS server, it stores the response and either allows or rejects client. Access point uses following RADIUS attributes from the Access-Accept response:

- Ascend-Data-Rate
- Ascend-Xmit-Rate
- Mikrotik-Wireless-Forward - Same as access-list **forwarding**.
- Mikrotik-Wireless-Enc-Algo - Same as access-list **private-algo**.
- Mikrotik-Wireless-Enc-Key - Same as access-list **private-key**.
- Mikrotik-Wireless-Psk - Same as access-list **private-pre-shared-key**.
- Mikrotik-Wireless-Mpkey - Same as Management-protection-key in Access list
- Session-Timeout - Time, after which client will be disconnected.
- Acct-Interim-Interval - Overrides value of **interim-update**.
- Class - If present, value of this attribute is saved and included in Accounting-Request messages.

Caching

Caching of RADIUS MAC authentication was added to support RADIUS authentication for clients that require from the access point very quick response to the association request. Such clients time out before response from RADIUS server is received. Access point caches authentication response for some time and can immediately reply to the repeated association request from the same client.

RADIUS EAP pass-through authentication

When using WPA EAP authentication type, clients that have passed MAC authentication are required to perform EAP authentication before being authorized to pass data on wireless network. With pass-through EAP method the access point will relay authentication to RADIUS server, and use following attributes in the Access-Request RADIUS message:

- User-Name - EAP supplicant identity. This value is configured in the **supplicant-identity** property of the client security profile.
- Nas-Port-Id - **name** of wireless interface.
- Calling-Station-Id - Client MAC address, encoded as "XX-XX-XX-XX-XX-XX".
- Called-Station-Id - MAC address and SSID of the access point, encoded as "XX-XX-XX-XX-XX-XX:SSID" (pairs of MAC address digits separated by minus sign, followed by colon, followed by SSID value).
- Acct-Session-Id - Added when **radius-eap-accounting**=*yes*.
- Acct-Multi-Session-Id - MAC address of access point and client, and unique 8 byte value, that is shared for all accounting sessions that share single EAP authentication. Encoded as *AA-AA-AA-AA-AA-AA-CC-CC-CC-CC-CC-CC-XX-XX-XX-XX-XX-XX-XX-XX*. Added when radius-eap-accounting=yes.

Access point uses following RADIUS attributes from the Access-Accept server response:

- Class - If present, value of this attribute is saved and included in Accounting-Request messages.
- Session-Timeout - Time, after which client will be disconnected. Additionally, access point will remember authentication result, and if during this time client reconnects, it will be authorized immediately, without repeating EAP authentication.
- Acct-Interim-Interval - Overrides value of **interim-update**.

Statically configured WEP keys

Different algorithms require different length of keys:

- *40bit-wep* - 10 hexadecimal digits (40 bits). If key is longer, only first 40 bits are used.
- *104bit-wep* - 26 hexadecimal digits (104 bits). If key is longer, only first 104 bits are used.
- *tkip* - At least 64 hexadecimal digits (256 bits).
- *aes-ccm* - At least 32 hexadecimal digits (128 bits).

Key must contain even number of hexadecimal digits.

WDS security configuration

WDS links can use all available security features. However, they require careful configuration of security parameters.

It is possible to use one security profile for all clients, and different security profiles for WDS links. Security profile for WDS link is specified in connect-list. Access point always checks connect list before establishing WDS link with another access point, and used security settings from matching connect list entry. WDS link will work when each access point will have connect list entry that matches the other device, has **connect**=*yes* and specifies compatible **security-profile**.

WDS and WPA/WPA2

If access point uses security profile with **mode**=*dynamic-keys*, then encryption will be used for all WDS links. Since WPA authentication and key exchange is not symmetrical, one of the access points will act as a client for the purpose of establishing secure connection. This is similar to how *static-mesh* and *dynamic-mesh* WDS modes work. Some problems, like single sided WDS link between two incorrectly configured access points that use non-*mesh* mode, is not possible if WPA encryption is enabled. However, non-*mesh* modes with WPA still have other issues (like constant reconnection attempts in case of configuration mismatch) that are solved by use of the *-mesh* WDS modes.

In general, WPA properties on both access points that establish WPA protected WDS link have to match. These properties are **authentication-types**, **unicast-ciphers**, **group-ciphers**. For non-*mesh* WDS mode these properties need to have the same values on both devices. In *mesh* WDS mode each access point has to support the other one as a client.

Theoretically it is possible to use RADIUS MAC authentication and other RADIUS services with WDS links. However, only one access point will interact with the RADIUS server, the other access point will behave as a client.

Implementation of *eap-tls* EAP method in RouterOS is particularly well suited for WDS link encryption. **tls-mode**=*no-certificates* requires no additional configuration, and provides very strong encryption.

WDS and WEP

mode, **static-sta-private-key** and **static-sta-private-algo** parameters in the security profile assigned to the WDS link need to have the same values on both access points that establish WDS link with WPA encryption.

Security profile and access point matching in the connect list

Client uses value of connect-list **security-profile** property to match only those access points that support necessary security.

- **mode**=*static-keys-required* and **mode**=*static-keys-optional* matches only access points with the same **mode** in interface **security-profile**.
- If **mode**=*dynamic-keys*, then connect list entry matches if all of the **authentication-types**, **unicast-ciphers** and **group-ciphers** contain at least one value that is advertised by access point.

Virtual interfaces

VirtualAP

It is possible to create virtual access points using the *add* command in the wireless menu. You must specify the *master-interface* which the virtual interface will belong to. If "master-interface" mode is "station", Virtual AP will work only when "master-interface" will be active. The Virtual AP can have it's own SSID and Security Profile.

Virtual AP interface will only work if master interface is in *ap-bridge*, *bridge*, *station* or wds-slave mode. It works only with 802.11 protocol, Nv2 is not supported.

This feature is useful for separating access for different types of users. You can assign different bandwidth levels and passwords and instruct users to connect to the specific virtual network, it will appear to wireless clients as a different SSID or a different device. For example, when using QuickSet to configure a guest network, the VirtualAP feature is used in the background.

To create a new virtual-ap: `/interface> wireless add mode=ap-bridge master-interface=wlan1 ssid=guests security-profile=guests` (*such security profile first needs to be created*)

Note: you can create up to 127 virtual interfaces per physical interface. It is not recommended to create more 30, since the performance will start to degrade.

Virtual Clients

Note: Starting from 6.35 only in wireless-rep or wireless-cm2 package

It is also possible to create virtual clients and have both an AP and a Client on the same physical interface. This allows to make a repeater setup with only using one hardware card. The process of configuration is exacly the same as above, but use mode **station**:

To create a new virtual-client: `/interface> wireless add mode=station master-interface=wlan1 ssid=where-to-connect security-profile=your-profile` (*such security profile first needs to be created*)

Note: Virtual interfaces will always use the Master interface wireless frequency. If the Master interface has 'auto' frequency enabled it will use the wireless frequency that the Master interface selected.

Sniffer

Sub-menu: `/interface wireless sniffer`

Wireless sniffer allows to capture frames including Radio header, 802.11 header and other wireless related information.

Property	Description
channel-time (; Default: **200ms**)	How long to sniff each channel. Used only if **multiple-channels=yes**
file-limit (*integer [10..4294967295]*; Default: **10**)	Allocated file size in bytes which will be used to store captured data. Applicable if **file-name** is specified.
file-name (*string*; Default:)	Name of the file where to store captured data.
memory-limit (*integer [10..4294967295]*; Default: **10**)	Allocated memory buffer in kilobytes used to store captured data.
multiple-channels (*yes* / *no*; Default: **no**)	Whether to sniff multiple channels or a single channel. **No** means that all channel settings will be taken from **/interface wireless**, **Yes** means that all channel settings will be taken from **scan-list** under **/interface wireless**.
only-headers (*yes* / *no*; Default: **no**)	If set to yes, then sniffer will capture only information stored in frame headers.
receive-errors (*yes* / *no*; Default: **no**)	Whether to process packets which have been received with errors judging by their FCS.
streaming-enabled (*yes* / *no*; Default: **no**)	Whether to stream captured data to the specified streaming server
streaming-max-rate (*integer [0..4294967295]* ; Default: **0**)	Maximum packets per second allowed. **0** equals unlimited
streaming-server (*IPv4*; Default: **0.0.0.0**)	IP address of the streaming server.

Use the command **/interface wireless info scan-list** to verify your **scan-list** defined under **/interface wireless channels** when using **multiple-channels=yes**

Packets

Sub-menu: `/interface wireless sniffer packet`

Sub-menu shows captured packets.

Scan

Scan command allows to see available AP in the frequency range defined in the scan-list. Using scan command the interface operation is disabled (wireless link is disconnected during the scan operation) Since RouterOS v6.35 (wireless-rep) background scan is supported which can be used during the wireless interface operation without disconnecting the wireless link. Background scan is supported only using 802.11 wireless protocol.

Scan tool will continue scanning for AP until user stops the scan process. It is possible to use 'rounds' setting for the scan tool to do scan through the scan-list entries specific times. It is useful when running scan tool using scripts. Example of scan command for one round:

```
/interface wireless scan wlan1 rounds=1
```

'save-file' option allows to do scripted/scheduled scans and save the results in file for future analysis. Also this feature together with rounds setting allows to get scan results from the remote wireless clients - executing that command will start the scan tool which disconnect the wireless link, does the scan through the scan-list frequencies and saves the results to file, exits the scan and connects the wireless link back. Example:

```
/interface wireless scan wlan1 rounds=1 save-file=scan1
```

To use background wireless scan the 'background=yes' setting should be provided. Example:

```
/interface wireless scan wlan1 background=yes
```

Background scan feature is working in such conditions:

- Wireless interface should be enabled
- For wireless interface in AP mode - when it is operating in 802.11 protocol mode and is on fixed channel (that is - channel selection and initial radar checking is over)
- For wireless interface in Station mode - when it is connected to 802.11 protocol AP.

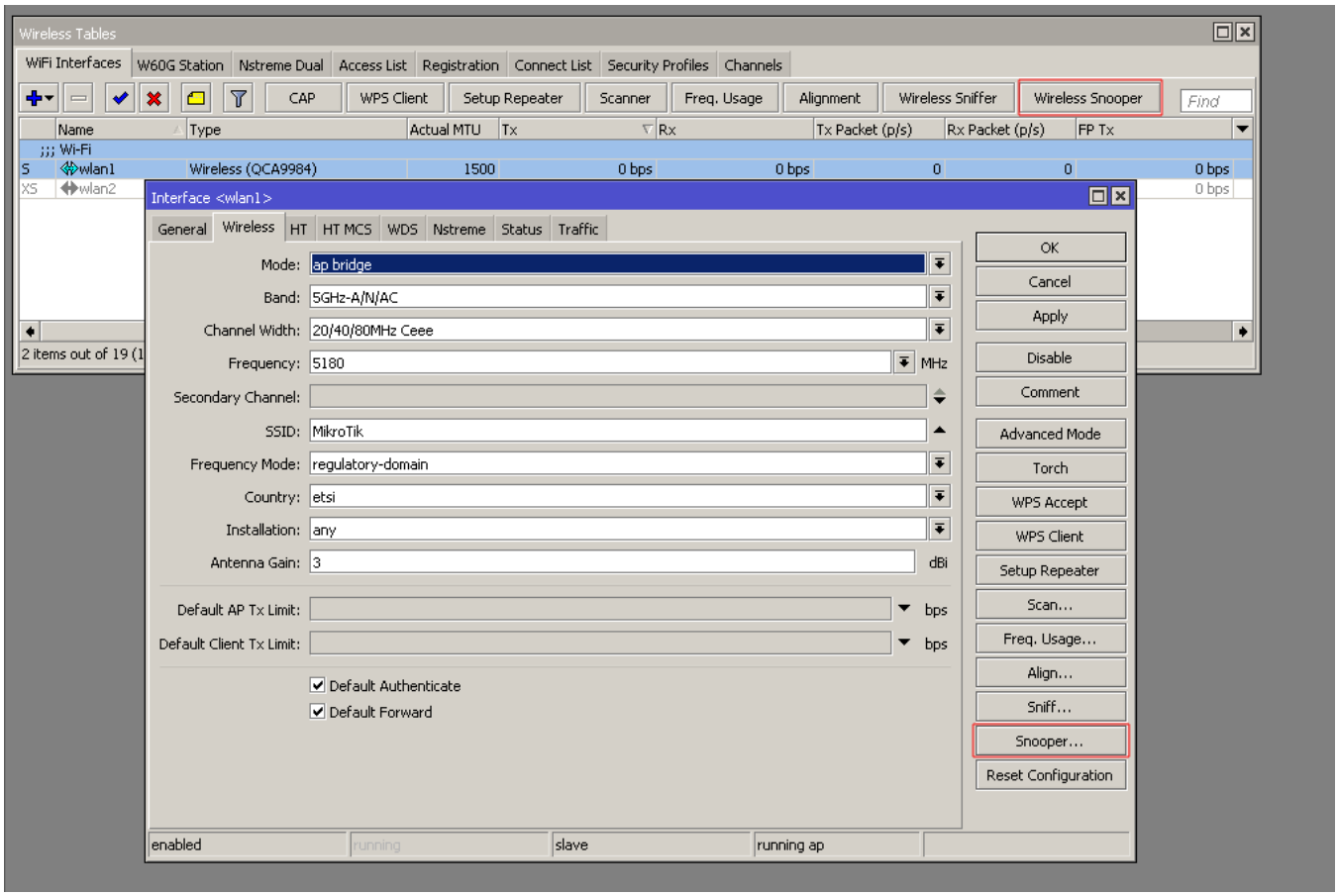
Scan command is supported also on the Virtual wireless interfaces with such limitations:

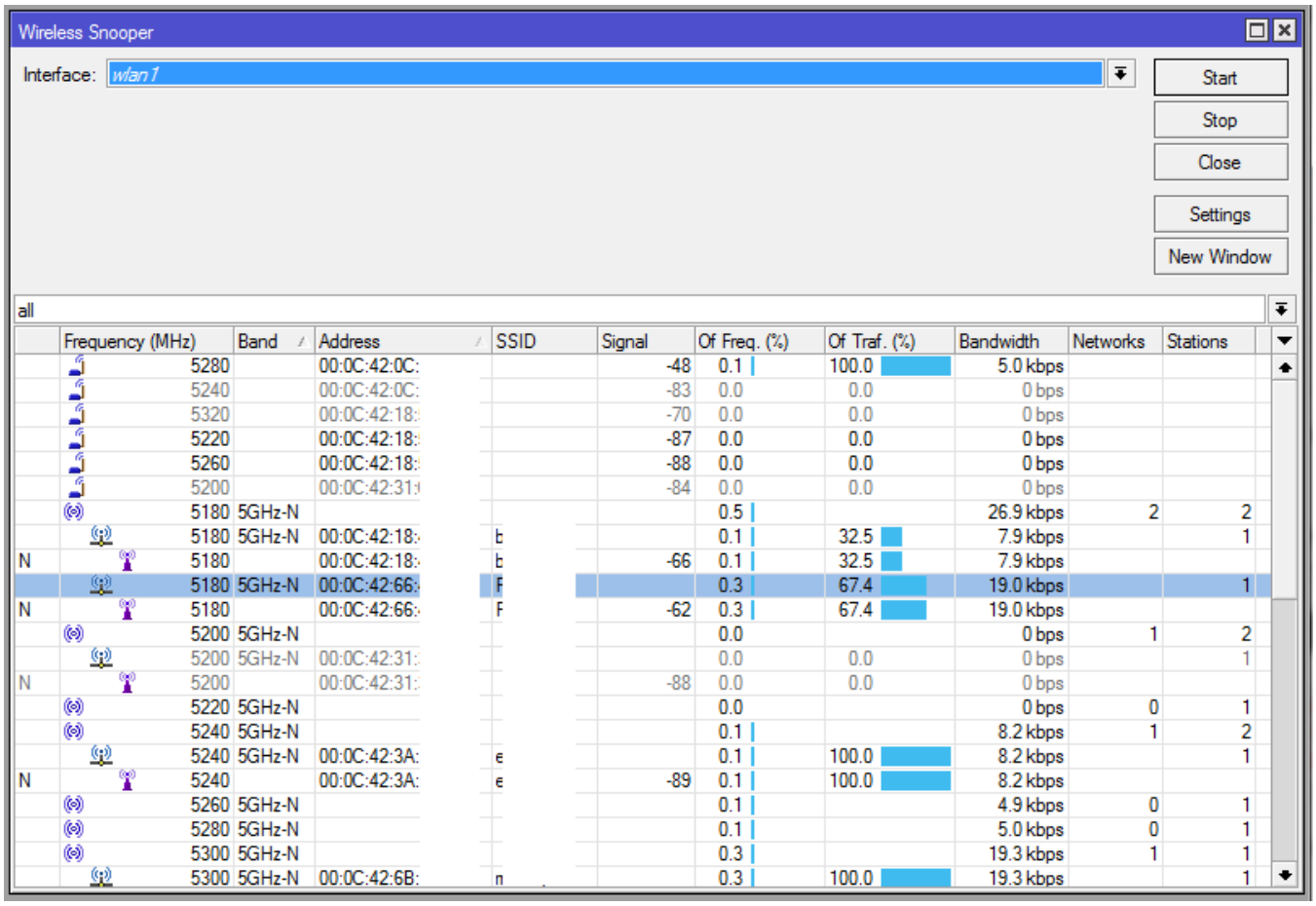
- It is possible when virtual interface and its master is fixed on channel (master AP is running or master station is connected to AP).
- Scan is only performed in channel master interface is on.
- It does not matter if background=yes|no - on virtual interface scan does not disconnect clients/AP, so it is always "background".

Snooper

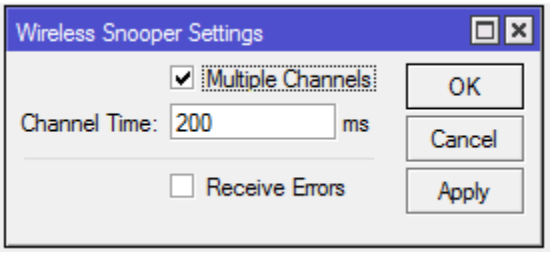
This tool monitors surrounding frequency usage, and displays which devices occupy each frequency. It's available both in console, and also in Winbox. Snooper will use frequencies from scan-list.

Sub-menu: `/interface wireless snooper`


Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
CAP
WPS Client
Setup Repeater
Scanner
Freq. Usage
Alignment
Wireless Sniffer
Wireless Snooper
Find
Name
Type
Actual MTU
Tx
Rx
Tx Packet (p/s)
Rx Packet (p/s)
FP Tx
;;; Wi-Fi
S wlan1 Wireless (QCA9984) 1500 0 bps 0 bps 0 0 0 bps
XS wlan2 0 bps
2 items out of 19 (1
Interface <wlan1>
General
Wireless
HT
HT MCS
WDS
Nstreme
Status
Traffic
Mode: ap bridge
Band: 5GHz-A/N/AC
Channel Width: 20/40/80MHz Ceee
Frequency: 5180 MHz
Secondary Channel:
SSID: MikroTik
Frequency Mode: regulatory-domain
Country: etsi
Installation: any
Antenna Gain: 3 dBi
Default AP Tx Limit: bps
Default Client Tx Limit: bps
Default Authenticate
Default Forward
OK
Cancel
Apply
Disable
Comment
Advanced Mode
Torch
WPS Accept
WPS Client
Setup Repeater
Scan...
Freq. Usage...
Align...
Sniff...
Snooper...
Reset Configuration
enabled
running
slave
running ap



Settings



Spectral scan

- See separate document Manual:Spectral_scan

WDS

Sub-menu: `/interface wireless wds`

Properties:

Property	Description
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	
comment (*string*; Default:)	
disable-running-check (*yes* | *no*; Default: **no**)	

disabled (*yes* | *no*; Default: **yes**)	
l2mtu (*integer [0..65536]*; Default:)	
master-interface (*string*; Default:)	
mtu (*integer [0..65536]*; Default: **1500**)	
name (*string*; Default:)	
wds-address (*MAC*; Default: **00:00:00:00:00:00**)	

Read-only properties:

Property	Description
dynamic (*yes* | *no*)	
mac-address (*MAC*)	
running (*yes* | *no*)	

WPS

Wireless interface supports WPS Server and also WPS Client (supported by wireless-rep package starting from RouterOS v6.35).

WPS Server

WPS Server allows to connect wireless clients that support WPS to AP protected with the Pre-Shared Key without specifying that key in the clients configuration.

WPS Server can be enabled by changing the WPS Mode setting for the wireless interface. Example:

```
/interface wireless set wlan1 wps-mode=push-button
```

Wps-mode has 3 options

- disabled
- push-button - WPS is activated by pushing physical button on the board (few boards has such button marked on the board case/label)
- push-button-virtual-only - WPS is activated by pushing "WPS Accept" button from the RouterOS wireless interface menu

By pushing the WPS physical/virtual button the AP enables the WPS functionality. If within 2 minutes the WPS process isn't initiated the WPS Accept Function is stopped.

WPS Server is enabled by default on few boards that has physical WPS button marked. For example, hap lite, hap, hap ac lite, hap ac, map lite

WPS Server is active only when wireless AP interface has Pre-Shared Key Authentication (PSK) enabled. It is possible to configure this mode for the Virtual AP interfaces as well.

WPS Client

WPS Client function allows the wireless client to get the Pre-Shared Key configuration of the AP that has WPS Server enabled. WPS Client can be enabled by such command:

```
/interface wireless wps-client wlan1
```

WPS Client command outputs all the information of the WPS Enabled AP on the screen. Example:

```
[admin@MikroTik] /interface wireless> wps-client wlan1
          status: disconnected, success
            ssid: MikroTik
     mac-address: E4:8D:8C:D6:E0:AC
      passphrase: presharedkey
  authentication: wpa2-psk
      encryption: aes-ccm
```

It is possible to specify additional settings for the WPS-Client command:

- create-profile - creates wireless security profile with the specified name, configures it with security details received from the WPS AP, specifies the wireless interface to use the new created security profile
- ssid - get WPS information only from AP with specified SSID
- mac-address - get WPS information only from AP with specified mac-address

Repeater

Wireless repeater will allow to receive the signal from the AP and repeat the signal using the same physical interface locally for connecting other clients. This will allow to extend the wireless service for the wireless clients. Wireless repeater function will configure the wireless interface to connect to the AP with station-bridge or station-pseudobridge option, create a virtual AP interface, create a bridge interface and add both (main and the virtual) interfaces to the bridge ports.

If your AP **supports button-enabled WPS** mode, you can use the automatic setup command:

```
/interface wireless setup-repeater wlan1
```

The setup-repeater does the following steps:

- searches for WPS AP with button pushed
- acquires SSID, key, channel from AP
- resets main master interface config (same as reset-configuration)
- removes all bridge ports that were added for virtual interfaces added to this master (so there are no dangling invalid bridge ports later)
- removes all virtual interfaces added to this master
- creates security profile with name "<interfacename>-<ssid>-repeater", if such security profile already exists does not create new, just updates settings
- configures master interface, interface mode is selected like this: if AP supports bridge mode, use station-bridge, else if AP supports WDS, use station-wds, else use station-pseudobridge
- creates virtual AP interface with same SSID and security profile as master
- if master interface is not in some bridge, creates new bridge interface and adds master interface to it
- adds virtual AP interface to the same bridge master interface is in.

If your AP **does not support WPS**, it is possible to specify the settings manually, using these parameters:

- **address** - MAC address of AP to setup repeater for (optional)
- **ssid** - SSID of AP to setup repeater for (optional)
- **passphrase** - key to use for AP - if this IS specified, command will just scan for AP and create security profile based on info in beacon and with this passphrase. If this IS NOT specified, command will do WPS to find out passphrase.

Roaming

Station Roaming

Station Roaming feature is available only for 802.11 wireless protocol and only for station modes. When RouterOS wireless client is connected to the AP using 802.11 wireless protocol it will periodically perform the background scan with specific time intervals. When the background scan will find an AP with better signal it will try to roam to that AP. The time intervals between the background scans will become shorter when the wireless signal becomes worse and the background scan interval will become longer when the wireless client signal will get better.

VLAN tagging

Sub-menu: `/interface wireless`

With VLAN tagging it is possible to separate Virtual AP traffic on Ethernet side of "locally forwarding" AP (the one on which wireless interfaces are bridged with Ethernet). This is necessary to separate e.g. "management" and "guest" network traffic of Ethernet side of APs.

VLAN is assigned for wireless interface and as a result all data coming from wireless gets tagged with this tag and only data with this tag will send out over wireless. This works for all wireless protocols except that on Nv2 there's no Virtual AP support.

You can configure your RADIUS authentication server to assign users or groups of users to a specific VLAN when they authenticate to the network. To use this option you will need to use RADIUS attributes.

Note: In case to use this option you must enable wireless-fp or wireless-cm2 package for RouterOS version up to 6.37. Starting from RouterOS v6.37 you can do that with regular wireless package.

Property	Description
vlan-mode (*no tag* | *user service tag* | *use tag*; Default: **no tag**)	Three VLAN modes are available: • *no-tag* - AP don't use VLAN tagging • *use-service-tag* - VLAN ID use 802.1ad tag type • *use-tag* - VLAN ID use 802.1q tag type
vlan-id (*integer [1..4095]*; Default: **1**)	VLAN identification number

Vlan tag override

Per-interface VLAN tag can be overridden on per-client basis by means of access-list and RADIUS attributes (for both - regular wireless and wireless controller).

This way traffic can be separated between wireless clients even on the same interface, but must be used with care - only "interface VLAN" broadcast /multicast traffic will be sent out. If working broadcast/multicast is necessary for other (overridden) VLANs as well, multicast-helper can be used for now (this changes every multicast packet to unicast and then it is only sent to clients with matching VLAN ids).

Winbox

Winbox is a small utility that allows the administration of Mikrotik RouterOS using a fast and simple GUI.

Note: Current Tx Power gives you information about transmit power currently used at specific data rate. Currently not supported for Atheros 802.11ac chips (e.g. QCA98xx).

Interworking Realms setting

For more information about interworking-profiles see the manual.

realms-raw - list of strings with hex values. Each string specifies contents of "NAI Realm Tuple", excluding "NAI Realm Data Field Length" field.

Each hex encoded string must consist of the following fields:

```
- NAI Realm Encoding (1 byte)
- NAI Realm Length (1 byte)
- NAI Realm (variable)
- EAP Method Count (1 byte)
- EAP Method Tuples (variable)
```

For example, value "00045465737401020d00" decodes as:

```
- NAI Realm Encoding: 0 (rfc4282)
- NAI Realm Length: 4
- NAI Realm: Test
- EAP Method Count: 1
- EAP Method Length: 2
- EAP Method Tuple: TLS, no EAP method parameters
```

Note, that setting "realms-raw=00045465737401020d00" produces the same advertisement contents as setting "realms=Test:eap-tls".

Refer to 802.11-2016, section 9.4.5.10 for full NAI Realm encoding.

WifiWave2 (7.12 and older)

Overview

This document applies to **7.12 and older**. The WifiWave2 package contains software for managing compatible 802.11ax and 802.11ac wave 2 wireless interfaces. New versions use the new wifi package and corresponding manual.

Builds for x86, ppc, mmips and tile architectures contain the configuration utilities needed to centrally manage interfaces (as a CAPsMAN controller). Builds for arm and arm64 also contain interface drivers and firmware.

The package can be downloaded as part of the 'Extra Packages' archive.

The WifiWave2 package in RouterOS adds certain Wave2 features, and 802.11ax devices require it. Some products which ship with the standard 'wireless' package, can replace it with wifiwave2, for more details, please see this section.

Configuration in the command line is done under /interface/wifiwave2/, when using a graphical configuration tool (WinBox or WebFig), wifiwave2 interfaces can be configured using either the 'Wireless' or 'QuickSet' tabs.

WifiWave2 Terminology

Before we move on let's familiarise ourselves with terms important for understanding the operation of the WifiWave2. These terms will be used throughout the article.

- **Profile** - refers to the configuration preset created under one of this WifiWave2 sub-menus: **aaa**, **channel**, **security**, **datapath**, or **interworking**.
- **Configuration profile** - configuration preset defined under /interface/wifiwave2/configuration, it can reference various profiles.
- **Station** - wireless client.

Basic Configuration:

Basic password-protected AP

```
/interface/wifiwave2
set wifi1 disabled=no configuration.country=Latvia configuration.ssid=MikroTik security.authentication-
types=wpa2-psk,wpa3-psk security.passphrase=8-63_characters
```

Open AP with OWE transition mode

Opportunistic wireless encryption (OWE) allows the creation of wireless networks that do not require the knowledge of a password to connect, but still offer the benefits of traffic encryption and management frame protection. It is an improvement on regular open access points.

However, since a network cannot be simultaneously encrypted and unencrypted, 2 separate interface configurations are required to offer connectivity to older devices that do not support OWE and offer the benefits of OWE to devices that do.

This configuration is referred to as OWE transition mode.

```
/interface/wifiwave2
add master-interface=wifi1 name=wifi1_owe configuration.ssid=MikroTik_OWE security.authentication-types=owe
security.owe-transition-interface=wifi1 configuration.hide-ssid=yes
set wifi1 configuration.country=Latvia configuration.ssid=MikroTik security.authentication-types="" security.
owe-transition-interface=wifi1_owe
enable wifi1,wifi1_owe
```

Client devices that support OWE will prefer the OWE interface. If you don't see any devices in your registration table that are associating with the regular open AP, you may want to move on from running a transition mode setup to a single OWE-encrypted interface.

Resetting configuration

WifiWave2 interface configurations can be reset by using the 'reset' command.

```
/interface/wifiwave2 reset wifi1
```

Configuration profiles

One of the new WifiWave2 additions is configuration profiles, you can create various presets, that can be assigned to interfaces as needed. Configuration settings for WifiWave2 are grouped in **profiles** according to the parameter sections found at end of this page - **aaa**, **channel**, **configuration**, **datapath**, **interworking**, and **security**, and can then be assigned to interfaces. **Configuration profiles** can include other profiles as well as separate parameters from other categories.

This optional flexibility is meant to allow each user to arrange their configuration in a way that makes the most sense for them, but it also means that each parameter may have different values assigned to it in different sections of the configuration.

The following priority determines, which value is used:

1. Value in interface settings
2. Value in a profile assigned to interface

3. Value in configuration profile assigned to interface
4. Value in a profile assigned to configuration profile (which in turn is assigned to interface).

If you are at any point unsure of which parameter value will be used for an interface, consult the **actual-configuration** menu. For an example of configuration profile usage, see the following example.

Example for dual-band home AP

```
# Creating a security profile, which will be common for both interfaces
/interface wifiwave2 security
add name=common-auth authentication-types=wpa2-psk,wpa3-psk passphrase="diceware makes good passwords"
wps=disable
# Creating a common configuration profile and linking the security profile to it
/interface wifiwave2 configuration
add name=common-conf ssid=MikroTik country=Latvia security=common-auth
# Creating separate channel configurations for each band
/interface wifiwave2 channel
add name=ch-2ghz frequency=2412,2432,2472 width=20mhz
add name=ch-5ghz frequency=5180,5260,5500 width=20/40/80mhz
# Assigning to each interface the common profile as well as band-specific channel profile
/interface wifiwave2
set wifi1 channel=ch-2ghz configuration=common-conf disabled=no
set wifi2 channel=ch-5ghz configuration=common-conf disabled=no

/interface/wifiwave2/actual-configuration print
 0 name="wifi1" mac-address=74:4D:28:94:22:9A arp-timeout=auto radio-mac=74:4D:28:94:22:9A
   configuration.ssid="MikroTik" .country=Latvia
   security.authentication-types=wpa2-psk,wpa3-psk .passphrase="diceware makes good passwords" .wps=disable
   channel.frequency=2412,2432,2472 .width=20mhz

 1 name="wifi2" mac-address=74:4D:28:94:22:9B arp-timeout=auto radio-mac=74:4D:28:94:22:9B
   configuration.ssid="MikroTik" .country=Latvia
   security.authentication-types=wpa2-psk,wpa3-psk .passphrase="diceware makes good passwords" .wps=disable
   channel.frequency=5180,5260,5500 .width=20/40/80mhz
```

Access List

The access list provides multiple ways of filtering and managing wireless connections.

RouterOS will check each new connection to see if its parameters match the parameters specified in any access list rule.

The rules are checked in the order they appear in the list. Only management actions specified in the first matching rule are applied to each connection.

Connections, which have been accepted by an access list rule, will be periodically checked, to see if they remain within the permitted **time** and **signal-range**. If they do not, they will be terminated.

Take care when writing access list rules which reject clients. After being repeatedly rejected by an AP, a client device may start avoiding it.

The access list has two kinds of parameters - filtering, and action. Filtering properties are only used for matching clients, to whom the access list rule should be applied to. Action parameters can change connection parameters for that specific client and potentially overriding its default connection parameters with ones specified in the access list rule.

MAC address authentication

Implemented through the **query-radius** action, MAC address authentication is a way to implement a centralized whitelist of client MAC addresses using a RADIUS server.

When a client device tries to associate with an AP, which is configured to perform MAC address authentication, the AP will send an access-request message to a RADIUS server with the device's MAC address as the user name and an empty password. If the RADIUS server answers with access-accept to such a request, the AP proceeds with whatever regular authentication procedure (passphrase or EAP authentication) is configured for the interface.

Access rule examples

Only accept connections to guest network from nearby devices during business hours

```
/interface/wifiwave2/access-list/print detail
Flags: X - disabled
 0   signal-range=-60..0 allow-signal-out-of-range=5m ssid-regexp="MikroTik Guest" time=7h-19h,mon,tue,wed,thu,
fri action=accept

 1   ssid-regexp="MikroTik Guest" action=reject
```

Reject connections from locally-administered ('anonymous'/'randomized') MAC addresses

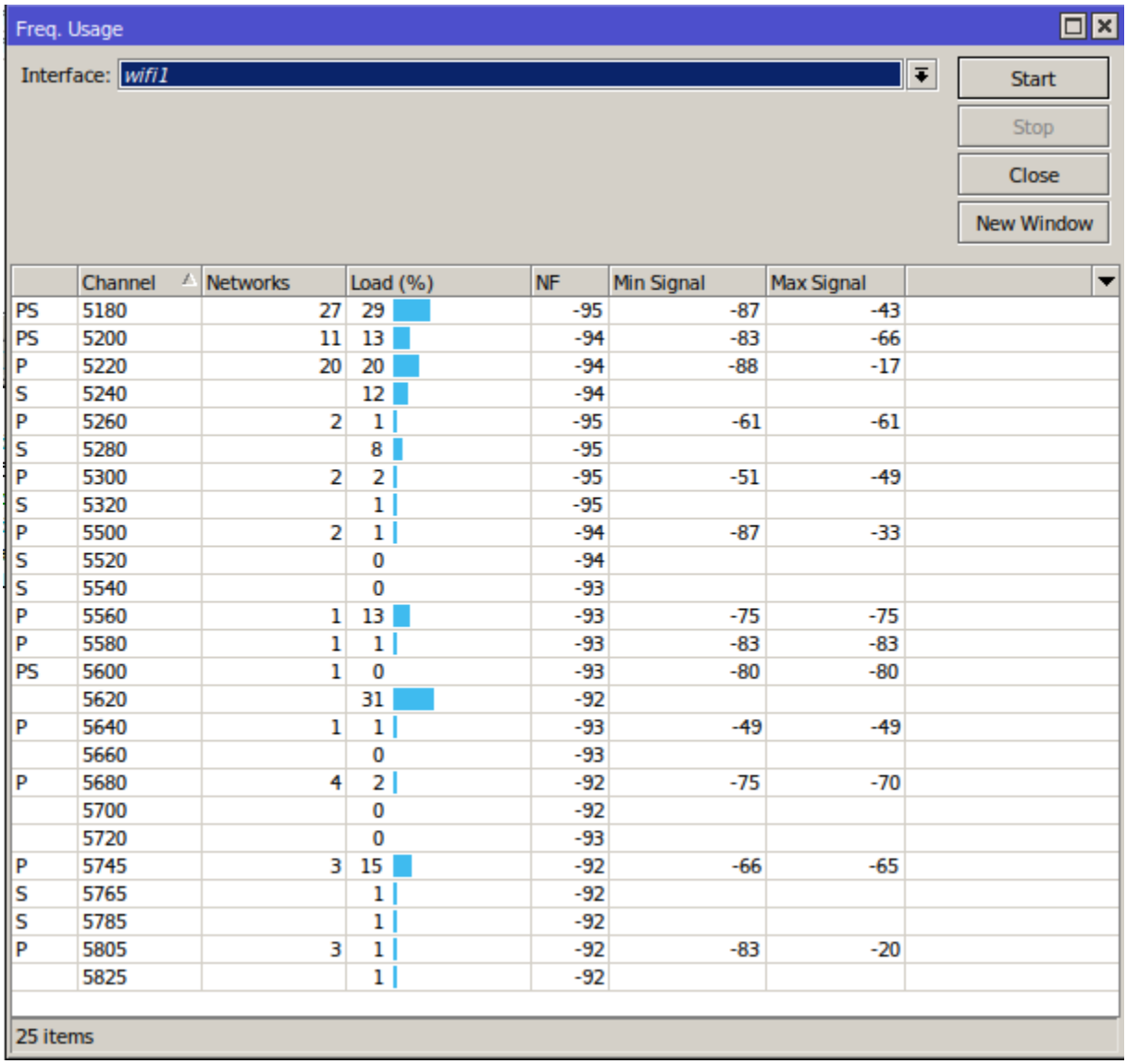
```
/interface/wifiwave2/access-list/print detail
Flags: X - disabled
 0   mac-address=02:00:00:00:00:00 mac-address-mask=02:00:00:00:00:00 action=reject
```

Assigning a different passphrase for a specific client can be useful, if you need to provide wireless access to a client, but don't want to share your wireless password, or don't want to create a separate SSID. When the matching client will connect to this network, instead of using the password defined in the interface configuration, the access list will make that client use a different password. Just make that the specific client doesn't get matched by a more generic access list rule first.

```
/interface wifiwave2 access-list
add action=accept disabled=no mac-address=22:F9:70:E5:D2:8E interface=wifi1 passphrase=StrongPassword
```

Frequency scan

The '/interface/wifiwave2/frequency-scan wifi1' command provides information about RF conditions on available channels that can be obtained by running the frequency-scan command. Used to approximate the spectrum usage, it can be useful to find less crowded frequencies.

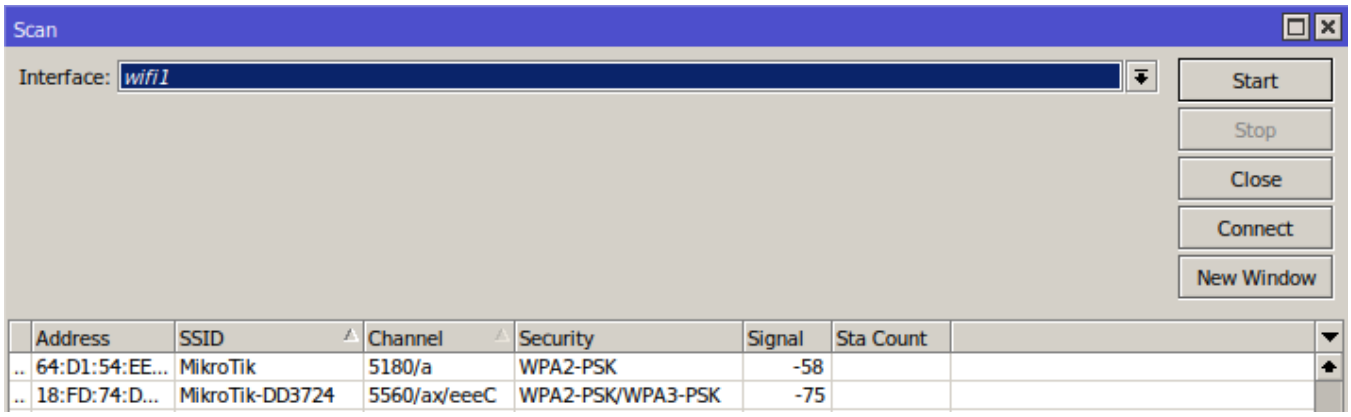


Running a frequency scan will disconnect all connected clients, or if the interface is in station mode, it will disconnect from AP.

Scan command

The '/interface wifiwave2 scan' command will scan for access points and print out information about any APs it detects. It doesn't show the frequency usage, per channel, but it will reveal all access points that are transmitting. You can use the "connect" button, to initiate a connection to a specific AP.

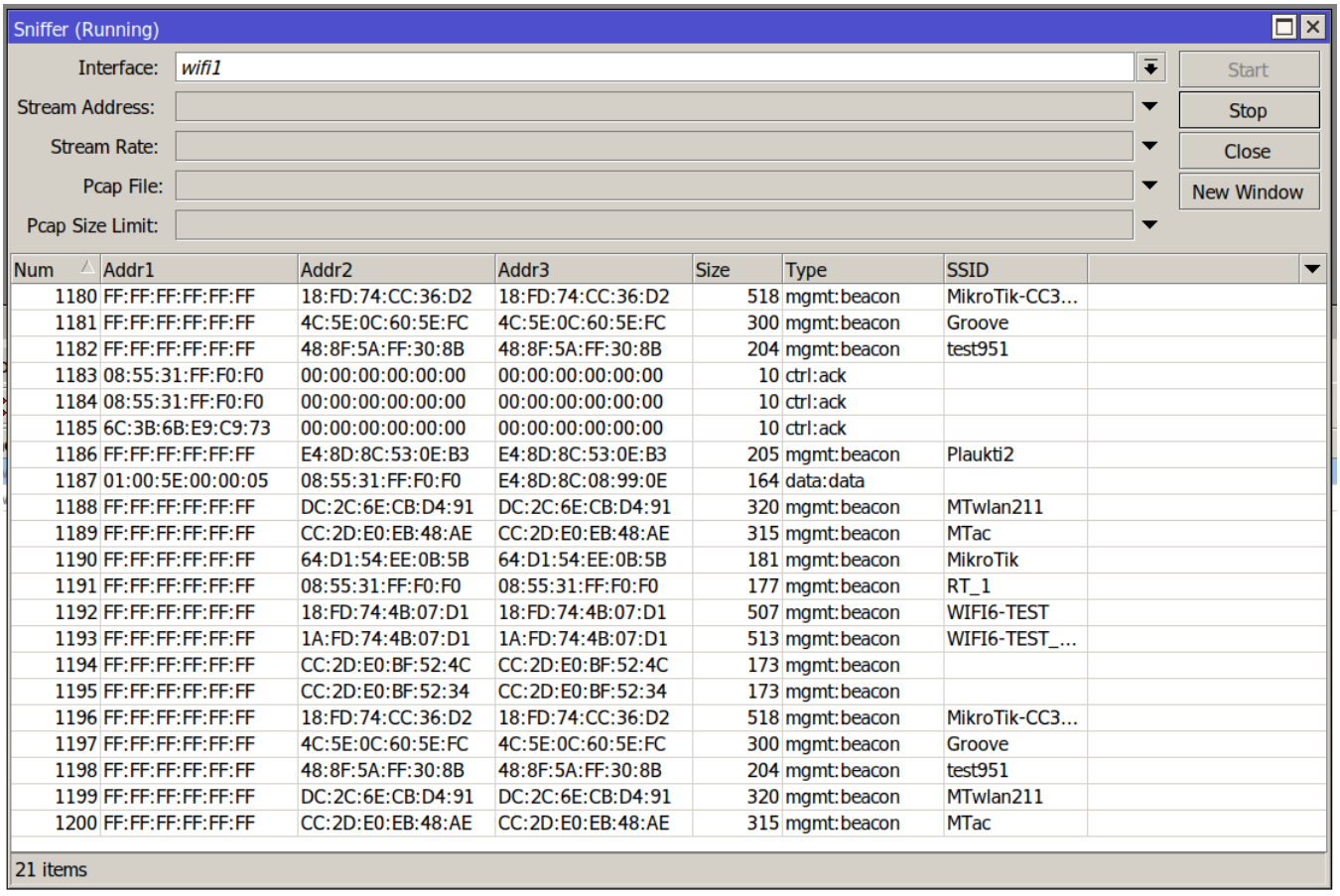
The scan command takes all the same parameters as the frequency-scan command.



Sniffer

The sniffer command enables monitor mode on a wireless interface. This turns the interface into a passive receiver for all WiFi transmissions.
The command continuously prints out information on received packets and can save them locally to a pcap file or stream them using the TZSP protocol.

The sniffer will operate on whichever channel is configured for the chosen interface.



WPS

WPS client

The wps-client command enables obtaining authentication information from a WPS-enabled AP.

```
/interface/wifiwave2/wps-client wifi1
```

WPS server

An AP can be made to accept WPS authentication by a client device for 2 minutes by running the following command.

```
/interface/wifiwave2 wps-push-button wifi1
```

Radios

Information about the capabilities of each radio can be gained by running the `/interface/wifiwave2/radio print detail` command. It can be useful to see what bands are supported by the interface and what channels can be selected. The country profile that is applied to the interface will influence the results.

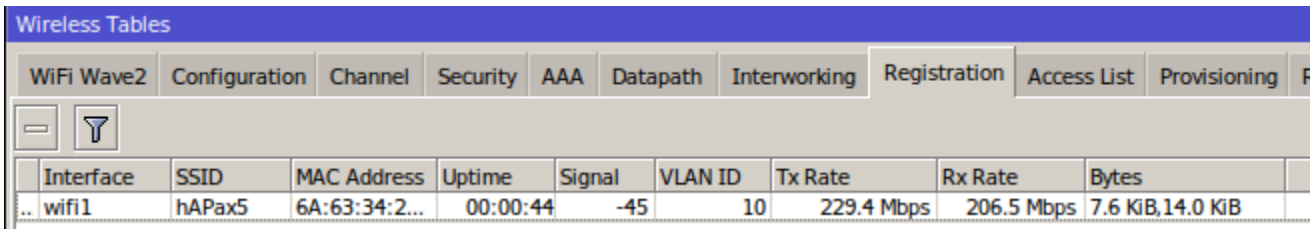
```
interface/wifiwave2/radio/print detail
Flags: L - local
 0 L radio-mac=48:A9:8A:0B:F7:4A phy-id=0 tx-chains=0,1 rx-chains=0,1
     bands=5ghz-a:20mhz,5ghz-n:20mhz,20/40mhz,5ghz-ac:20mhz,20/40mhz,20/40/80mhz,5ghz-ax:20mhz,
      20/40mhz,20/40/80mhz
     ciphers=tkip,ccmp,gcmp,ccmp-256,gcmp-256,cmac,gmac,cmac-256,gmac-256 countries=all
     5g-channels=5180,5200,5220,5240,5260,5280,5300,5320,5500,5520,5540,5560,5580,5600,5620,5640,5660,
            5680,5700,5720,5745,5765,5785,5805,5825
     max-vlans=128 max-interfaces=16 max-station-interfaces=3 max-peers=120 hw-type="QCA6018"
     hw-caps=sniffer interface=wifi1 current-country=Latvia
     current-channels=5180/a,5180/n,5180/n/Ce,5180/ac,5180/ac/Ce,5180/ac/Ceee,5180/ax,5180/ax/Ce,
                 5180/ax/Ceee,5200/a,5200/n,5200/n/eC,5200/ac,5200/ac/eC,5200/ac/eCee,5200/ax...
                 ...5680/n/eC,5680/ac,5680/ac/eC,5680/ax,5680/ax/eC,5700/a,5700/n,5700/ac,5700/ax
     current-gopclasses=115,116,128,117,118,119,120,121,122,123 current-max-reg-power=30
```

While Radio information gives us information about supported channel width, it is also possible to deduce this information from the product page, to do so you need to check the following parameters: **number of chains**, **max data rate**. Once you know these parameters, you need to check the modulation and coding scheme (MCS) table, for example, here: https://mcsindex.com/.

If we take hAP ax^2, as an example, we can see that number of chains is 2, and the max data rate is 1200 - 1201 in the MCS table. In the MCS table we need to find entry for 2 spatial streams - chains, and the respective data rate, which in this case shows us that 80MHz is the maximum supported channel width.

Registration table

'/interface/wifiwave2/registration-table/' displays a list of connected wireless clients and detailed information about them.



De-authentication

Wireless peers can be manually de-authenticated (forcing re-association) by removing them from the registration table.

```
/interface/wifiwave2/registration-table remove [find where mac-address=02:01:02:03:04:05]
```

WifiWave2 CAPsMAN

WifiWave2 CAPsMAN allows applying wireless settings to multiple MikroTik WifiWave2 AP devices from a central configuration interface.

More specifically, the Controlled Access Point system Manager (CAPsMAN) allows the centralization of wireless network management. When using the CAPsMAN feature, the network will consist of a number of 'Controlled Access Points' (CAP) that provide wireless connectivity and a 'system Manager' (CAPsMAN) that manages the configuration of the APs, it also takes care of client authentication.

WifiWave2 CAPsMAN only passes wireless configuration to the CAP, all forwarding decisions are left to the CAP itself - there is no CAPsMAN forwarding mode.

Requirements:

- Any RouterOS device, that supports the WifiWave2 package, can be a controlled wireless access point (CAP) as long as it has at least a Level 4 RouterOS license.
- WifiWave2 CAPsMAN server can be installed on any RouterOS device that supports the WifiWave2 package, even if the device itself does not have a wireless interface
- Unlimited CAPs (access points) supported by CAPsMAN

> WifiWave2 CAPsMAN can only control WifiWave2 interfaces, and WifiWave2 CAPs can join only WifiWave2 CAPsMAN, similarly, regular CAPsMAN only supports non-WifiWave2 caps.

CAPsMAN - CAP simple configuration example:

CAPsMAN in WifiWave2 uses the same menu as a regular WifiWave2 interface, meaning when you pass configuration to CAPs, you have to use the same configuration, security, channel configuration, etc. as you would for regular WifiWave2 interfaces.

> You can configure sub configuration menus, directly under "/interface/wifiwave2/configuration" or reference previously created profiles in the main configuration profile

CAPsMAN:

```
#create a security profile
/interface wifiwave2 security
add authentication-types=wpa3-psk name=sec1 passphrase=HaveAg00dDay

#create configuraiton profiles to use for provisioning
/interface wifiwave2 configuration
add country=Latvia name=5ghz security=sec1 ssid=CAPsMAN_5
add name=2ghz security=sec1 ssid=CAPsMAN2
add country=Latvia name=5ghz_v security=sec1 ssid=CAPsMAN5_v

#configure provisioning rules, configure band matching as needed
/interface wifiwave2 provisioning
add action=create-dynamic-enabled master-configuration=5ghz slave-configurations=5ghz_v supported-bands=\
    5ghz-n
add action=create-enabled master-configuration=2ghz supported-bands=2ghz-n

#enable CAPsMAN service
/interface wifiwave2 capsman
set ca-certificate=auto enabled=yes
```

CAP:

```
#enable CAP service, in this case CAPsMAN is on same LAN, but you can also specify "caps-man-addresses=x.x.x.x"
here
/interface/wifiwave2/cap set enabled=yes

#set configuration.manager= on the WifiWave2 interface that should act as CAP
/interface/wifiwave2/set wifi1,wifi2 configuration.manager=capsman-or-local
```

If the CAP is hAP ax^2 or hAP ax^3, it is strongly recommended to enable RSTP in the bridge configuration, on the CAP configuration.manager should only be set on the CAP device itself, don't pass it to the CAP vai configuration profile that you provision.

The interface that should act as CAP needs additional configuration under "interface/wifiwave2/set wifiX configuration.manager="

CAPsMAN - CAP VLAN configuration example:

In this example, we will assign VLAN20 to our main SSID, and will add VLAN30 for the guest network, ether5 from CAPsMAN is connected to CAP.

CAPsMAN:

```
/interface bridge
add name=br vlan-filtering=yes
/interface vlan
add interface=br name=VLAN20 vlan-id=20
add interface=br name=VLAN30 vlan-id=30
#definfing channel is optional
/interface wifiwave2 channel
add frequency=5180,2412 name=CH
/interface wifiwave2 datapath
add bridge=br name=VLAN20 vlan-id=20
add bridge=br name=VLAN30 vlan-id=30
/interface wifiwave2 security
add authentication-types=wpa2-psk,wpa3-psk name=security
#make sure to change the country to one where you reside in
/interface wifiwave2 configuration
add channel=CH country=Latvia datapath=VLAN20 name=2Ghz_main security=security ssid=2G_MAIN
add channel=CH country=Latvia datapath=VLAN30 name=2Ghz_guest security=security ssid=2G_Guest
add channel=CH country=Latvia datapath=VLAN20 name=5Ghz_main security=security ssid=5G_MAIN
add channel=CH country=Latvia datapath=VLAN30 name=5Ghz_guest security=security ssid=5G_Guest
/ip pool
add name=dhcp_pool0 ranges=192.168.1.2-192.168.1.254
add name=dhcp_pool1 ranges=192.168.20.2-192.168.20.254
add name=dhcp_pool2 ranges=192.168.30.2-192.168.30.254
/ip dhcp-server
add address-pool=dhcp_pool0 interface=br name=dhcp1
add address-pool=dhcp_pool1 interface=VLAN20 name=dhcp2
add address-pool=dhcp_pool2 interface=VLAN30 name=dhcp3
/interface bridge port
add bridge=br interface=ether5
/interface bridge vlan
add bridge=br tagged=ether5,br vlan-ids=20
add bridge=br tagged=ether5,br vlan-ids=30
/interface wifiwave2 capsman
set enabled=yes interfaces=br
/interface wifiwave2 provisioning
add action=create-dynamic-enabled master-configuration=2Ghz_main name-format=2G-%I slave-
configurations=2Ghz_guest supported-bands=2ghz-ax
add action=create-dynamic-enabled master-configuration=5Ghz_main name-format=5G-%I slave-
configurations=5Ghz_guest supported-bands=5ghz-ax
/ip address
add address=192.168.1.1/24 interface=br network=192.168.1.0
add address=192.168.20.1/24 interface=VLAN20 network=192.168.20.0
add address=192.168.30.1/24 interface=VLAN30 network=192.168.30.0
/ip dhcp-client
add interface=ether1 disabled=no
/ip dhcp-server network
add address=192.168.1.0/24 gateway=192.168.1.1
add address=192.168.20.0/24 gateway=192.168.20.1
add address=192.168.30.0/24 gateway=192.168.30.1
```

CAP:

```
/interface bridge
add name=bridgeLocal
/interface wifiwave2 datapath
add bridge=bridgeLocal comment=defconf disabled=no name=capdp
/interface wifiwave2
set [ find default-name=wifi1 ] configuration.manager=capsman datapath=capdp disabled=no
set [ find default-name=wifi2 ] configuration.manager=capsman datapath=capdp disabled=no
/interface bridge port
add bridge=bridgeLocal comment=defconf interface=ether1
add bridge=bridgeLocal comment=defconf interface=ether2
add bridge=bridgeLocal comment=defconf interface=ether3
add bridge=bridgeLocal comment=defconf interface=ether4
add bridge=bridgeLocal comment=defconf interface=ether5
/interface wifiwave2 cap
set discovery-interfaces=bridgeLocal enabled=yes slaves-datapath=capdp
/ip dhcp-client
add interface=bridgeLocal disabled=no
```

Advanced examples

Enterprise wireless security with User Manager v5

Assigning VLAN tags to wireless traffic can be achieved by following the generic VLAN configuration example here.

Replacing stock wireless

The wifiwave2 package can be installed on some products, which ship with the bundled 'wireless' package, replacing it.

> Installing the wifiwave2 package disables other means of configuring wireless interfaces. Before installation, make sure to back up any wireless and regular CAPsMAN configuration you may want to retain.

Compatibility

Due to storage, RAM, and architecture requirements, only the following products can replace their bundled wireless software package with wifiwave2:

- hAP ac³ (non-LTE)
- Audience and Audience LTE6 kit
- RB4011iGS+5HacQ2HnD*

> * The 2.4GHz wireless interface on the RB4011iGS+5HacQ2HnD is not compatible with the wifiwave2 package. It will not be usable with the package installed.

It is also possible to install the WifiWave2 package on other devices to use WifiWave2 CAPsMAN: builds for x86, ppc, mmips and tile architectures contain the configuration utilities needed to centrally manage interfaces (as a CAPsMAN controller). Builds for arm and arm64 also contain interface drivers and firmware.

Benefits

- WPA3 authentication and OWE (opportunistic wireless encryption)
- 802.11w standard management frame protection
- 802.11r/k/v
- MU-MIMO and beamforming
- 400Mb/s maximum data rate in the 2.4GHz band for IPQ4019 interfaces
- OFDMA

Lost features

The following notable features of the bundled wireless package do not have equivalents in the wifiwave2 package

- Nstreme and Nv2 wireless protocols

Property Reference

AAA properties

Properties in this category configure an access point's interaction with AAA (RADIUS) servers.

Certain parameters in the table below take *format-string* as their value. In a *format-string*, certain characters are interpreted in the following way:

Character	Interpretation
a	Hexadecimal character making up the MAC address of the client device in lower case
A	Hexadecimal character making up the MAC address of the client device in upper case
i	Hexadecimal character making up the MAC address of the AP's interface in lower case
I (capital 'i')	Hexadecimal character making up the MAC address of the AP's interface in upper case
N	The entire name of the AP's interface (e.g. 'wifi1')
S	The entire SSID

All other characters are used without interpreting them in any way. For examples, see default values.

Property	Description
called-format (*format-string*)	Format for the value of the Called-Station-Id RADIUS attribute, in AP's messages to RADIUS servers. Default: II-II-II-II-II-II:S
calling-format (*format-string*)	Format for the value of the Calling-Station-Id RADIUS attribute, in AP's messages to RADIUS servers. Default: AA-AA-AA-AA-AA-AA
interim-update (*time interval)*	Interval at which to send interim updates about traffic accounting to the RADIUS server. Default: 5m
mac-caching (*time interval* | *'disabled'*)	Length of time to cache RADIUS server replies, when MAC address authentication is enabled. This resolves issues with client device authentication timing out due to (comparatively high latency of RADIUS server replies. Default value: disabled.
name (*string*)	A unique name for the AAA profile. No default value.
nas-identifier (*string*)	Value of the NAS-Identifier attribute, in AP's messages to RADIUS servers. Defaults to the host name of the device (/system/identity).
password-format (*format-string*)	Format for value to use in calculating the value of the User-Password attribute in AP's messages to RADIUS servers when performing MAC address authentication. Default value: "" (an empty string).
username-format (*format-string*)	Format for the value of the User-Name attribute in APs messages to RADIUS servers when performing MAC address authentication. Default value : AA:AA:AA:AA:AA:AA

Channel properties

Properties in this category specify the desired radio channel.

Property	Description
band (*2ghz-g* | *2ghz-n* | *2ghz-ax* | *5ghz-a* | *5ghz-ac* | *5ghz-an* | *5ghz-ax*)	Supported frequency band and wireless standard. Defaults to newest supported standard. **Note that band support is limited by radio capabilities.**
frequency (*list of integers or integer ranges*)	For an interface in AP mode, specifies frequencies (in MHz) to consider when picking control channel center frequency. For an interface in station mode, specifies frequencies on which to scan for APs. Leave unset (default) to consider all frequencies supported by the radio and permitted by the applicable regulatory profille. The parameter can contain 1 or more comma-separated values of integers or, optionally, ranges of integers denoted using the syntax RangeBeginning-RangeEnd:RangeStep Examples of valid channel.frequency values: • 2412 • 2412,2432,2472 • 5180-5240:20,5500-5580:20
secondary-frequency (*list of integers* | 'disabled')	Frequency (in MHz) to use for the center of the secondary part of a split 80+80MHz channel. Only official 80MHz channels (5210, 5290, 5530, 5610, 5690, 5775) are supported. Leave unset (default) for automatic selection of secondary channel frequency.
skip-dfs-channels (*10min-cac* | *all* | *disabled*)	Whether to avoid using channels, on which channel availability check (listening for presence of radar signals) is required. • *10min-cac* - interface will avoid using channels, on which 10 minute long CAC is required • *all* - interface will avoid using all channels, on which CAC is required • *disabled* (default) - interface may select any supported channel, regardless of CAC requirements
width (*20mhz* | *20/40mhz* | *20/40mhz-Ce* | *20/40mhz-eC* | *20/40/80mhz* | *20/40/80+80mhz* | *20/40/80/160mhz*)	Width of radio channel. Defaults to widest channel supported by the radio hardware.

Configuration properties

This section includes properties relating to the operation of the interface and the associated radio.

Property	Description
antenna-gain (*integer 0..30*)	Overrides the default antenna gain. The *master* interface of each radio sets the antenna gain for every interface which uses the same radio. This setting cannot override the antenna gain to be lower than the minimum antenna gain of a radio. No default value.
beacon-interval (*time interval 100ms..1s*)	Interval between beacon frames of an AP. Default: 100ms. ⓘ The 802.11 standard defines beacon interval in terms of *time units* (1 TU = 1.024 ms). The actual interval between beacons will be 1 TU for every 1 ms configured. ⚠ Every AP running on the same radio (i.e. a master AP and all its 'virtual'/'slave' APs) must use the same beacon interval.

chains (*list of integer 0..7*)	Radio chains to use for receiving signals. Defaults to all chains available to the corresponding radio hardware.
country (*name of a country*)	Determines, which regulatory domain restrictions are applied to an interface. Defaults to "United States". ⚠ It is important to set this value correctly to comply with local regulations and ensure interoperability with other devices.
dtim-period (*integer 1..255*)	Period at which to transmit multicast traffic, when there are client devices in power save mode connected to the AP. Expressed as a multiple of the beacon interval. Higher values enable client devices to save more energy, but increase network latency. Default: 1
hide-ssid (*no* | *yes*)	• *yes* - AP does not include its SSID in beacon frames, and does not reply to probe requests that have broadcast SSID. • *no* - AP includes its SSID in the beacon frames, and replies to probe requests that have broadcast SSID. Default: no
manager (*capsman* | *capsman -or-local* | *local***)**	capsman - the interface will act as CAP only, this option should **not** be passed via provisioning rules to the CAP capsman-or-local - the interface will get configuration via CAPsMAN or use its own, if /interface/wifiwave2/cap is not enabled. local - interface won't contact CAPsMAN in order to get configuration. Default: local
mode (*ap* | *station*)	Interface operation mode • *ap* (default) - interface operates as an access point • *station* - interface acts as a client device, scanning for access points advertising the configured SSID • station-bridge - interface acts as a client device and enables support for a 4-address frame format, so that the interface can be used as a bridge port ⓘ The station-bridge mode, as implemented in the wifiwave2 package, is incompatible with APs running the bundled 'wireless' package and vice versa.
multicast-enhance (*enabled* | *disabled*)	With the multicast-enhance feature enabled, an AP will convert every multicast-addressed IP or IPv6 packet into multiple unicast-addressed frames for each connected station. This may improve link throughput and reliability since, unlike multicast frames, unicasts are acknowledged by stations and transmitted using a higher data rate. Default: disabled
qos-classifier (*dscp-high-3-bits* | *priority*)	• dscp-high-3-bits - interface will transmit data packets using a WMM priority equal to the value of the 3 most significant bits of the IP DSCP field • priority - interface will transmit data packets using a WMM priority equal to that set by IP firewall or bridge filter Default: priority ⓘ 802.11ac wireless chipsets do not support the dscp-high-3-bits classifier mode.
ssid (*string*)	The name of the wireless network, aka the (E)SSID. No default value.
tx-chains (*list of integer 0..7*)	Radio chains to use for transmitting signals. Defaults to all chains available to the corresponding radio hardware.

tx-power (*integer 0..40*)	A limit on the transmit power (in dBm) of the interface. Can not be used to set power above limits imposed by the regulatory profile. Unset by default.

Datapath properties

Parameters relating to forwarding packets to and from wireless client devices.

Property	Description
bridge (*bridge interface*)	Bridge interface to add interface to, as a bridge port. No default value.
bridge-cost (*integer*)	Bridge port cost to use when adding as bridge port. Default: 10
bridge-horizon (*none* | *integer)*	Bridge horizon to use when adding as bridge port Default: none.
client-isolation (*no* | *yes*)	Determines whether client devices connecting to this interface are (by default) isolated from others or not. This policy can be overridden on a per-client basis using access list rules, so a an AP can have a mixture of isolated and non-isolated clients. Traffic from an isolated client will not be forwarded to other clients and unicast traffic from a non-isolated client will not be forwarded to an isolated one. Default: no
interface-list (*interface list*)	List to which add the interface as a member. No default value.
vlan-id (*none* | *integer* 1..4095)	Default VLAN ID to assign to client devices connecting to this interface (only relevant to interfaces in AP mode). When a client is assigned a VLAN ID, traffic coming from the client is automatically tagged with the ID and only packets tagged with with this ID are forwarded to the client. Default: none ⚠ 802.11n/ac interfaces do not support this type of VLAN tagging under the wifiwave2 package, but they can be configured as VLAN access ports in bridge settings.

Security Properties

Parameters relating to authentication.

Property	Description
authentication-types (*list of wpa-psk, wpa2-psk, wpa-eap, wpa2-eap, wpa3-psk, owe, wpa3-eap, wpa3-eap-192*)	Authentication types to enable on the interface. The default value is an empty list (no authenticaion, an open network). Configuring a passphrase, adds to the default list the *wpa2-psk* authentication method (if the interface is an AP) or both *wpa-psk* and *wpa2-psk* (if the interface is a station). Configuring an *eap-username* and an *eap-password* adds to the default list *wpa-eap and wpa2-eap* authentication methods.
connect-group (*string*)	APs within the same connect group do not allow more than 1 client device with the same MAC address. This is to prevent malicious authorized users from intercepting traffic intended to other users ('MacStealer' attack) or performing a denial of service attack by spoofing the MAC address of a victim. Handling of new connections with duplicate MAC addresses depends on the connect-priority of AP interfaces involved. By default, all APs are assigned the same connect-group.

connect-priority (accept-priority/hold-priority (*integers*))	Theese parameters determine, how a connection is handled if the MAC address of the client device is the same as that of another active connection to another AP.
	If (accept-priority of AP2) **<** (hold-priority of AP1), a connection to AP2 wil cause the client to be dropped from AP1.
	If (accept-priority of AP2) **=** (hold-priority of AP1), a connection to AP2 will be allowed only if the MAC address can no longer be reached via AP1.
	If (accept-priority of AP2) **>** (hold-priority of AP1), a connection to AP2 will not be accepted.
	
	If omitted, hold-priority is the same as accept-priority.
	By default, APs, which perform user authentication, have higher priority (lower integer value), than open APs.

dh-groups (*list of 19, 20, 21*)	Identifiers of elliptic curve cryptography groups to use in SAE (WPA3) authentication.
disable-pmkid (*no* | *yes*)	For interfaces in AP mode, disables inclusion of a PMKID in EAPOL frames. Disabling PMKID can cause compatibility issues with client devices which make use of it. • *yes* - Do not include PMKID in EAPOL frames. • *no* (default) - include PMKID in EAPOL frames.
eap-accounting (*no* | *yes*)	Send accounting information to RADIUS server for EAP-authenticated peers. Default: no.

Properties related to EAP, are only relevant to interfaces in station mode. APs delegate (passthrough) EAP authentication to the RADIUS server.

eap-anonymous-identity (*string*)	Optional anonymous identity for EAP outer authentication. No default value.
eap-certificate-mode (*dont-verify-certificate* | *no-certificates* | *verify-certificate* | *verify-certificate-with-crl*)	Policy for handling the TLS certificate of the RADIUS server. • verify-certificate - require server to have a valid certificate. Check that it is signed by a trusted certificate authority. • dont-verify-certificate (default) - Do not perform any checks on the certificate. • no-certificates - Attempt to establish the TLS tunnel by performing anonymous Diffie-Hellman key exchange. To be used if the RADIUS server has no certificate at all. • verify-certificate-with-crl - Same as *verify-certificate,* but also checks if the certificate is valid by checking the Certificate Revocation List.
eap-methods (*list of peap, tls, ttls*)	EAP methods to consider for authentication. Defaults to all supported methods.
eap-password (*string*)	Password to use, when the chosen EAP method requires one. No default value.
eap-tls-certificate (*certificate*)	Name or id of a certificate in the device's certificate store to use, when the chosen EAP authentication method requires one. No default value.
eap-username (*string*)	Username to use when the chosen EAP method requires one. No default value.

Take care when configuring encryption ciphers.

All client devices MUST support the group encryption cipher used by the AP to connect, and some client devices (notably, Intel® 8260) will also fail to connect if the list of unicast ciphers includes any they don't support.

encryption (*list of ccmp, ccmp-256, gcmp, gcmp-256, tkip*)	A list of ciphers to support for encrypting unicast traffic. Defaults to *ccmp*.

Properties related to 802.11r fast BSS transition only apply to interfaces in AP mode. Wifiwave2 interfaces in station mode do not support 802.11r.

For a client device to successfully roam between 2 APs, the APs need to be managed by the same instance of RouterOS. For information on how to centrally manage multiple APs, see CAPsMAN

ft (*no* | *yes*)	Whether to enable 802.11r fast BSS transitions (roaming). Default: no.
ft-mobility-domain (*integer 0..65535*)	The fast BSS transition mobility domain ID. Default: 44484 (0xADC4).
ft-nas-identifier (string of *2..96 hex characters*)	Fast BSS transition PMK-R0 key holder identifier. Default: MAC address of the interface.
ft-over-ds (*no* | *yes*)	Whether to enable fast BSS transitions over DS (distributed system). Default: no.
ft-preserve-vlanid (*no* | *yes*)	• no - when a client connects to this AP via 802.11r fast BSS transition, it is assigned a VLAN ID according to the access and/or interface settings • yes (default) - when a client connects to this AP via 802.11r fast BSS transition, it retains the VLAN ID, which it was assigned during initial authentication The default behavior is essential when relying on a RADIUS server to assign VLAN IDs to users, since a RADIUS server is only used for initial authentication.
ft-r0-key-lifetime (*time interval 1s.. 6w3d12h15m*)	Lifetime of the fast BSS transition PMK-R0 encryption key. Default: 600000s (~7 days)
ft-reassociation-deadline (*time interval 0..70s*)	Fast BSS transition reassociation deadline. Default: 20s.
group-encryption (*ccmp* | *ccmp-256* | *gcmp* | *gcmp-256* | *tkip*)	Cipher to use for encrypting multicast traffic. Defaults to *ccmp*.
group-key-update (*time interval*)	Interval at which the group temporal key (key for encrypting broadcast traffic) is renewed. Defaults to 24 hours.
management-encryption (*cmac* | *cmac-256* | *gmac* | *gmac-256*)	Cipher to use for encrypting protected management frames. Defaults to *cmac*.
management-protection (*allowed* | *disabled* | *required*)	Whether to use 802.11w management frame protection. **Incompatible with management frame protection in standard wireless package**. Default value depends on value of selected authentication type. WPA2 allows use of management protection, WPA3 requires it.
owe-transition-interface (*interface*)	Name or internal id of an interface whose MAC address and SSID to advertise as the matching AP when running in OWE transition mode. Required for setting up open APs that offer OWE, but also work with older devices that don't support the standard. See configuration example below.
passphrase (*string of up to 63 characters*)	Passphrase to use for PSK authentication types. Defaults to an empty string - "". WPA-PSK and WPA2-PSK authentication requires a minimum of 8 chars, while WPA3-PSK does not have minimum passphrase length.
sae-anti-clogging-threshold (*'disabled'* | *integer*)	Due to SAE (WPA3) associations being CPU resource intensive, overwhelming an AP with bogus authentication requests makes for a feasible denial-of-service attack. This parameter provides a way to mitigate such attacks by specifying a threshold of in-progress SAE authentications, at which the AP will start requesting that client devices include a cookie bound to their MAC address in their authentication requests. It will then only process authentication requests which contain valid cookies. Default: 5.
sae-max-failure-rate (*'disabled'* | *integer*)	Rate of failed SAE (WPA3) associations per minute, at which the AP will stop processing new association requests. Default: 40.
sae-pwe (*both* | *hash-to-element* | *hunting-and-pecking*)	Methods to support for deriving SAE password element. Default: both.

wps (*disabled* | *push-button*)	• *push-button* (default) - AP will accept WPS authentication for 2 minutes after 'wps-push-button' command is called. Physical WPS button functionality not yet implemented. • *disabled* - AP will not accept WPS authentication

Steering properties

Properties in this category govern mechanisms for advertising potential roaming candidates to client devices.

Property	Description
neighbor-group (*string*)	When sending neighbor reports and BSS transition management requests, an AP will list all other APs within its neighbor group as potential roaming candidates. By default, a dynamic neighbor group is created for each set of APs with the same SSID and authentication settings. APs operating in the 5GHz band are indicated to be preferable to ones operating in the 2.4GHz band.
rrm (*no* | *yes*)	Enables sending of 802.11k neighbor reports. Default: yes
wnm (*no* | *yes*)	Enables sending of solicited 802.11v BSS transition management requests. Default: yes

Miscellaneous properties

Property	Description
arp (*disabled* | *enabled* | *local-proxy-arp* | *proxy-arp* | *reply-only*)	Address Resolution Protocol mode: • *disabled* - the interface will not use ARP • *enabled* - the interface will use ARP (default) • *local-proxy-arp* - the router performs proxy ARP on the interface and sends replies to the same interface • *proxy-arp* - the router performs proxy ARP on the interface and sends replies to other interfaces • *reply-only* - the interface will only reply to requests originated from matching IP address/MAC address combinations which are entered as static entries in the ARP table. No dynamic entries will be automatically stored in the ARP table. Therefore for communications to be successful, a valid static entry must already exist.
arp-timeout (*time interval* | *'auto'*)	Determines how long a dynamically added ARP table entry is considered valid since the last packet was received from the respective IP address. Value *auto* equals to the value of *arp-timeout* in */ip settings*, which defaults to 30s.
disable-running-check (*no* | *yes*)	• *yes* - interface's *running* property will be true whenever the interface is not disabled • *no* (default) - interface's *running* property will only be true when it has established a link to another device
disabled (*no* | *yes*) (X)	Hardware interfaces are disabled by default. Virtual interfaces are not.
mac-address (*MAC*)	MAC address (BSSID) to use for an interface. Hardware interfaces default to the MAC address of the associated radio interface. Default MAC addresses for virtual interfaces are generated by 1. Taking the MAC address of the associated master interface 2. Setting the second-least-significant bit of the first octet to 1, resulting in a locally administered MAC address 3. If needed, incrementing the last octet of the address to ensure it doesn't overlap with the address of another interface on the device
mtu (*integer [32..2290]*; Default: **1500**)	Layer3 Maximum transmission unit.
l2mtu (*integer [32..2290]*; Default: **2290**)	Layer2 Maximum transmission unit.

master-interface (*interface*)	Multiple interface configurations can be run simultaneously on every wireless radio. Only one of them determines the radio's state (whether it is enabled, what frequency it's using, etc). This 'master' interface, is *bound* to a radio with the corresponding *radio-mac.* To create additional ('virtual') interface configurations on a radio, they need to be *bound* to the corresponding master interface. No default value.
name (*string*)	A name for the interface. Defaults to *wifiN*, where *N* is the lowest integer that has not yet been used for naming an interface.

Read-only properties

Property	Description
bound (*boolean*) (B)	True for *master* interfaces that are currently available for WiFi manager. True for a virtual interface (configurations linked to a master interface) when both the interface itself and its master interface are not disabled and the *master* interface has a bound flag.
default-name (*string*)	The default name for an interface.
inactive (*boolean*) (I)	False for interfaces in AP mode when they've selected a channel for operation (i.e. configuration has been successfully applied). False for interfaces in station mode when they've connected to an AP (i.e. configuration has been successfully applied, an with AP with matching settings has been found). True otherwise.
master (*boolean*) (M)	True for physical interfaces on router itself or detected CAP if running as CAPsMAN. False for virtual interfaces.
radio-mac (*MAC*)	The MAC address of the associated radio.
running (*boolean*) (R)	True, when an interface has established a link to another device. If *disable-running-check* is set to 'yes', true whenever the interface is not disabled.

Access List

Filtering parameters	
Parameter	**Description**
interface (*interface* | *interface-list* | *'any'*)	Match if connection takes place on the specified interface or interface belonging to specified list. Default: any.
mac-address (*MAC address*)	Match if the client device has the specified MAC address. No default value.
mac-address-mask (*MAC address*)	Modifies the **mac-address** parameter to match if it is equal to the result of performing bit-wise AND operation on the client MAC address and the given address mask. Default: FF:FF:FF:FF:FF:FF (i.e. client's MAC address must match value of **mac-address** exactly)
signal-range (*min..max*)	Match if the strength of received signal from the client device is within the given range. Default: '-120..120'
ssid-regexp (*regex*)	Match if the given regular expression matches the SSID.
time (*start-end,days*)	Match during the specified time of day and (optionally) days of week. Default: 0s-1d

Action parameters	
Parameter	**Description**
allow-signal-out-of-range (*time period* | 'always')	The length of time which a connected peer's signal strength is allowed to be outside the range required by the **signal -range** parameter, before it is disconnected. If the value is set to 'always', peer signal strength is only checked during association. Default: 0s.
action (*accept* | *reject* | *query-radius*)	Whether to authorize a connection • *accept* - connection is allowed • *reject* - connection is not allowed • *query-radius* - connection is allowed if MAC address authentication of the client's MAC address succeeds Default: *accept*
client-isolation (*no* | *yes*)	Whether to isolate the client from others connected to the same AP. No default value.
passphrase (*string*)	Override the default passphrase with given value. No default value.
radius-accounting (*no* | *yes*)	Override the default RADIUS accounting policy with given value. No default value.
vlan-id (*none* | *integer 1..4095*)	Assign the given VLAN ID to matched clients. No default value.

Frequency scan

Information about RF conditions on available channels can be obtained by running the frequency-scan command.

Command parameters	
Parameter	**Description**
duration (*time interval)*	Length of time to perform the scan for before exiting. Useful for non-interactive use. Not set by default.
freeze-frame-interval (*time interval)*	Time interval at which to update command output. Default: 1s.
frequency (*list of frequencies /ranges)*	Frequencies to perform the scan on. See channel.frequency parameter syntax above for more detail. Defaults to all supported frequencies.
numbers (*string)*	Either the name or internal id of the interface to perform the scan with. Required. Not set by default.
rounds (*integer)*	Number of times to go through list of scannable frequencies before exiting. Useful for non-interactive use. Not set by default.
save-file (string)	Name of file to save output to. Not set by default.

Output parameters	
Parameter	**Description**
channel (*integer)*	Frequency (in MHz) of the channel scanned.
networks (*integer)*	Number of access points detected on the channel.
load (integer*)*	Percentage of time the channel was busy during the scan.
nf (integer)	Noise floor (in dBm) of the channel.
max-signal (*integer*)	Maximum signal strength (in dBm) of APs detected in the channel.
min-signal (*integer*)	Minimum signal strength (in dBm) of APs detected in the channel.

primary (*boolean*) (P)	Channel is in use as the primary (control) channel by an AP.
secondary (boolean) (S)	Channel is in use as a secondary (extension) channel by an AP.

Scan command

The '/interface wifiwave2 scan' command will scan for access points and print out information about any APs it detects.

The scan command takes all the same parameters as the frequency-scan command.

Output parameters	
Parameter	**Description**
active (*boolean*) (A)	Signifies that beacons from the AP have been received in the last 30 seconds.
address (*MAC*)	The MAC address (BSSID) of the AP.
channel (*string*)	The control channel frequency used by the AP, its supported wireless standards and control/extension channel layout.
security (*string*)	Authentication methods supported by the AP.
signal (*integer*)	Signal strength of the AP's beacons (in dBm).
ssid (*string*)	The extended service set identifier of the AP.
sta-count (*integer*)	The number of client devices associated with the AP. Only available if the AP includes this information in its beacons.

Sniffer

Command parameters	
Parameter	**Description**
duration (*time interval*)	Automatically interrupt the sniffer after the specified time has passed. No default value.
filter (*string*)	A string that specifies a filter to apply to captured frames. Only frames matched by the filter expression will be displayed, saved or streamed. This works similarly to filter strings in libpcap, for example. The filter can match • Address fields (addr1, addr2, addr3) • Wireless frame type and subtype, including shortcuts such as 'beacon' (type == 0 && subtype == 8) • Flags (to-ds, from-ds, retry, power, protected) A string can include the following operators: • == (exact match) • != (does not equal) • && (logical AND) • || (logical OR) • () (for grouping filter expressions)
number (*interface*)	Interface to use for sniffing.
pcap-file (*string)*	Save captured frames to a file with the given name. No default value (captured frames are not saved to a file by default).
pcap-size-limit (*integer*)	File size limit (in bytes) when storing captured frames locally. When this limit has been reached, no new frames are added to the capture file. No default value.
stream-address (IP address)	Stream captured packets via the TZSP protocol to the given address. No default value (captured packets are not streamed anywhere by default).

stream-rate (*integer*)	Limit on the rate (in packets per second) at which captured frames are streamed via TZSP.

WPS

interface/wifiwave2/wps-client wifi

Command parameters	
Parameter	**Description**
duration (*time interval*)	Length of time after which the command will time out if no AP is found. Unlimited by default.
interval (*time interval*)	Time interval at which to update command output. Default: 1s.
mac-address (*MAC*)	Only attempt connecting to AP with the specified MAC (BSSID). Not set by default.
numbers (*string*)	Name or internal id of the interface with which to attempt connection. Not set by default.
ssid (*string*)	Only attempt to connect to APs with the specified SSID. Not set by default.

Radios

Information about the capabilities of each radio can be gained by running the `/interface/wifiwave2/radio print detail` command.

Property	**Description**
2g-channels (*list of integers*)	Frequencies supported in the 2.4GHz band.
5g-channels (*list of integers*)	Frequencies supported in the 5GHz band.
bands (*list of strings*)	Supported frequency bands, wireless standards and channel widths.
ciphers (*list of strings*)	Supported encryption ciphers.
countries (*list of strings*)	Regulatory domains supported by the interface.
min-antenna-gain (*integer*)	Minimum antenna gain permitted for the interface.
phy-id (*string*)	A unique identifier.
radio-mac (*MAC*)	MAC address of the radio interface. Can be used to match radios to interface configurations.
rx-chains (*list of integers*)	IDs for radio chains available for receiving radio signals.
tx-chains (*list of integers*)	IDs for radio chains available for transmitting radio signals.

Registration table

The registration table contains read-only information about associated wireless devices.

Parameter	**Description**
authorized (*boolean*) (A)	True when the peer has successfully authenticated.
bytes (*list of integers*)	Number of bytes in packets transmitted to a peer and received from it.
interface (*string*)	Name of the interface, which was used to associate with the peer.
mac-address (*MAC*)	The MAC address of the peer.
packets (*list of integers*)	Number of packets transmitted to a peer and received from it.
rx-rate *(string)*	Bitrate of received transmissions from peer.
signal (*integer)*	Strength of signal received from the peer (in dBm).

tx-rate (*string)*	Bitrate used for transmitting to the peer.
uptime (*time interval)*	Time since association.

CAPsMAN Global Configuration

Menu: /interface/wifiwave2/capsman

Property	Description
ca-certificate *(auto | certificate name)*	Device CA certificate, CAPsMAN server requires a certificate, certificate on CAP is optional.
certificate (*auto | certificate name | none*; Default: **none**)	Device certificate
enabled *(no | yes*)	Disable or enable CAPsMAN functionality
package-path (*string |*; Default:)	Folder location for the RouterOS packages. For example, use "/upgrade" to specify the upgrade folder from the files section. If an empty string is set, CAPsMAN can use built-in RouterOS packages, note that in this case only CAPs with the same architecture as CAPsMAN will be upgraded.
require-peer-certificate (*yes | no*; Default: **no**)	Require all connecting CAPs to have a valid certificate
upgrade-policy (*none | require-same-version | suggest-same-upgrade*; Default: **none**)	Upgrade policy options • none - do not perform upgrade • require-same-version - CAPsMAN suggest to upgrade the CAP RouterOS version and, if it fails it will not provision the CAP. (Manual provision is still possible) • suggest-same-version - CAPsMAN suggests to upgrade the CAP RouterOS version and if it fails it will still be provisioned
interfaces *(all | interface name | none; Default:* ***all****)*	Interfaces on which CAPsMAN will listen for CAP connections

CAPsMAN Provisioning

Provisioning rules for matching radios are configured in **/interface/wifiwave2/provisioning/** menu:

Property	Description
action (*create-disabled | create-enabled | create-dynamic-enabled | none*; Default: **none**)	Action to take if rule matches are specified by the following settings: • **create-disabled** - create disabled static interfaces for radio. I.e., the interfaces will be bound to the radio, but the radio will not be operational until the interface is manually enabled; • **create-enabled** - create enabled static interfaces. I.e., the interfaces will be bound to the radio and the radio will be operational; • **create-dynamic-enabled** - create enabled dynamic interfaces. I.e., the interfaces will be bound to the radio, and the radio will be operational; • **none** - do nothing, leaves radio in the non-provisioned state;
comment (*string*; Default:)	Short description of the Provisioning rule
common-name-regexp (*string*; Default:)	Regular expression to match radios by common name. Each CAP's common name identifier can be found under "/interface/wifiwave2/radio" as value "REMOTE-CAP-NAME"
supported-bands (*2ghz-ax | 2ghz-g | 2ghz-n | 5ghz-a | 5ghz-ac | 5ghz-ax | 5ghz-n*; Default:)	Match radios by supported wireless modes.
identity-regexp (*string*; Default:)	Regular expression to match radios by router identity

address-ranges (*IpAddressRange[, IpAddressRanges] max 100x*; Default: **""**)	Match CAPs with IPs within configured address range. Will only work for CAPs that joined CAPsMAN using IP, not MAC address.
master-configuration (*string*; Default:)	If **action** specifies to create interfaces, then a new master interface with its configuration set to this configuration profile will be created
name-format (*string*)	Base string to use when constructing names of provisioned interfaces. Each new interface will be created by taking the base string and appending a number to the end of it. If included in the string, character sequence **%I** will be replaced by the system identity of the cAP. **%C** will be replaced with the cAP's TLS certificate's Common Name. Default: "cap-wifi"
radio-mac (*MAC address*)	MAC address of radio to be matched. No default value.
slave-configurations (*string*; Default:)	If **action** specifies to create interfaces, then a new slave interface for each configuration profile in this list is created.
disabled (*yes* | *no*; Default: **no**)	Specifies if the provision rule is disabled.

CAP configuration

Menu: /interface/wifiwave2/cap

Property	Description
caps-man-addresses *(list of IP addresses; Default:* ***empty***)	List of Manager IP addresses that CAP will attempt to contact during discovery
caps-man-names ()	An ordered list of CAPs Manager names that the CAP will connect to, if empty - CAP does not check Manager name
discovery-interfaces (*list of interfaces*;)	List of interfaces over which CAP should attempt to discover Manager
lock-to-caps-man (*no* | *yes;* Default: **no**)	Sets, if CAP should lock to the first CAPsMAN it connects to
slaves-static ()	Creates Static Virtual Interfaces, allows the possibility to assign IP configuration to those interfaces. MAC address is used to remember each static-interface when applying the configuration from the CAPsMAN.
caps-man-certificate-common-names ()	List of Manager certificate CommonNames that CAP will connect to, if empty - CAP does not check Manager certificate CommonName
certificate ()	Certificate to use for authenticating
enabled (*yes* | *no*; Default: **no**)	Disable or enable the CAP feature
slaves-datapath ()	

W60G

Summary

Packages: `system,wireless`

802.11ad implementation capable of providing Gigabit Ethernet speeds over wireless network.

Extend your Gigabit network over a transparent AES encrypted wireless 60GHz link without usual wired or wireless network problems.

General interface properties

Sub-menu: `/interface w60g`

> **Warning:** Wireless Wire kit devices comes in pre-configured, connected pairs. Manual configuration is optional

Property	Description
arp (*disabled* | *enabled* | *proxy-arp* | *reply-only*; Default: **enabled**)	`Read more >>`
arp-timeout (*auto* | *integer*; Default: **auto**)	ARP timeout is time how long ARP record is kept in ARP table after no packets are received from IP. Value **auto** equals to the value of **arp-timeout** in **/ip settings**, default is 30s
comment (*string*; Default:)	Short description of the interface
disabled (*yes* | *no*; Default: **yes**)	Whether interface is disabled
frequency (*58320* | *60480* | *62640* | *64800* | *66000* | *auto*; Default: **auto**)	Frequency used in communication (Only active on bridge device)
isolate-stations (*yes* | *no*; Default: **yes**)	Don't allow communication between connected clients (from RouterOS 6.41)
l2mtu (*integer [0..7882]*; Default: **1600**)	Layer2 Maximum transmission unit
mac-address (*MAC*; Default:)	MAC address of the radio interface
mdmg-fix (*yes* | *no*; Default: **no**)	Experimental feature working only on wAP60Gx3 devices, providing better point to multi point stability in some cases
mode (*ap-bridge* | *bridge* | *sniff* | *station-bridge*; Default: **bridge**)	Operation mode
mtu (*integer [32..8192]*; Default: **1500**)	Layer3 Maximum transmission unit
name (*string*; Default: **wlan60-1**)	Name of the interface

password (*string*; Default: **randomly generated**)	Password used for AES encryption
put-stations-in-bridge (; Default:)	Put newly created station device interfaces in this bridge
region (*asia* | *australia* | *canada* | *china* | *eu* | *japan* | *no-region-set* | *usa*; Default: **no-region-set**)	Parameter to limit frequency use
scan-list (*58320,60480,62640,64800,66000*; Default: **58320,60480,62640,64800**)	Scan list to limit connectivity over frequencies in station mode
ssid (*string (0..32 chars)*; Default: **value of System Identity**)	SSID (service set identifier) is a name that identifies wireless network
tx-sector (*integer [0..63]* | *auto*; Default: **auto**)	Disables beamforming and locks to selected radiation pattern

Sub-menu: `/interface w60g print stats`

Provides more detailed information about Beaforming occurrences and some debug information:

```
/interface w60g print stats name: wlan60-1
beamforming-event: 310
tx-io-msdu: 0
tx-sw-msdu: 154 663
tx-fw-msdu: 102
tx-ppdu: 220 147
tx-ppdu-from-q: 40 327
tx-mpdu-new: 154 663
tx-mpdu-total: 184 759
tx-mpdu-retry: 30 096
rx-ppdu: 166 636
rx-mpdu-crc-err: 4 817
rx-mpdu-crc-ok: 285 649
```

Station interface properties

> **Warning:** ap-bridge device requires License level 4 (click for more information) to support more than one connected client

From RouterOS 6.41 - Point To Multi Point support is added.

There are several important changes and improvements in later versions. Please always upgrade to latest versions!

Connected clients are treated as individual interfaces, after successful connection new station interface is created.

After update default configuration still works - newly created station interface is moved to default bridge.

Sub-menu: `/interface w60g station`

Property	Description
parent (*string*; Default: **wlan60-***)	Parent interface name
put-in-bridge (*none* | *parent* | *bridge-name*; Default: **parent**)	Add station device interface to specific bridge
remote-address (*MAC*; Default: **matches bridge interface MAC**)	MAC address of bridge interface, station is connecting to

Scan

```
/interface w60g scan wlan60-1
```

Scan command searches for and displays available AP(s) in the frequency range supported by the W60G interface.

Using scan command the interface operation is disabled (wireless link is disconnected during the scan operation)

Currently it is impossible to do background scans.

Monitor

```
/interface w60g monitor wlan60-1
connected: yes frequency: 58320
remote-address: 04:D6:AA:AA:AA:AA
mcs: 8
phy-rate: 2.3Gbps
signal: 80 rssi: -68
tx-sector: 28
tx-sector-info: center
distance: 160.9m
```

Monitor shows current state of active connection. Distance measurement tool provides very precise distance measurements. "tx-sector-info" (feature in testing stage) provides information from currently used beamforming pattern and shows direction to center - theoretical highest power output point.

Align

```
/interface w60g align wlan60-1
connected: yes
frequency: 58320
remote-address: 04:D6:AA:AA:AA:AB
tx-mcs: 6
tx-phy-rate: 1540.0Mbps
signal: 70
rssi: -62
10s-average-rssi: -63.1
tx-sector: 62
tx-sector-info: left 19 degrees, up 26.6 degrees
rx-sector: 96
distance: 220.88m
tx-packet-error-rate: 5%
```

In align mode frames between two devices are exchanged more rapidly and information about signal quality is displayed more often. Use "rssi", "10s-average-rssi" and "tx-sector-info" (available from 6.44beta39) values for more precise link alignment. When devices enter align mode - link is lost for a few seconds.

Sniff

Sniff mode allows to capture nearby 802.11ad frames. To use sniff mode same frequency needs to be used and interface operational mode needs to be set to sniff:

```
/interface w60g set wlan60-1 mode=sniff
```

Now this interface can be used in Tools/Packet Sniffer for packet capture. Sniffer mode can't be used together with regular interface working modes.

Point to Multi Point setup example

All MikroTik devices can be interconnected. There are three different versions of wAP60G devices currently available:

- Wireless Wire kit
- wAP 60G
- SXTsq60 Lite60
- wAP 60G AP

And

- Wireless Wire Dish

Hardware wise wAP devices are identical, but there are some software limitations -

wAP 60G AP is designed for Access Point usage in PtMP (Point to Multi Point) setups, but can be also used as PtP (Point to Point) or as Station device. It's already equipped with level4 license for more than one connected client support More about Licenses

Wireless Wire kit, **Wireless Wire Dish**, **SXTsq Lite60** and **wAP60G** devices comes with level3 license. Wireless wire dish should be only used as Client device due to it's narrow radiation pattern.

License upgrade is needed to unlock more than one simultaneously connected client in Access Point mode, but devices can connect to Access Points as regular Station devices.

> **Warning:** Before configuration, make sure devices are running latest software versions: How to upgrade

Minimal configuration for transparent wireless link is matching SSID, correct mode (bridge || station-bridge) and Wireless and Ethernet interfaces put in same bridge.

In current example we will look at usage case where **wAP60G AP** is used as Access Point, **wAP60G** and **Wireless Wire kit** devices are used as Station devices, forming 4 unit network.

> **Warning:** It's recommended to change default IP addresses to avoid connection issues to the devices

wAP60G AP units come pre-configured with WISP Bridge default configuration

SSID and bridge between Wireless and Ethernet interfaces is already configured. It's recommended to set up Wireless password and change SSID. If device has been reset, you can also set correct mode and enable interface.

One liner that does all previously mentioned steps:

```
/interface w60g set wlan60-1 password="put_your_safe_password_here" ssid="put_your_new_ssid_here" disabled=no
mode=ap-bridge
```

Wireless Wire and wAP60G units come pre-configured with PTP Bridge default configuration.

Wireless Wire devices have already randomly generated matching SSID and Wireless password.

Bridge device (Bridge or Access point device with one connected client support) needs Wireless mode change to station-bridge.

One liner that can be used to set devices in client mode:

```
/interface w60g set wlan60-1 password="put_your_safe_password_here" ssid="put_your_new_ssid_here" disabled=no
mode=station-bridge
```

If configuration is done from empty configuration (reset without default configuration) -

new bridge needs to be created containing Wireless and Ethernet interfaces and IP address for easy access should be added.

```
{ /interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=wlan60-1
/ip address add address=192.168.88.1/24 interface=bridge1
}
```

For Access Point add this line to ensure all connected stations will be put in same bridge.

```
/interface w60g set wlan60-1 put-stations-in-bridge=bridge1
```

After successful connection for each Client device new entry will appear on Access Point device under:

```
/interface w60g station print
```

```
Flags: X - disabled, R - running

0 name="wlan60-station-1" parent=wlan60-1 remote-address=AA:AA:AA:AA:AA:AA mtu=1500 mac-address=AA:AA:AA:AA:AA:
AB arp=enabled arp-timeout=auto put-in-bridge=parent

0 name="wlan60-station-2" parent=wlan60-1 remote-address=AA:AA:AA:AA:AB:AA mtu=1500 mac-address=AA:AA:AA:AA:AA:
AC arp=enabled arp-timeout=auto put-in-bridge=parent

0 name="wlan60-station-3" parent=wlan60-1 remote-address=AA:AA:AA:AA:AC:AA mtu=1500 mac-address=AA:AA:AA:AA:AA:
AD arp=enabled arp-timeout=auto put-in-bridge=parent

0 name="wlan60-station-4" parent=wlan60-1 remote-address=AA:AA:AA:AA:AD:AA mtu=1500 mac-address=AA:AA:AA:AA:AA:
AE arp=enabled arp-timeout=auto put-in-bridge=parent
```

For each client separate settings can be applied (queues, VLANS, Firewall rules, etc) providing more flexibility in configuration.

To limit client-client communication in same bridge isolate-stations option can be used on Access Point device:

```
/interface w60g set wlan60-1 isolate-stations=yes
```

Point to Point GUI configuration example

Point to Point GUI configuration example

Troubleshooting and Recommendations

MikroTik 60GHz solutions functionality includes support for for ATPC (Adaptive Transmit Power Control)

Physical Properties

Atmospheric attenuation for the wireless frequencies used in 802.11ad standard is very high, this should be taken in account before deploying links.

The Wireless Wire kit have been tested in distances up to 200 meters.

Wireless Wire dish kit is tested at distances up to 2500 meters For stability and full speed availability this kit is recommended for distances up to 1500 meters.

wAP60G devices are equipped with phase array 60° beamforming antennas, that can help signal find the way around objects in short distances but it's still vital to keep the line of sight clear on higher distances.

LHG60G device single radiation pattern is less than 1 degree (both Horizontal and Vertical), All patterns combined provide close to 3 degree coverage in both Horizontal and Vertical planes, best one for each situation is calculated by using beamforming algorithm. Beam width and direction depends on used predefined calibrated sector.

Device RF characteristics

60 GHz devices

Device	Width of single antenna pattern and full span in degrees	EIRP	Tx-power	Center sectors*
wAP 60G	15-20 degrees single pattern and full span 60 degrees over horizontal and 30 degrees vertical plane	< 40 dBm		27,28,35,36
wAP 60G AP	15-20 degrees single pattern and full span 60 degrees over horizontal and 30 degrees vertical plane	< 40 dBm		27,28,35,36
Wireless Wire	15-20 degrees single pattern and full span 60 degrees over horizontal and 30 degrees vertical plane	< 40 dBm		27,28,35,36
wAP 60Gx3 AP	15-20 degrees single pattern and full span 180 degrees over horizontal and 30 degrees vertical plane	< 40 dBm		27,28,35,36
SXTsq Lite 60	15-20 degrees single pattern and full span 60 degrees over horizontal and 30 degrees vertical plane	< 40 dBm		27,28,35,36

Cube Lite 60	4-8 degrees single pattern and full span 12 degrees over horizontal and 12 degrees vertical plane	< 40 dBm	< 10 dBm	27,28,35,36
Cube 60G ac	4-8 degrees single pattern and full span 12 degrees over horizontal and 12 degrees vertical plane	< 40 dBm	< 10 dBm	27,28,35,36
Cube 60Pro ac	4-8 degrees single pattern and full span 11 degrees over horizontal and 11 degrees vertical plane	< 40 dBm	< 10 dBm	27,28,35,36
CubeSA 60Pro ac	15 degrees single pattern and full span 60 degrees over horizontal and 30 degrees vertical plane	< 40 dBm	< 10 dBm	27,28,35,36
LHG Lite 60	< 1 degree single pattern and full span 3 degrees over horizontal and 3 degrees vertical plane	< 55 dBm	< 10 dBm	27,28,35,36
LHG 60G	< 1 degree single pattern and full span 3 degrees over horizontal and 3 degrees vertical plane	< 55 dBm	< 10 dBm	27,28,35,36
Wireless Wire Dish	< 1 degree single pattern and full span 3 degrees over horizontal and 3 degrees vertical plane	< 55 dBm	< 10 dBm	27,28,35,36
Wireless Wire nRAY	< 1 degree single pattern and full span 3 degrees over horizontal and 3 degrees vertical plane	< 55 dBm or <40 dBm with EU region	< 10 dBm	31

***center sector is calibrated center of beamforming array**

Regions

MikroTik 802.11ad devices support frequency range: 57240 MHz - 67080 MHz, frequency and channel use can be limited if "region" parameter is used.

Region	lower frequency	upper frequency	usable channels
USA	57.24 GHz	70.20 GHz	1, 2, 3, 4, 5, 6
Canada	57.24 GHz	63.72 GHz	1, 2, 3
Asia	57.24 GHz	63.72 GHz	1, 2, 3
EU	57.24 GHz	65.88 GHz	1, 2, 3, 4
Japan	57.24 GHz	65.88 GHz	1, 2, 3, 4
Australia	57.24 GHz	65.88 GHz	1, 2, 3, 4
China	59.40 GHz	63.72 GHz	2, 3

Connection issues

In order to connect devices they need to be in direct visibility, "scan-list" on client device needs to include "frequency" used on AP device. LHG60 devices require very precise alignment in order to get best performance in higher distances.

SNMP OIDs for monitoring

From RouterOS>=6.42rc6 SNMP support for W60G interface monitoring is added

```
For main interfaces:
1.3.6.1.4.1.14988.1.1.1.8.1.2.1  integer  Mode
1.3.6.1.4.1.14988.1.1.1.8.1.3.1  string   SSID
1.3.6.1.4.1.14988.1.1.1.8.1.4.1  integer  Connected status
1.3.6.1.4.1.14988.1.1.1.8.1.5.1  string   Remote MAC
1.3.6.1.4.1.14988.1.1.1.8.1.6.1  integer  Frequency
1.3.6.1.4.1.14988.1.1.1.8.1.7.1  integer  MCS
1.3.6.1.4.1.14988.1.1.1.8.1.8.1  integer  Signal quality
1.3.6.1.4.1.14988.1.1.1.8.1.9.1  integer  tx-sector
1.3.6.1.4.1.14988.1.1.1.8.1.11.1 string   Sector info
1.3.6.1.4.1.14988.1.1.1.8.1.12.1 integer  RSSI
1.3.6.1.4.1.14988.1.1.1.8.1.13.1 gauge32  PHY rate
```

station interfaces will be numbered under different table:

```
1.3.6.1.4.1.14988.1.1.1.9.1.2.(interfaceID) = integer Connected status
1.3.6.1.4.1.14988.1.1.1.9.1.3.(interfaceID) = Hex-STRING mac-address
1.3.6.1.4.1.14988.1.1.1.9.1.4.(interfaceID) = INTEGER: MCS
1.3.6.1.4.1.14988.1.1.1.9.1.5.(interfaceID) = INTEGER: Signal Quality Index
1.3.6.1.4.1.14988.1.1.1.9.1.6.(interfaceID) = INTEGER: tx-sector
1.3.6.1.4.1.14988.1.1.1.9.1.8.(interfaceID) = Gauge32: data-rate [Mbps]
1.3.6.1.4.1.14988.1.1.1.9.1.9.(interfaceID) = INTEGER: RSSI
1.3.6.1.4.1.14988.1.1.1.9.1.10.(interfaceID) = INTEGER: distance [cm]
```

InterfaceID is added from 3 and increases by +1 for each connected station. More information about SNMP functionality and MIB files can be found in SNMP manual

Configuration Reset For Wireless Wire kits

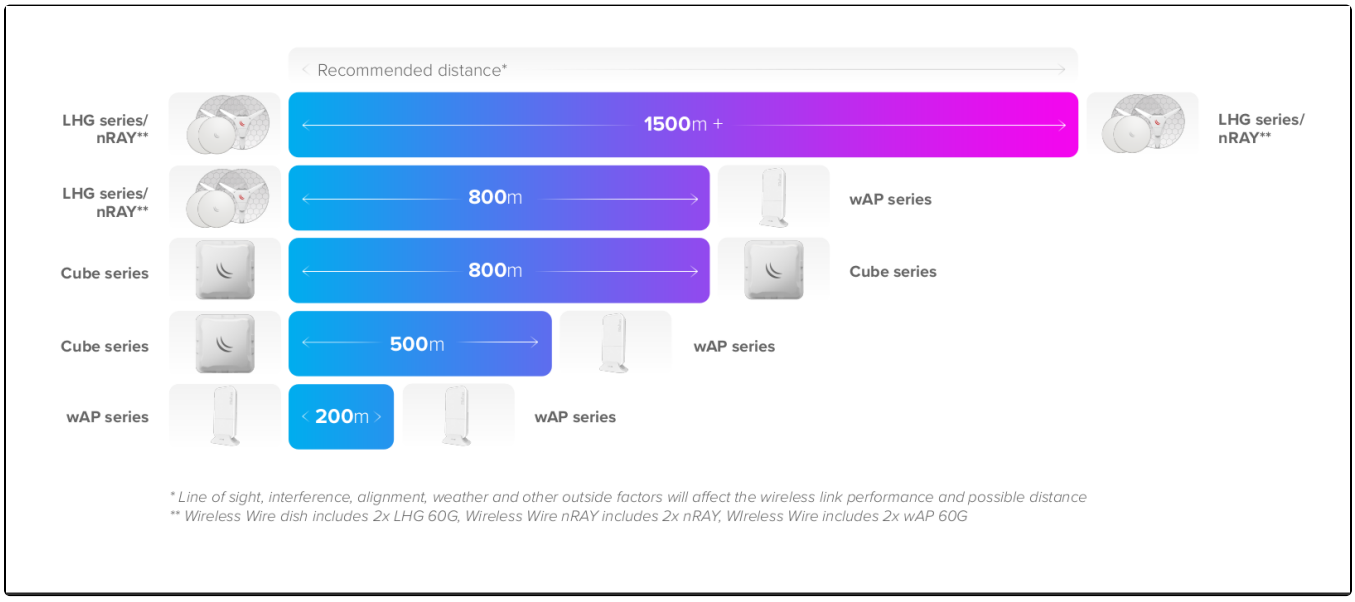
Reset button has same functionality as on other devices, explained in detail here

5 second button hold on startup (USR LED light starts flashing) - resets to password protected state.

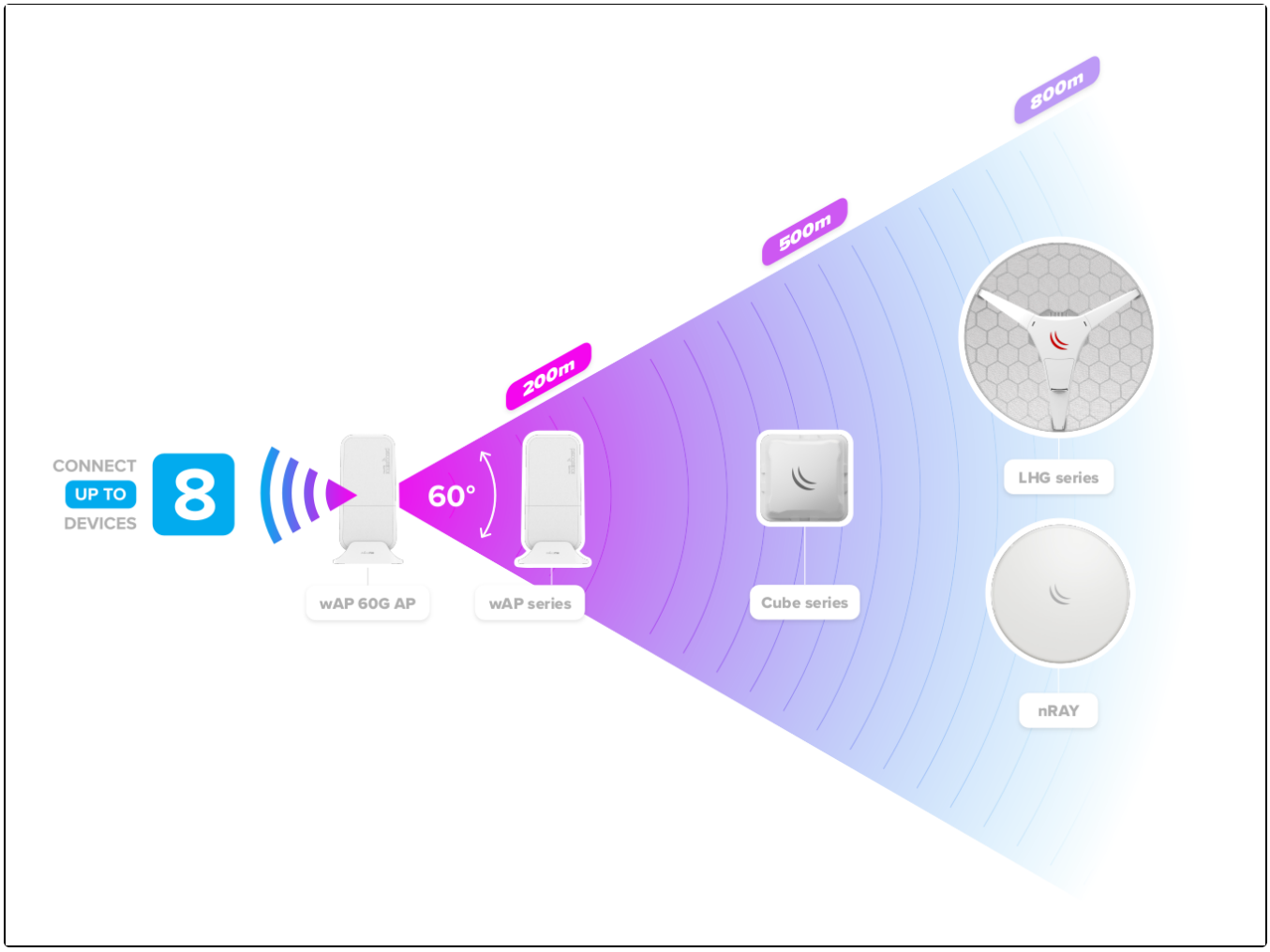
10 second button hold on startup (USR LED turns solid after flashing) - completely removes configuration.

> **Warning:** After complete removal of configuration, only mac-telnet connection may be established

Distance guide




LHG series
800m
nRAY
500m
Cube series
200m
wAP series
180°
wAP 60Gx3 AP
8
CONNECT
UP TO
DEVICES



For Point To Multi Point as **AP** device with license level 4 or higher is required.

Product code	License level	Link to product description
Cube-60ad	3	Cube Lite60
CubeG-5ac60ad	3	Cube 60G ac
CubeG-5ac60adpair	3	Wireless Wire Cube
LHG-60ad	3	LHG Lite60
LHGG-60ad	3	LHG 60G
LHGG-60adkit	3	Wireless Wire Dish
nRAYG-60adpair	3	Wireless Wire nRAY
wAPG-60ad	3	wAP 60G
wAPG-60adkit	3	Wireless Wire
wAPG-60ad-A	**4**	wAP 60G AP
wAPG-60ad-SA	**4**	wAP 60Gx3 AP
CubeG-5ac60ay	**4**	Cube 60Pro ac
CubeG-5ac60ay-SA	**4**	CubeSA 60Pro ac

Fail-over PtMP CLI example

Summary

This example shows how to configure automatic fail-over (bonding) 5Ghz link in combination with 60Ghz devices in CLI.
When a connection between 60Ghz wireless is lost, it will automatically use the bonded interface.
Example is done from empty configuration state with [WinBox] utility

Connect to the device step by step

1. After configuration reset - only mac-telnet is possible.
   Connect to device by connecting to it's MAC address or use WinBox New terminal to find device MAC address of the W60G device by issuing command:

   ```
   /ip neighbor print
   ```

2. To connect to the W60G device issue a command:

   ```
   /tool mac-telnet mac-address
   ```

3. Enter username and password. By default username is **admin** and password is either blank or printed on the device sticker.

   ```
   [admin@KD_GW] > /tool mac-telnet C4:AD:34:84:EE:5D
   Login: admin
   Password:
   Trying C4:AD:34:84:EE:5D...
   Connected to C4:AD:34:84:EE:5D
   ```

Configure bridge

1. Add new bridge and assign bridge members to it by issuing the following command:

   ```
   /interface bridge add name=bridge
   ```

   To check if the bridge has been created issue a command:

   ```
   [admin@MikroTik] > /interface bridge print
   Flags: X - disabled, R - running
   0 R name="bridge" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled arp-timeout=auto mac-address=1A:7F:
   BB:41:B0:94 protocol-mode=rstp
   fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m priority=0x8000 max-message-age=20s
   forward-delay=15s transmit-hold-count=6
   vlan-filtering=no dhcp-snooping=no
   ```

Set up 60Ghz wireless connection

All previously explained steps are identical to Bridge and Station devices. When configuring wireless interface different modes needs to be used.

For ap-bridge device -

- Choose SSID, Password, frequency and choose bridge mode option that will act as **ap-bridge** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface w60g set wlan60-1 mode=ap-bridge frequency=auto ssid=MySSID
password=choosepassword isolate-stations=no
[admin@MikroTik] > /interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=ap-bridge ssid="MySSID"
frequency=auto default-scan-list=58320,60480,62640,64800 password="choosepassword" tx-sector=auto put-
stations-in-bridge=none isolate-stations=no
[admin@MikroTik] > /interface w60g enable wlan60-1
```

For Station devices -

- Choose the same SSID, Password, frequency as the bridge device and choose station-bridge mode option that will act as a **station** for the setup, please see the example.
- It is possible to connect up to 8 station devices to a single AP to act as a fail-over.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface w60g set wlan60-1 mode=station-bridge frequency=auto ssid=MySSID
password=choosepassword isolate-stations=no
[admin@MikroTik] > /interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=station-bridge ssid="MySSID"
frequency=auto default-scan-list=58320,60480,62640,64800password="choosepassword" tx-sector=auto put-
stations-in-bridge=bridge isolate-stations=no
[admin@MikroTik] > /interface w60g enable wlan60-1
```

Set up 5Ghz wireless connection

For ap-bridge device -

- Choose SSID, Password, frequency and choose bridge mode option that will act as a **bridge** for the setup, please see the example.
- Enable 5Ghz interface after required parameters have been set.

```
[admin@MikroTik] > /interface wireless security-profiles set [ find default=yes ] supplicant-
identity=MikroTik authentication-types=wpa2-psk mode=dynamic-keys wpa2-pre-shared-key=choosepassword
[admin@MikroTik] > /interface wireless set wlan1 frequency=auto scan-list=default installation=outdoor
mode=ap-bridge ssid=MikroTik1 channel-width=20/40/80mhz-Ceee wireless-protocol=any security-
profile=default band=5ghz-a/n/ac
[admin@MikroTik] > /interface wireless enable wlan1
```

For Station devices -

- Choose the same SSID, Password, frequency as the bridge device and choose station-bridge mode option that will act as a **station** for the setup, please see the example.
- It is possible to connect up to 8 station devices to a single AP to act as a fail-over.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface wireless security-profiles set [ find default=yes ] supplicant-
identity=MikroTik authentication-types=wpa2-psk mode=dynamic-keys wpa2-pre-shared-key=choosepassword
[admin@MikroTik] > /interface wireless set wlan1 frequency=auto scan-list=default installation=outdoor
mode=station-bridge ssid=MikroTik1 channel-width=20/40/80mhz-Ceee wireless-protocol=any security-
profile=default band=5ghz-a/n/ac
[admin@MikroTik] > /interface wireless enable wlan1
```

Configure bridge for Access Point

1. Configure bridge for AP to ensure that 5ghz is working as fail-over. It is required to bridge **wlan1**, **ether1**, and all 60ghz **station interfaces**.
   In the example it shows only 2 station devices but it is possible to add up to 8 devices.

   For ap-bridge device please set **configuration** as follows:

```
[admin@MikroTik] >/interface bridge port
add bridge=bridge hw=no interface=ether1
add bridge=bridge interface=wlan1
add bridge=bridge interface=wlan60-station-1
add bridge=bridge interface=wlan60-station-2
[admin@MikroTik] > interface/bridge/port/pr
# INTERFACE         BRIDGE  HW  PVID  PRIORITY  PATH-COST  INTERNAL-PATH-COST  HORIZON
0 ether1            bridge      no     1  0x80            10                  10  none
1 wlan1             bridge          1  0x80            10                  10  none
2 wlan60-station-1  bridge          1  0x80            10                  10  none
3 wlan60-station-2  bridge          1  0x80            10                  10  none
```

Configure bridge and bonding for station devices

1. Configure bonding and assign slave interfaces in this setup it is selected as built in wlan1 interface, but it can be also ether interface in other kind of setups.

   For station-bridge devices please set **bonding** as:

   ```
   [admin@MikroTik] > /interface bonding add mode=active-backup name=bond1 primary=wlan60-1 slaves=wlan60-
   1,wlan1
   ```

2. Add interface members (ether1 and bond1) to newly created bridge.

   ```
   [admin@MikroTik] > /interface bridge port add interface=ether1 bridge=bridge
   [admin@MikroTik] > /interface bridge port add interface=bond1  bridge=bridge
   [admin@MikroTik] > /interface bridge port print
   Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
    #     INTERFACE                            BRIDGE                              HW   PVID PRIORITY  PATH
   COST    HORIZON

   0     ether1                                 bridge                            yes     1     0x80
    1     bond1
   bridge                             yes     1     0x80         10                 10       none
   ```

Additional configuration

Link should be established after all previously explained steps are done. It's recommended to set up administrator password on both devices.

Troubleshooting

Ensure connection is established to the correct device by checking the device settings like serial number and model name by issuing a command:

```
[admin@MikroTik] > /system routerboard print
```

If bridge wlan60-1 interface in bridge settings is inactive and configuration is done properly to enable the interface on a device - issue a command:

```
[admin@MikroTik] > /interface w60g enable wlan60-1
```

Fail-over PtP CLI example

Summary

This example shows how to configure automatic fail-over (bonding) 5Ghz link in combination with 60Ghz devices in CLI.
When a connection between 60Ghz wireless is lost, it will automatically use the bonded interface.
Example is done from empty configuration state with [WinBox] utility

Connect to the device step by step

1. After configuration reset - only mac-telnet is possible.
   Connect to device by connecting to it's MAC address or use WinBox New terminal to find device MAC address of the W60G device by issuing command:

   ```
   /ip neighbor print
   ```

2. To connect to the W60G device issue a command:

   ```
   /tool mac-telnet mac-address
   ```

3. Enter username and password. By default username is **admin** and password is either blank or printed on the device sticker.

   ```
   [admin@KD_GW] > /tool mac-telnet C4:AD:34:84:EE:5D
   Login: admin
   Password:
   Trying C4:AD:34:84:EE:5D...
   Connected to C4:AD:34:84:EE:5D
   ```

Configure bridge

1. Add new bridge and assign bridge members to it by issuing the following command:

   ```
   /interface bridge add name=bridge
   ```

   To check if the bridge has been created issue a command:

   ```
   [admin@MikroTik] > /interface bridge print
   Flags: X - disabled, R - running
   0 R name="bridge" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled arp-timeout=auto mac-address=1A:7F:
   BB:41:B0:94 protocol-mode=rstp
   fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m priority=0x8000 max-message-age=20s
   forward-delay=15s transmit-hold-count=6
   vlan-filtering=no dhcp-snooping=no
   ```

Set up 60Ghz wireless connection

All previously explained steps are identical to Bridge and Station devices. When configuring wireless interface different modes needs to be used.

For bridge device -

- Choose SSID, Password, frequency and choose bridge mode option that will act as a **bridge** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface w60g set wlan60-1 mode=bridge frequency=auto ssid=MySSID
password=choosepassword isolate-stations=yes
[admin@MikroTik] > /interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=bridge ssid="MySSID"
frequency=auto default-scan-list=58320,60480,62640,64800 password="choosepassword" tx-sector=auto put-
stations-in-bridge=bridge isolate-stations=yes
[admin@MikroTik] > /interface w60g enable wlan60-1
```

For Station device -

- Choose the same SSID, Password, frequency as the bridge device and choose station-bridge mode option that will act as a **station** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface w60g set wlan60-1 mode=station-bridge frequency=auto ssid=MySSID
password=choosepassword
[admin@MikroTik] > /interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=station-bridge ssid="MySSID"
frequency=auto default-scan-list=58320,60480,62640,64800password="choosepassword" tx-sector=auto put-
stations-in-bridge=bridge isolate-stations=yes
[admin@MikroTik] > /interface w60g enable wlan60-1
```

Set up 5Ghz wireless connection

For bridge device -

- Choose SSID, Password, frequency and choose bridge mode option that will act as a **bridge** for the setup, please see the example.
- Enable 5Ghz interface after required parameters have been set.

```
[admin@MikroTik] > /interface wireless security-profiles set [ find default=yes ] supplicant-
identity=MikroTik authentication-types=wpa2-psk mode=dynamic-keys wpa2-pre-shared-key=choosepassword
[admin@MikroTik] > /interface wireless set wlan1 frequency=auto scan-list=default installation=outdoor
mode=bridge ssid=MikroTik1 channel-width=20/40/80mhz-Ceee wireless-protocol=any security-
profile=default band=5ghz-a/n/ac
[admin@MikroTik] > /interface wireless enable wlan1
```

For Station device -

- Choose the same SSID, Password, frequency as the bridge device and choose station-bridge mode option that will act as a **station** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > /interface wireless security-profiles set [ find default=yes ] supplicant-
identity=MikroTik authentication-types=wpa2-psk mode=dynamic-keys wpa2-pre-shared-key=choosepassword
[admin@MikroTik] > /interface wireless set wlan1 frequency=auto scan-list=default installation=outdoor
mode=station-bridge ssid=MikroTik1 channel-width=20/40/80mhz-Ceee wireless-protocol=any security-
profile=default band=5ghz-a/n/ac
[admin@MikroTik] > /interface wireless enable wlan1
```

Configure bridge and bonding

1. Configure bonding and assign slave interfaces in this setup it is selected as built in wlan1 interface, but it can be also ether interface in other kind of setups.

   For bridge device please set **bonding** as:

   ```
   [admin@MikroTik] > /interface bonding add comment=bondingbackup mode=active-backup name=bond1
   primary=wlan60-station-1 slaves=wlan60-station-1,wlan1
   ```

   For station-bridge device please set **bonding** as:

```
[admin@MikroTik] > /interface bonding add comment=defconf mode=active-backup name=bond1 primary=wlan60-
1 slaves=wlan60-1,wlan1
```

2. Add interface members (ether1 and bond1) to newly created bridge.

```
[admin@MikroTik] > /interface bridge port add interface=ether1 bridge=bridge
[admin@MikroTik] > /interface bridge port add interface=bond1  bridge=bridge
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                                BRIDGE                                HW   PVID PRIORITY  PATH
COST    HORIZON

0     ether1                                   bridge                               yes     1     0x80
 1     bond1
bridge                              yes     1     0x80         10                 10       none
```

Additional configuration

Link should be established after all previously explained steps are done. It's recommended to set up administrator password on both devices.

Troubleshooting

Ensure connection is established to the correct device by checking the device settings like serial number and model name by issuing a command:

```
[admin@MikroTik] > /system routerboard print
```

If bridge wlan60-1 interface in bridge settings is inactive and configuration is done properly to enable the interface on a device - issue a command:

```
[admin@MikroTik] > /interface w60g enable wlan60-1
```

Fail-over PtP GUI example

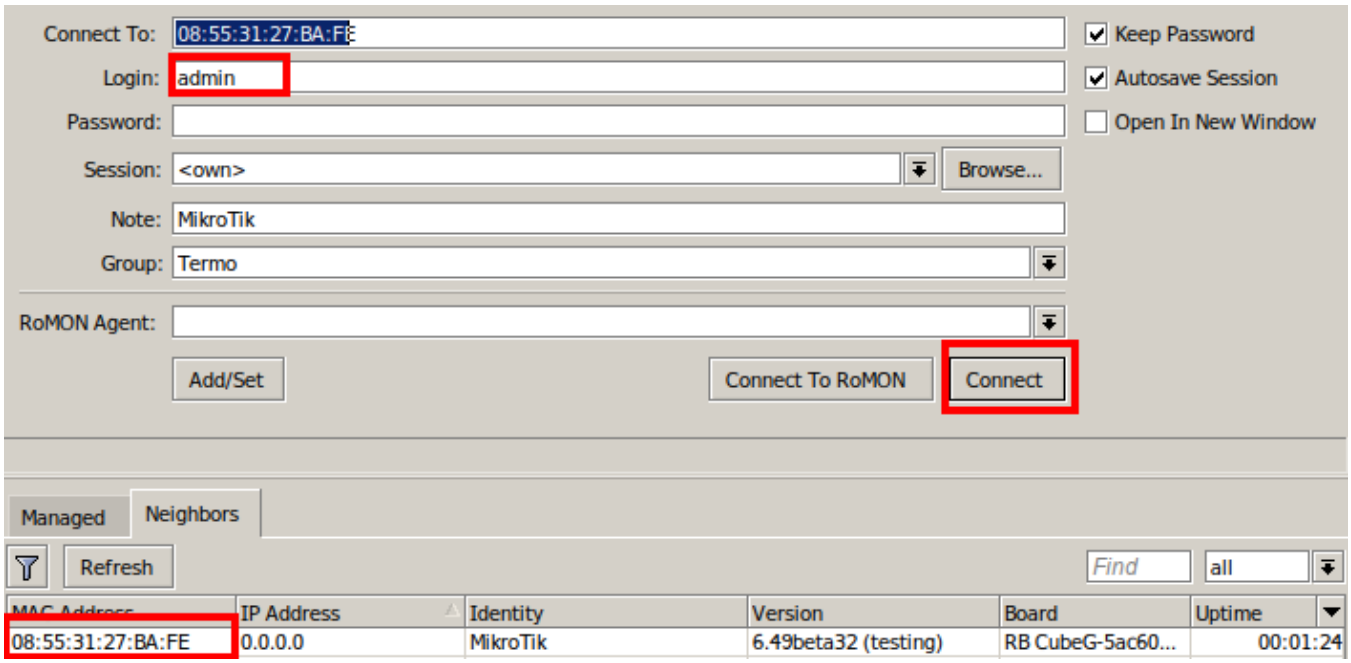
Summary

This example shows how to configure automatic fail-over (bonding) 5Ghz link in combination with 60Ghz devices in GUI.
When a connection between 60Ghz wireless is lost, it will automatically use the bonded interface.
Example is done from empty configuration state with [WinBox] utility

Connect to the device

After configuration reset - only mac-telnet is possible. In main WinBox screen press on Neighbours, choose your devices MAC address and press Connect:

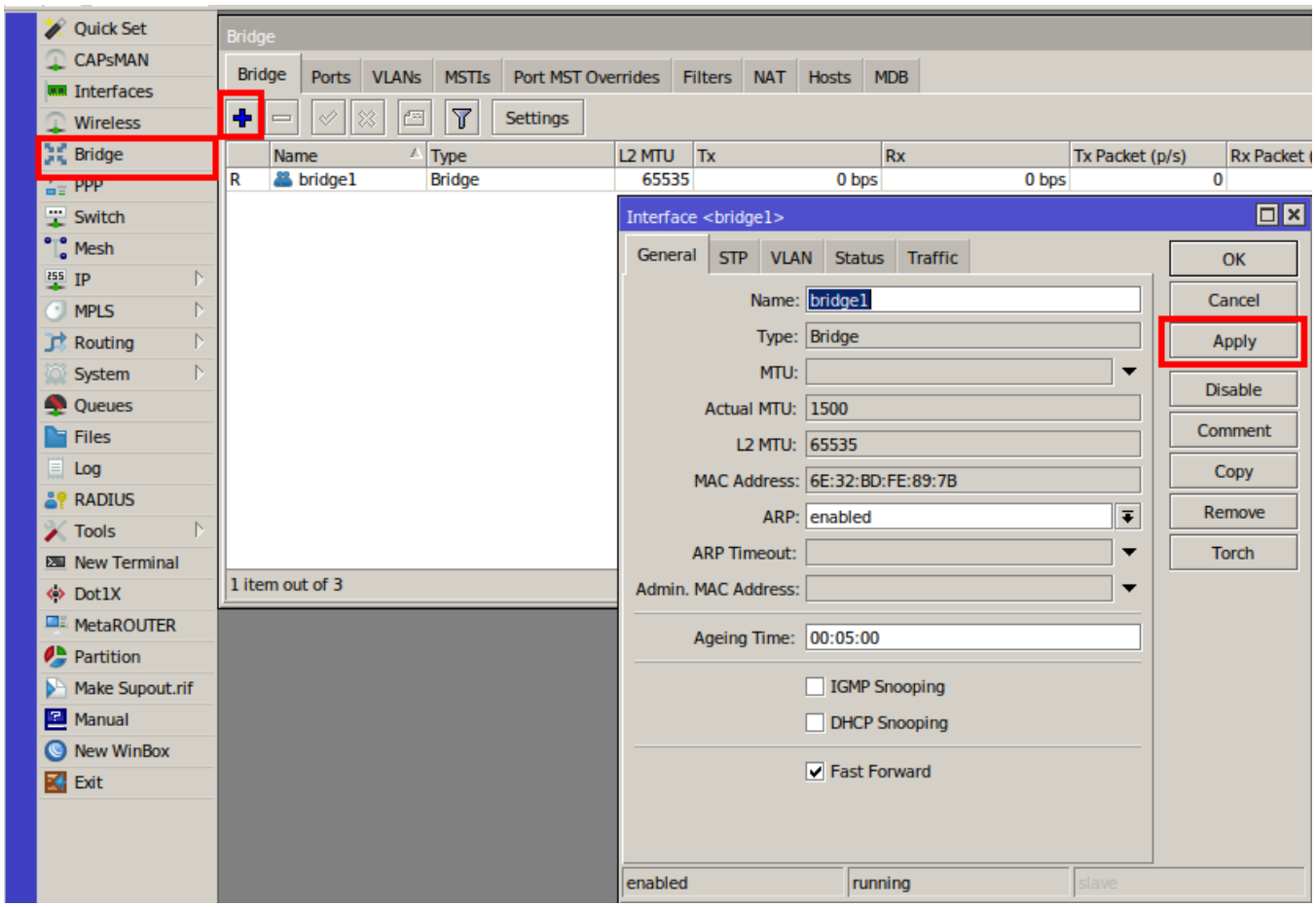
1. Select correct device **MAC Address;**
2. Login by default is "**admin**" and no password is set;
3. Press **Connect.**



Configure bridge

Add new bridge.

1. Open Bridge sub-menu;
2. Press on "+" to add new bridge;
3. Apply your changes.



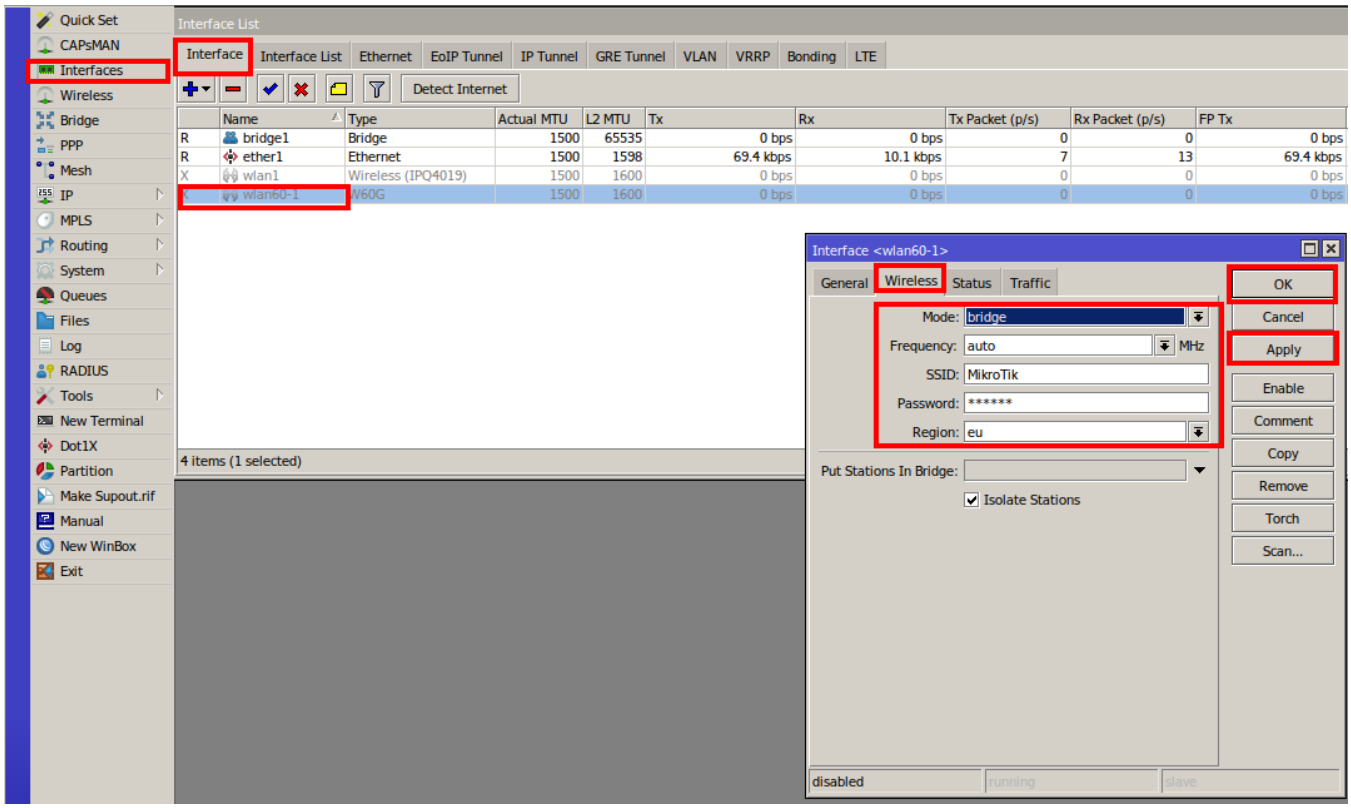
Later in the instructions it requires to assign bridge members to it. This will allow to pass traffic from Ethernet to W60G interface without routing.

Set up 60Ghz wireless connection

All previously explained steps are identical to **bridge** and **station** devices. Different modes needs to be used when configuring wireless interfaces.

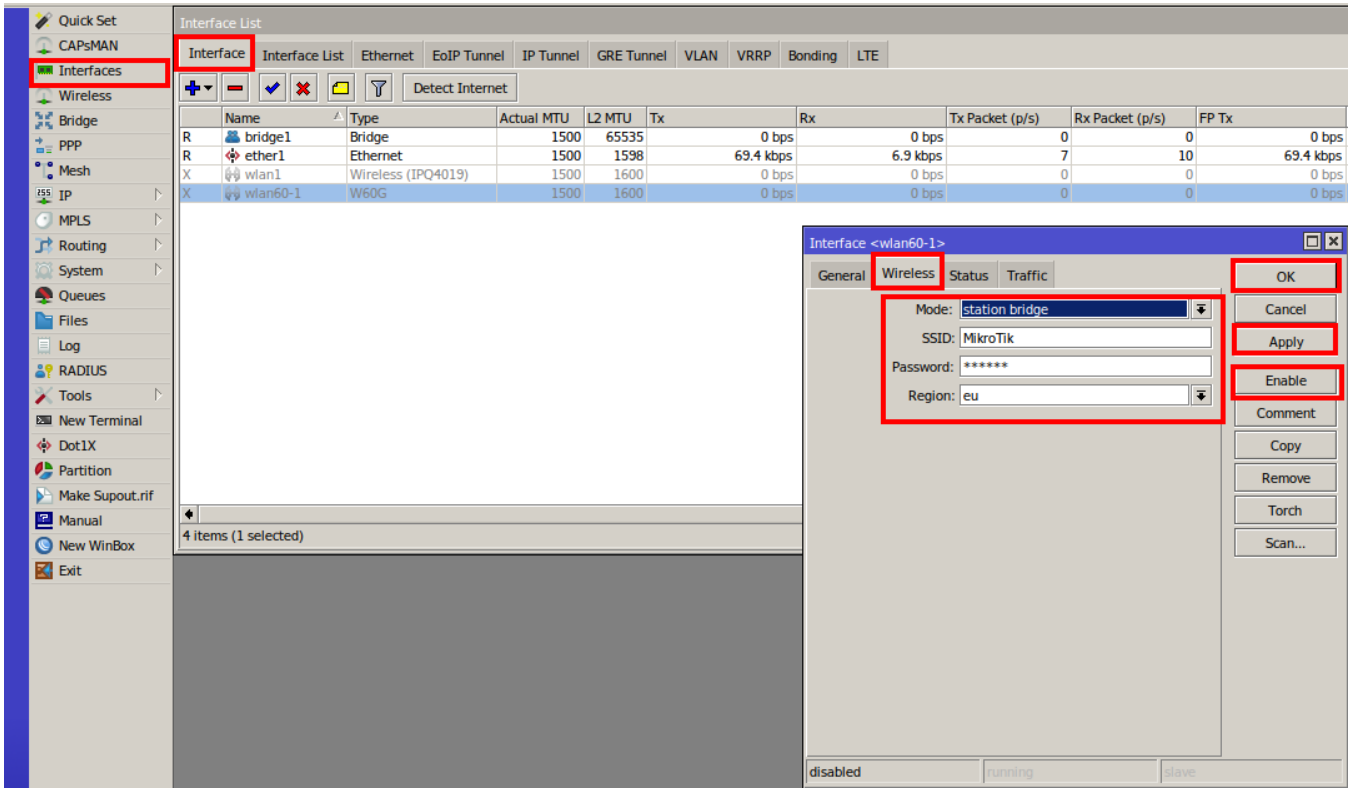
Configure **bridge** device as follows:

1. Open Interface menu;
2. Double click on wlan60-1 interface;
3. Press on Wireless sub-menu and set mode to **bridge (**or **ap-bridge** for PtmP);
4. Set SSID and password and region;
5. Select previously created bridge under "Put Stations In Bridge";
6. Apply your changes;
7. Press enable to start transmitting.



Configure **station** device as follows:

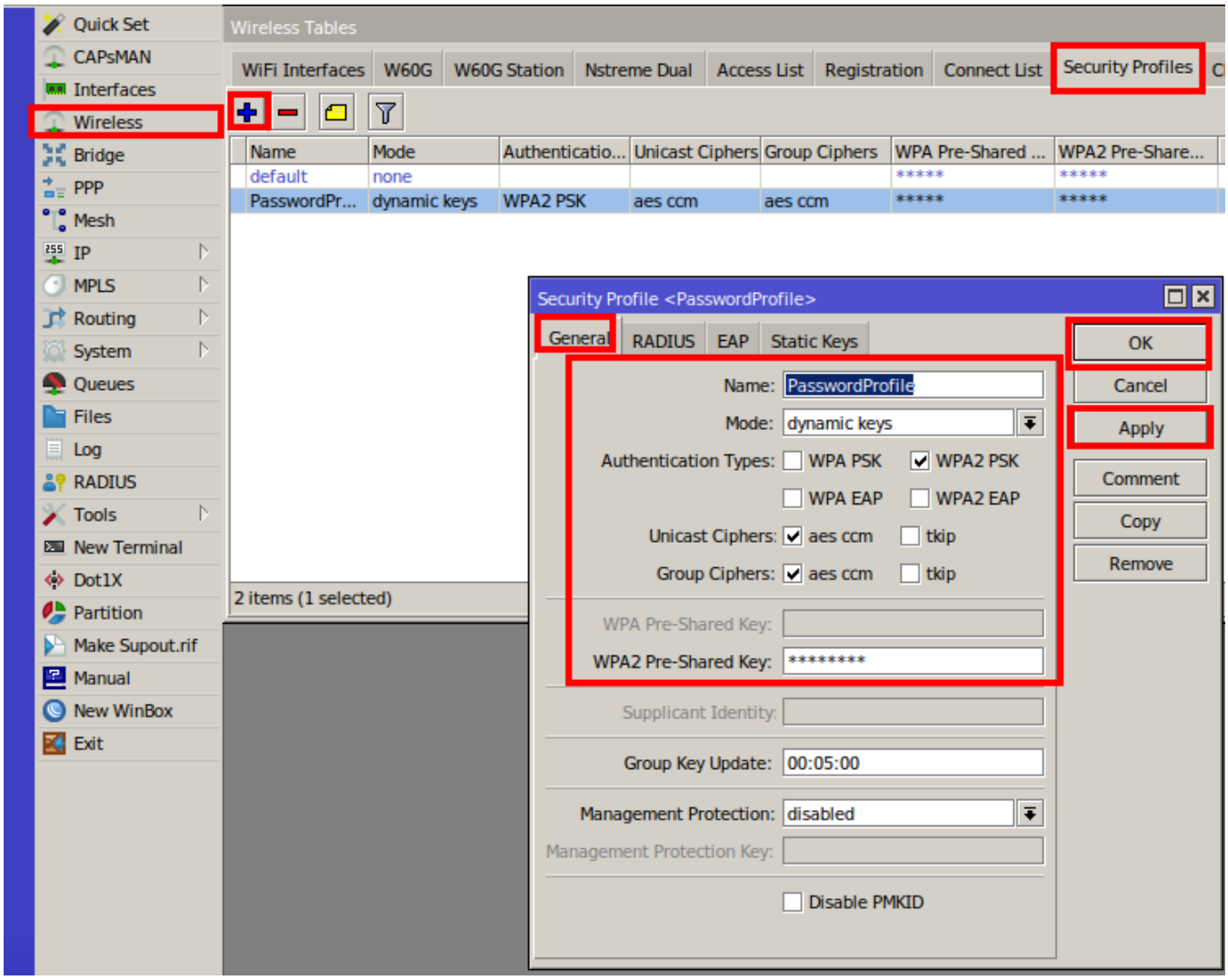
1. Open Interface menu;
2. Double click on wlan60-1 interface;
3. Press on Wireless sub-menu and set mode to **station bridge;**
4. Set SSID and password;
5. Apply your changes;
6. Press enable to start transmitting.



Set up 5Ghz wireless connection

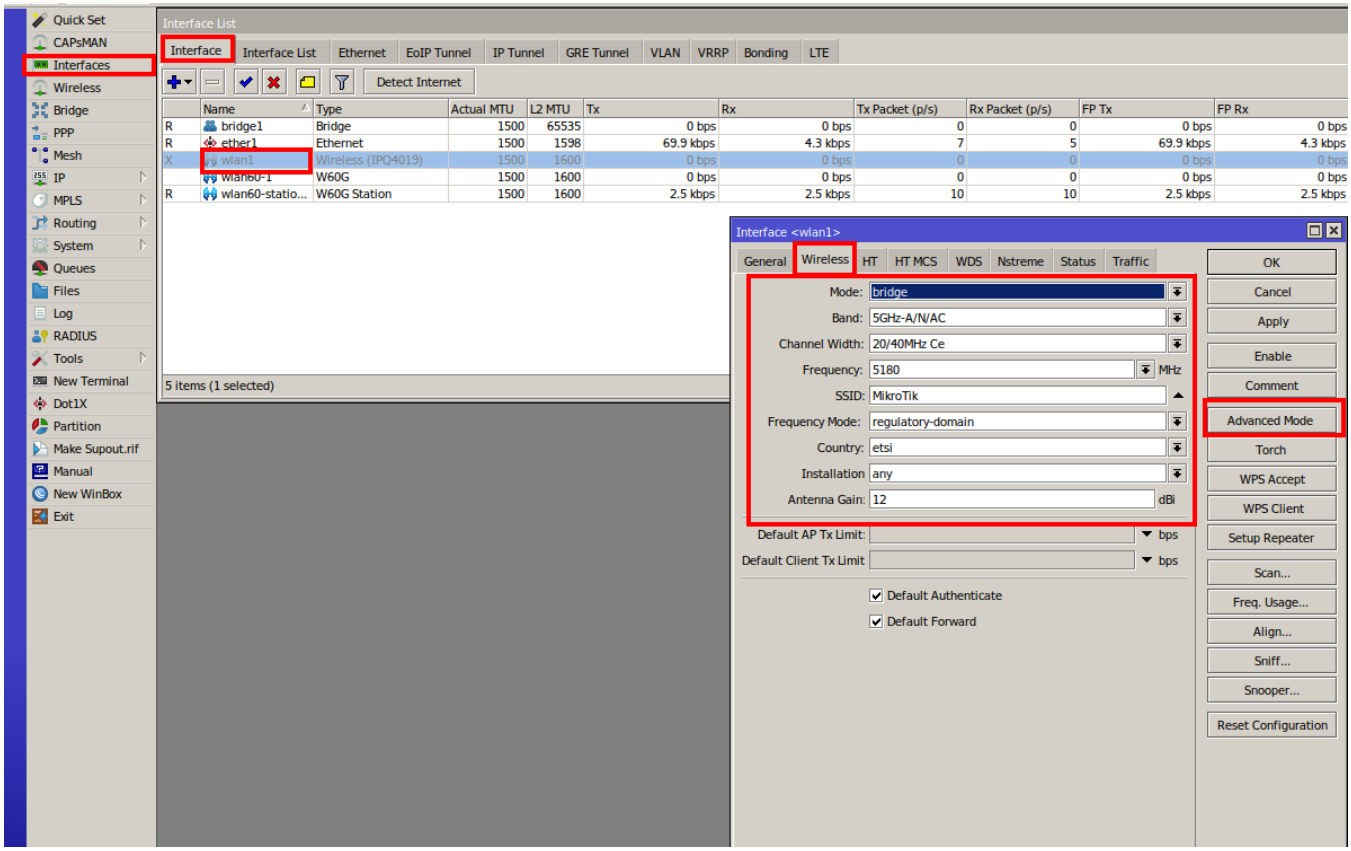
Choose Security Profile for your devices -

1. Choose **Wireless** menu
2. Choose **Security Profiles** sub-menu
3. Add new profile with "**+**" sign
4. Choose **name**, **mode**, **authentication type** and a secure password.
5. **Apply** the configuration.

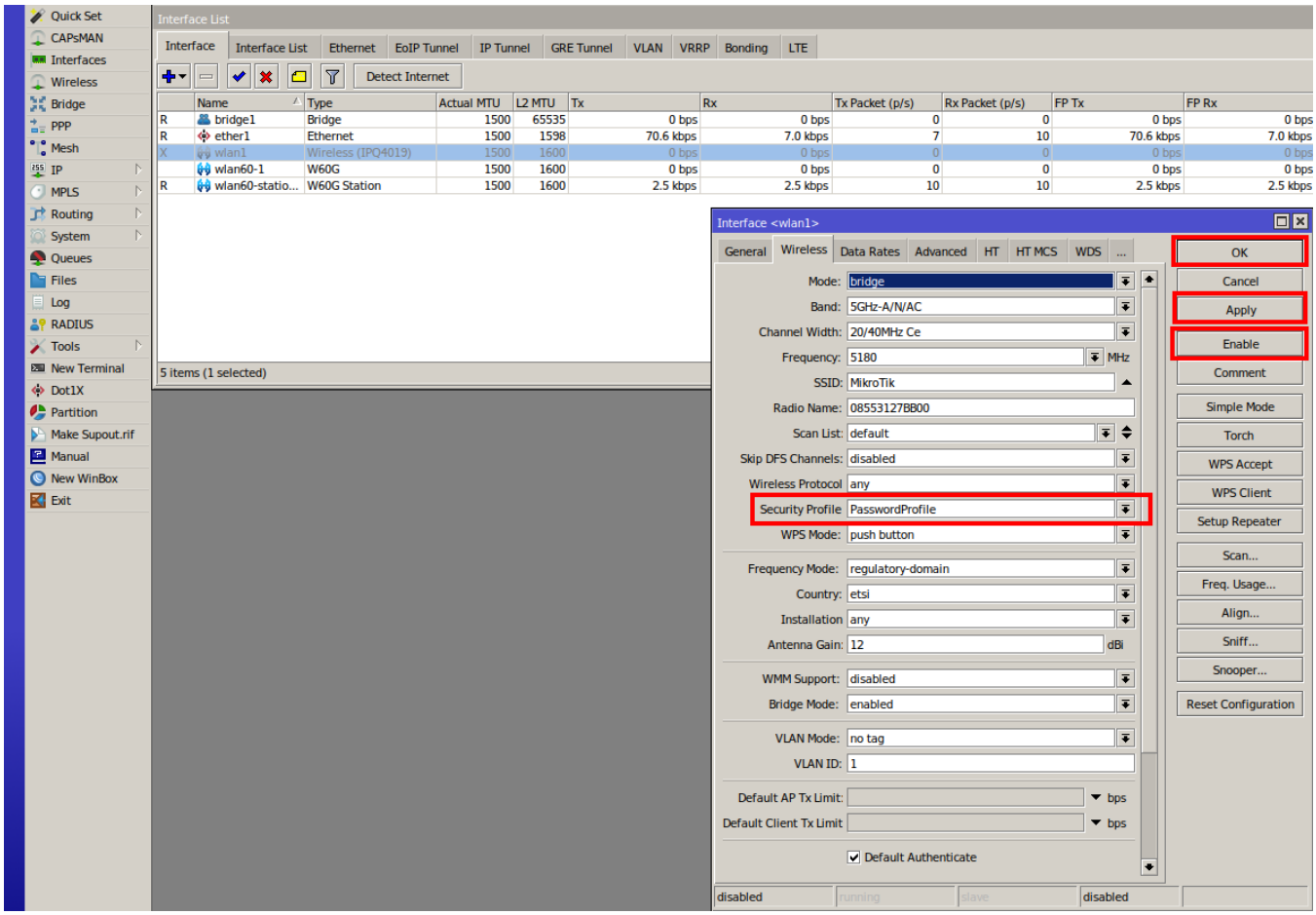


For bridge device -

1. Open **Interfaces** menu;
2. Double click on **wlan1** interface;
3. Press on **Wireless** sub-menu and set mode to **bridge** (or **ap-bridge** for PtmP);
4. Set **SSID**, **password** and **country.**
5. Press on **Advanced Mode.**

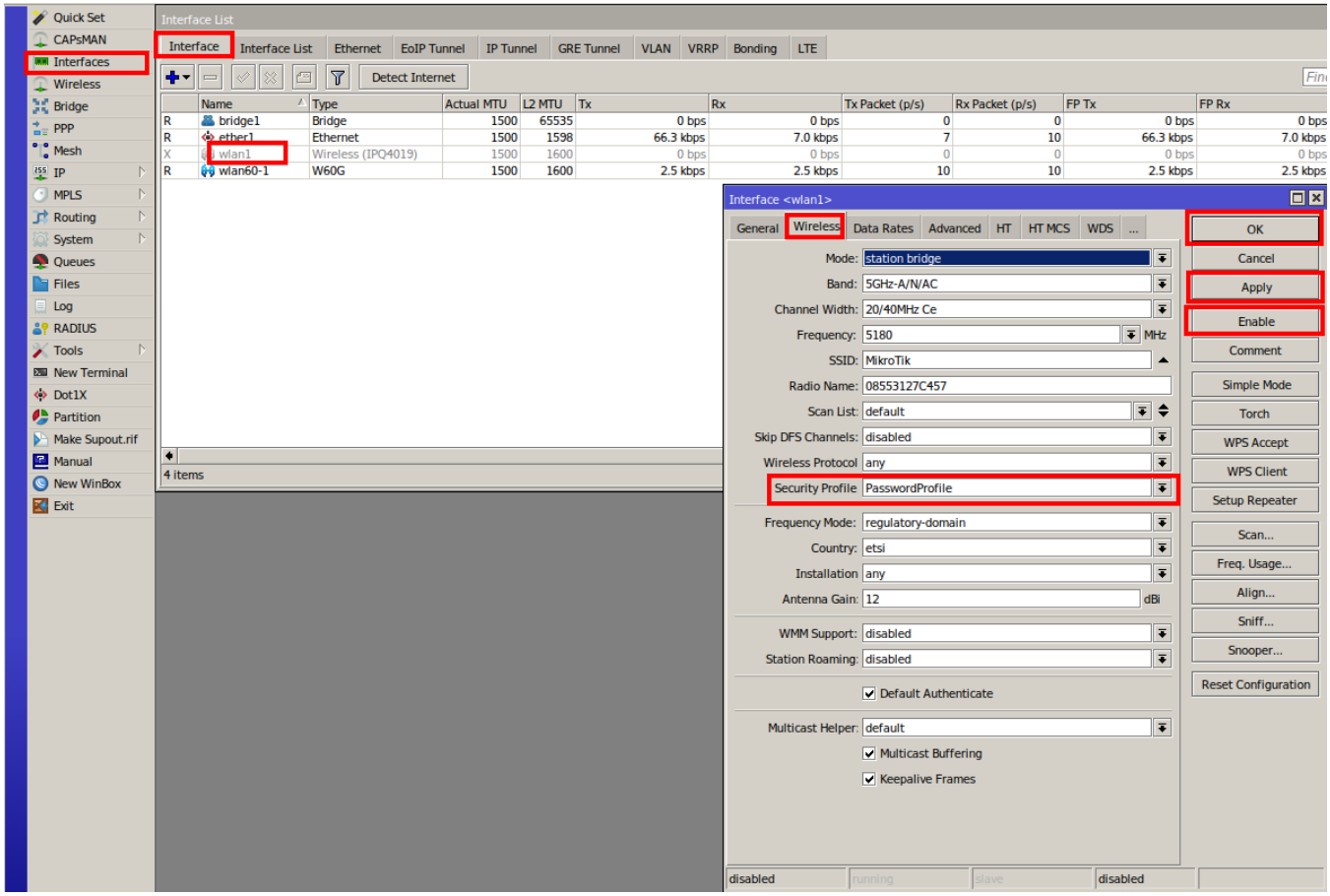


1. Choose your **Security Profile**;
2. **Apply** your changes;
3. Press **enable** to start transmitting.

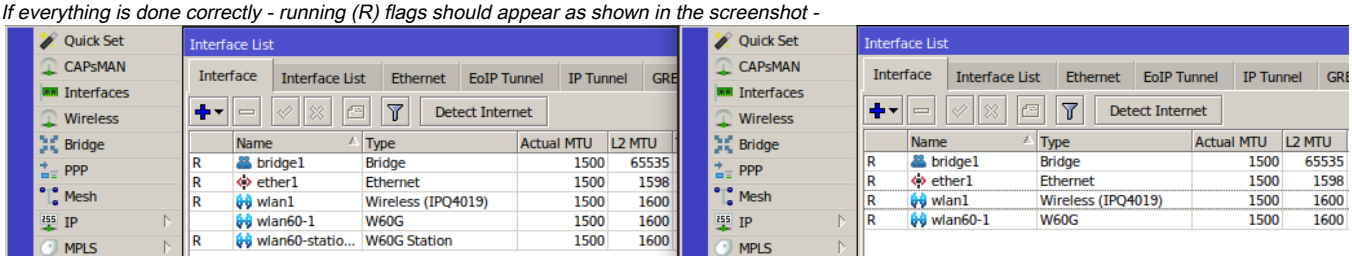


For station device -

1. Open **Interfaces** menu;
2. Double click on **wlan1** interface;
3. Press on **Wireless** sub-menu and set mode to **station-bridge**;
4. Set **SSID**, **password** and **country**;
5. Press on **advanced** mode (similar to bridge device*);
6. Choose **Security Profile**;
7. **Apply** your changes;
8. Press **enable** to start transmitting.



If everything is done correctly - running (R) flags should appear as shown in the screenshot -

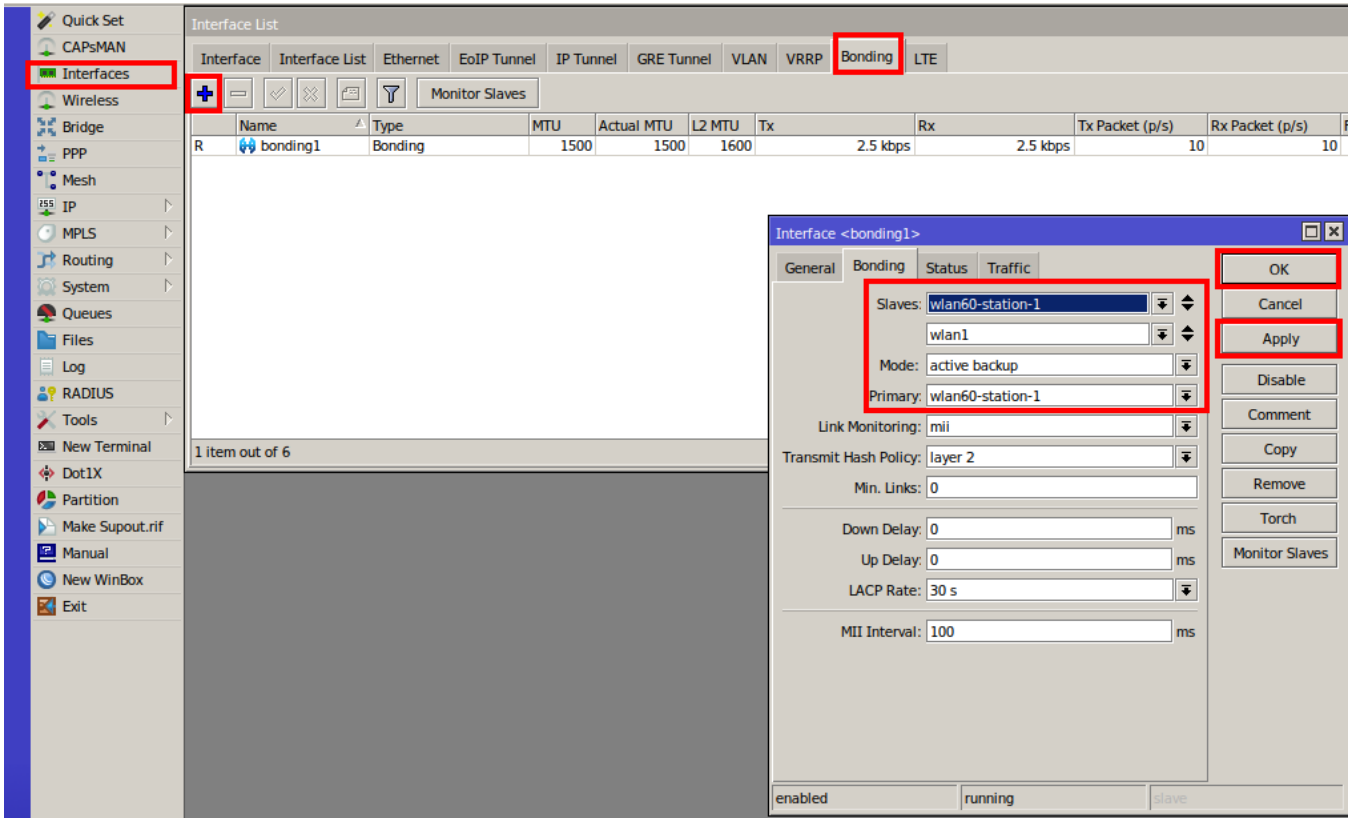


Configure bonding

Configure bonding and assign slave interfaces in this setup it is selected as built in wlan1 interface, but it can be also ether interface in other kind of setups.

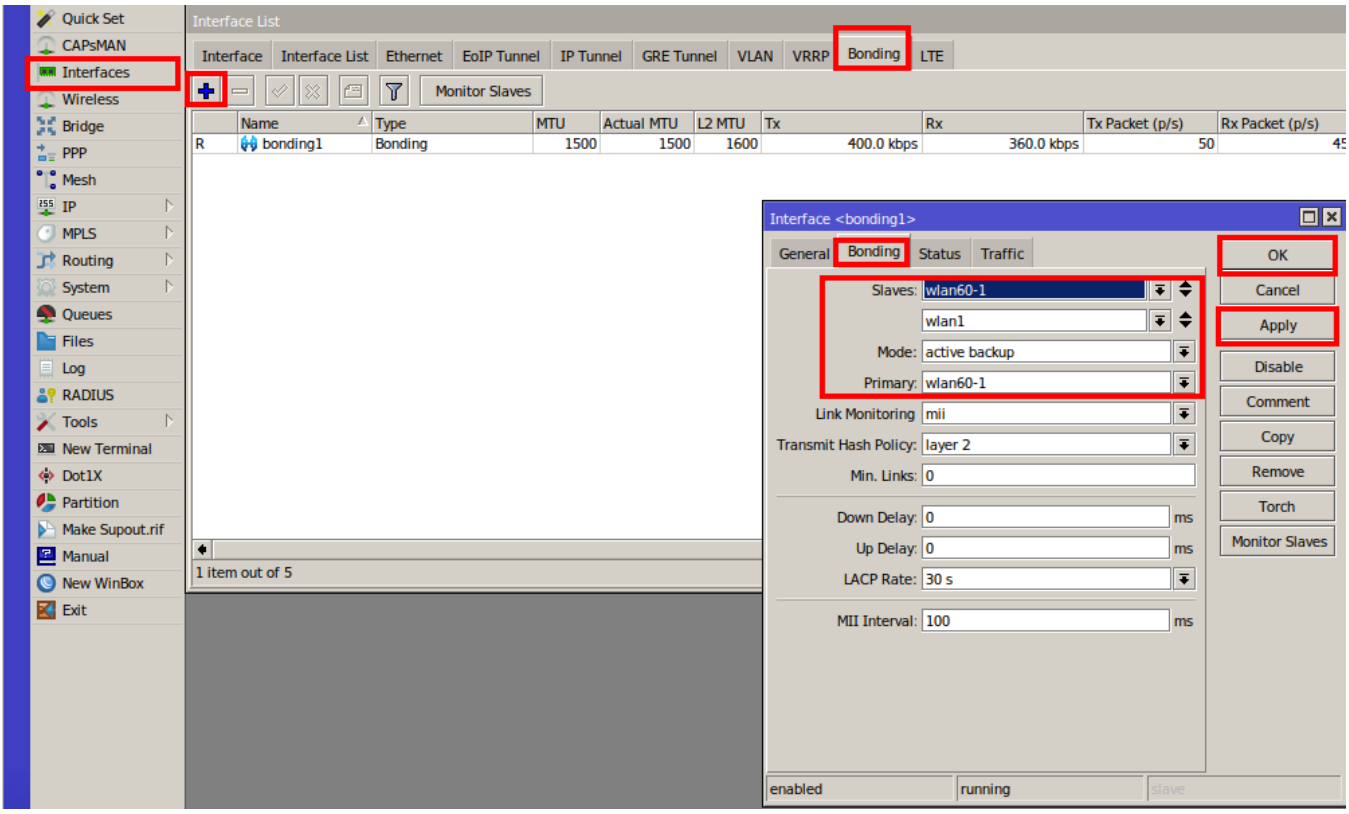
For bridge device -

1. Press on **Bonding** sub-menu;
2. Add new member with "**+**";
3. Add interface members (**wlan1** and **wlan60-station-1**) to **bonding** interface as **Slaves;**
4. Add interface member **wlan60-station-1** as **Primary** interface;
5. Choose Mode as **active backup**;
6. **Apply** configuration.



For station device -

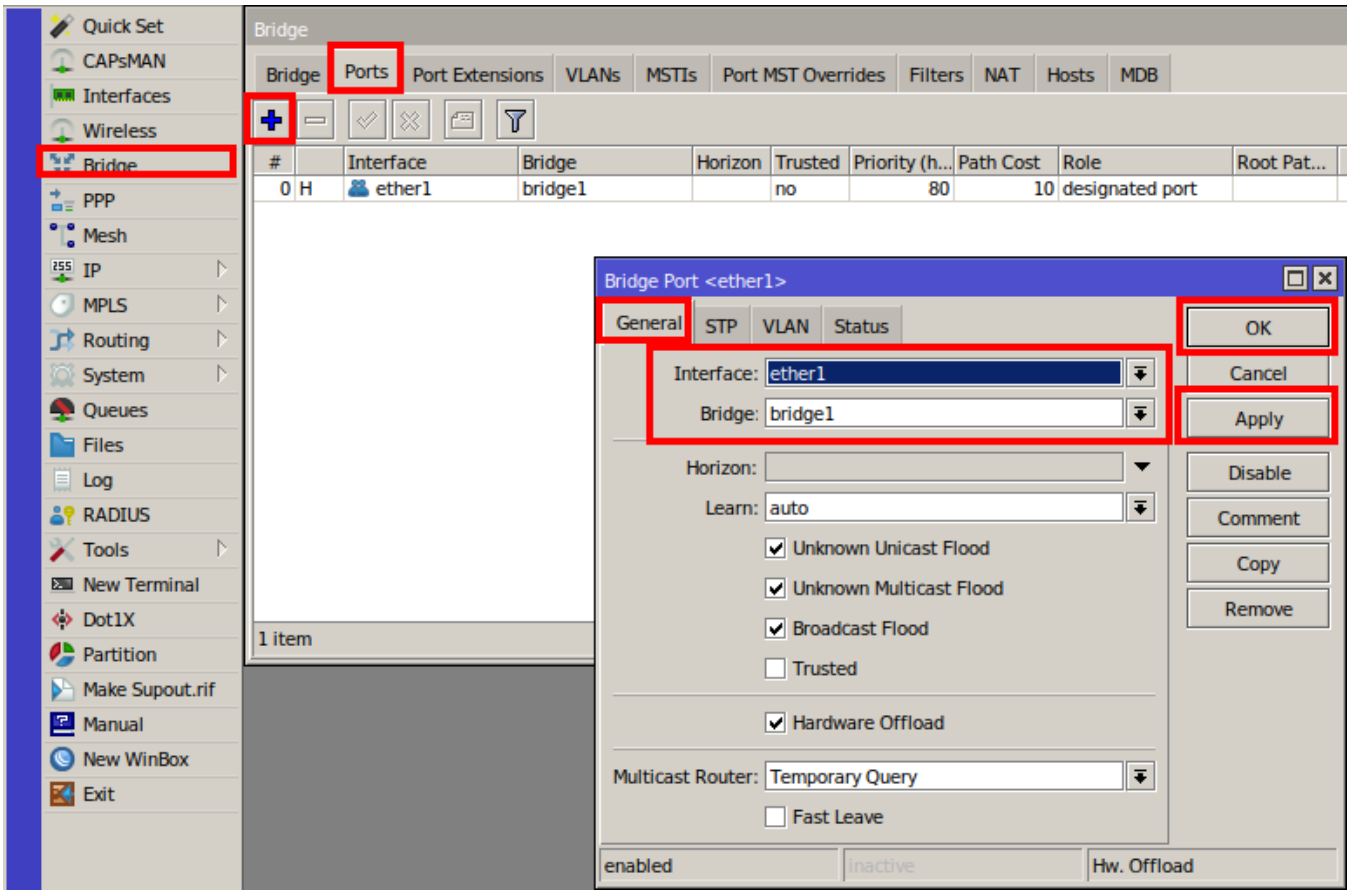
1. Press on **Bonding** sub-menu;
2. Add new member with "**+**";
3. Add interface members (**wlan1** and **wlan60-1**) to **bonding** interface as **Slaves;**
4. Add interface member **wlan60-1** as **Primary** interface;
5. Choose Mode as **active backup**;
6. **Apply** configuration.



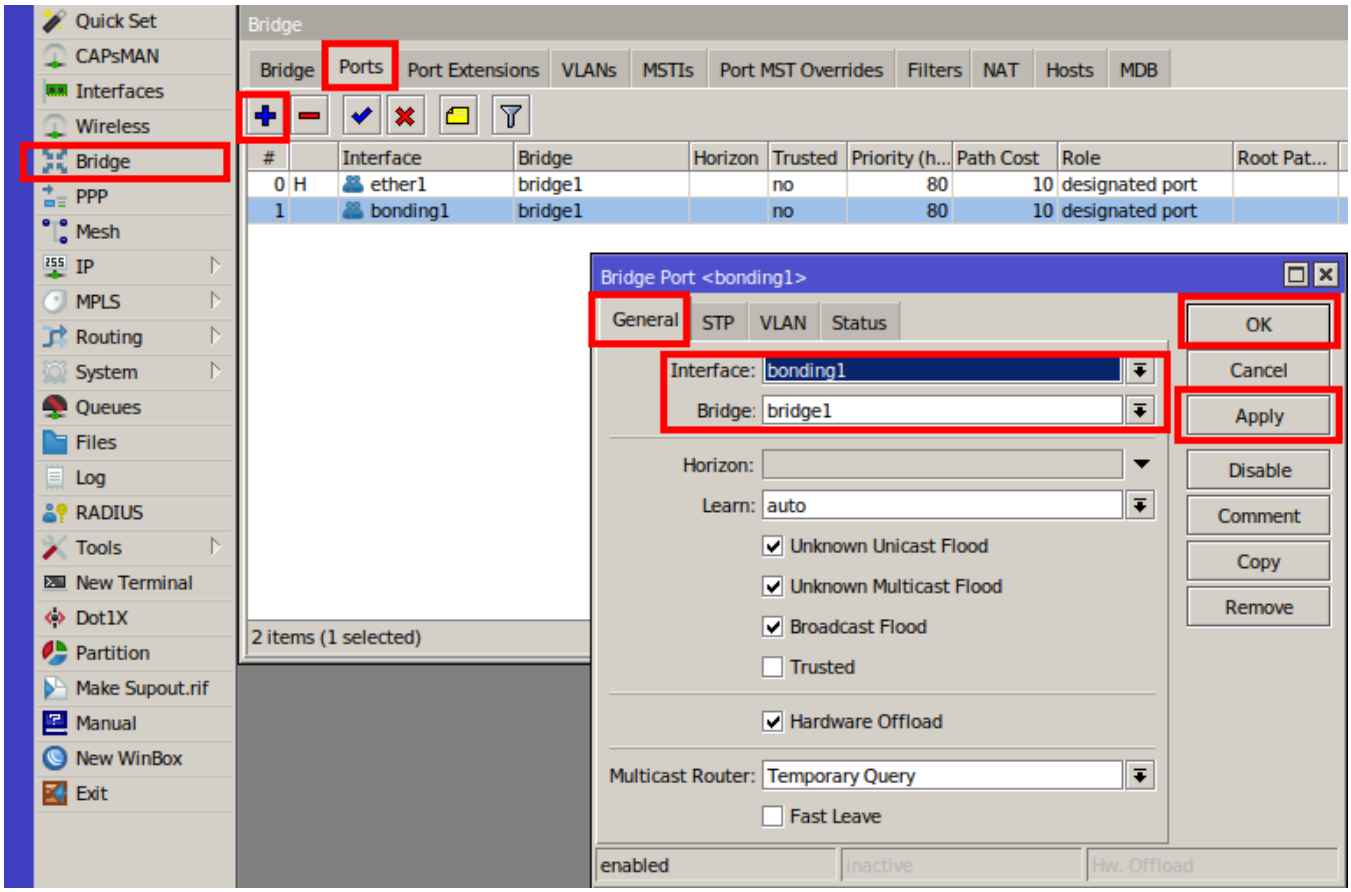
Configure bridge

Configuring bridge settings including the bonding interface is mandatory for the active-backup to work on used devices (In this case bridge and station devices settings are the same).

1. Press on **Bridge** sub-menu;
2. Add new member with "**+**";
3. Add interface member as **ether1** and Bridge member as **bridge1;**
4. **Apply** configuration.



1. Press on **Bridge** sub-menu;
2. Add new member with "**+**";
3. Add interface member as **bonding1** and Bridge member as **bridge1;**
4. **Apply** configuration.



Additional configuration

Interfaces when enabled from greyed out will become active.

Link should be established after all previously explained steps are done. It's recommended to set up administrator password on both devices.

PtP CLI example

Summary

This example shows how to configure transparent wireless bridge in CLI from one W60G device to another.

Example is done from empty configuration state with [WinBox] utility

Connect to the device step by step

1. After configuration reset - only mac-telnet is possible.
   Connect to device by connecting to it's MAC address or use WinBox New terminal to find device MAC address of the W60G device by issuing command:

   ```
   /ip neighbor print
   ```

2. To connect to the W60G device issue a command:

   ```
   /tool mac-telnet mac-address
   ```

3. Enter username and password. By default username is **admin** and no password is set

   ```
   [admin@KD_GW] > /tool mac-telnet C4:AD:34:84:EE:5D
   Login: admin
   Password:
   Trying C4:AD:34:84:EE:5D...
   Connected to C4:AD:34:84:EE:5D
   ```

Configure bridge

1. Add new bridge and assign bridge members to it by issuing the following command:

   ```
   /interface bridge add name=bridge
   ```

   To check if the bridge has been created issue a command:

   ```
   [admin@MikroTik] > /interface bridge print
   Flags: X - disabled, R - running
   0 R name="bridge" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled arp-timeout=auto mac-address=1A:7F:
   BB:41:B0:94 protocol-mode=rstp
   fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m priority=0x8000 max-message-age=20s
   forward-delay=15s transmit-hold-count=6
   vlan-filtering=no dhcp-snooping=no
   ```

2. Add interface members (ether1 and wlan60-1) to newly created bridge.

   ```
   [admin@MikroTik] > /interface bridge port add interface=ether1 bridge=bridge
   [admin@MikroTik] > /interface bridge port add interface=wlan60-1 bridge=bridge
   [admin@MikroTik] > /interface bridge port print
   Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
    #     INTERFACE                                BRIDGE                                HW   PVID PRIORITY  PATH
   COST    HORIZON

   0     ether1                                   bridge                               yes     1     0x80
    1 I   wlan60-
   1                                bridge                                     1     0x80         10
   ```

Set up wireless connection

All previously explained steps are identical to Bridge and Station devices. When configuring wireless interface different modes needs to be used.

For bridge device -

- Choose SSID, Password, frequency and choose bridge mode option that will act as a **bridge** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > interface w60g set wlan60-1 mode=bridge frequency=auto ssid=MySSID
password=choosepassword put-stations-in-bridge=bridge isolate-stations=yes
[admin@MikroTik] > interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=bridge ssid="MySSID"
frequency=auto default-scan-list=58320,60480,62640,64800 password="choosepassword" tx-sector=auto put-
stations-in-bridge=bridge isolate-stations=yes
[admin@MikroTik] > interface w60g enable wlan60-1
```

For Station device -

- Choose the same SSID, Password, frequency as the bridge device and choose station-bridge mode option that will act as a **station** for the setup, please see the example.
- Enable W60G interface after required parameters have been set.

```
[admin@MikroTik] > interface w60g set wlan60-1 mode=station-bridge frequency=auto ssid=MySSID
password=choosepassword
[admin@MikroTik] > interface w60g print
Flags: X - disabled, R - running
0 X name="wlan60-1" mtu=1500 l2mtu=1600 mac-address=C4:AD:34:84:EE:5E arp=enabled arp-timeout=auto
region=no-region-set mode=station-bridge
ssid="MySSID" frequency=auto default-scan-list=58320,60480,62640,64800 password="choosepassword" tx-
sector=auto put-stations-in-bridge=bridge isolate-stations=yes
[admin@MikroTik] > /interface w60g enable wlan60-1
```

Additional configuration

Link should be established after all previously explained steps are done. It's recommended to set up administrators password on both devices.

Troubleshooting

Ensure connection is established to the correct device by checking the device settings like serial number and model name by issuing a command:

```
[admin@MikroTik] > /system routerboard print
```

If bridge wlan60-1 interface in bridge settings is inactive and configuration is done properly to enable the interface on a device - issue a command:

```
[admin@MikroTik] > /interface w60g enable wlan60-1
```

PtP GUI example

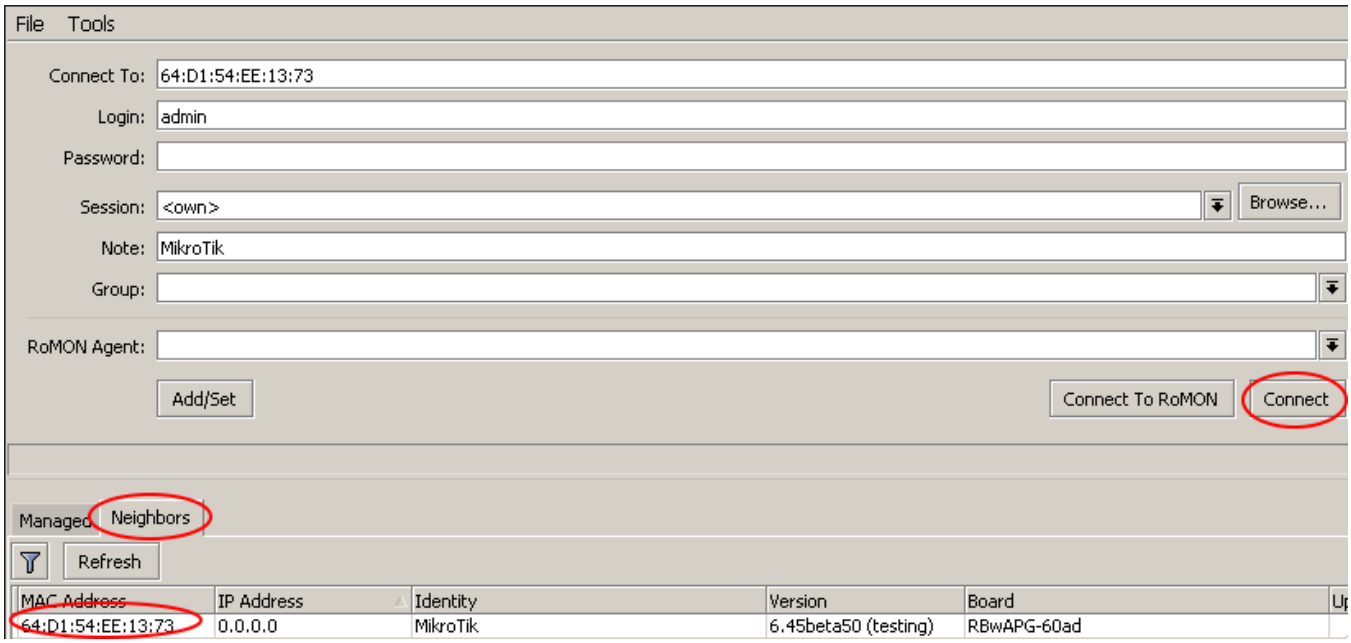
Summary

This example shows how to configure transparent wireless bridge in GUI from one W60G device to another.

Example is done from empty configuration state with [WinBox] utility

Connect to the device

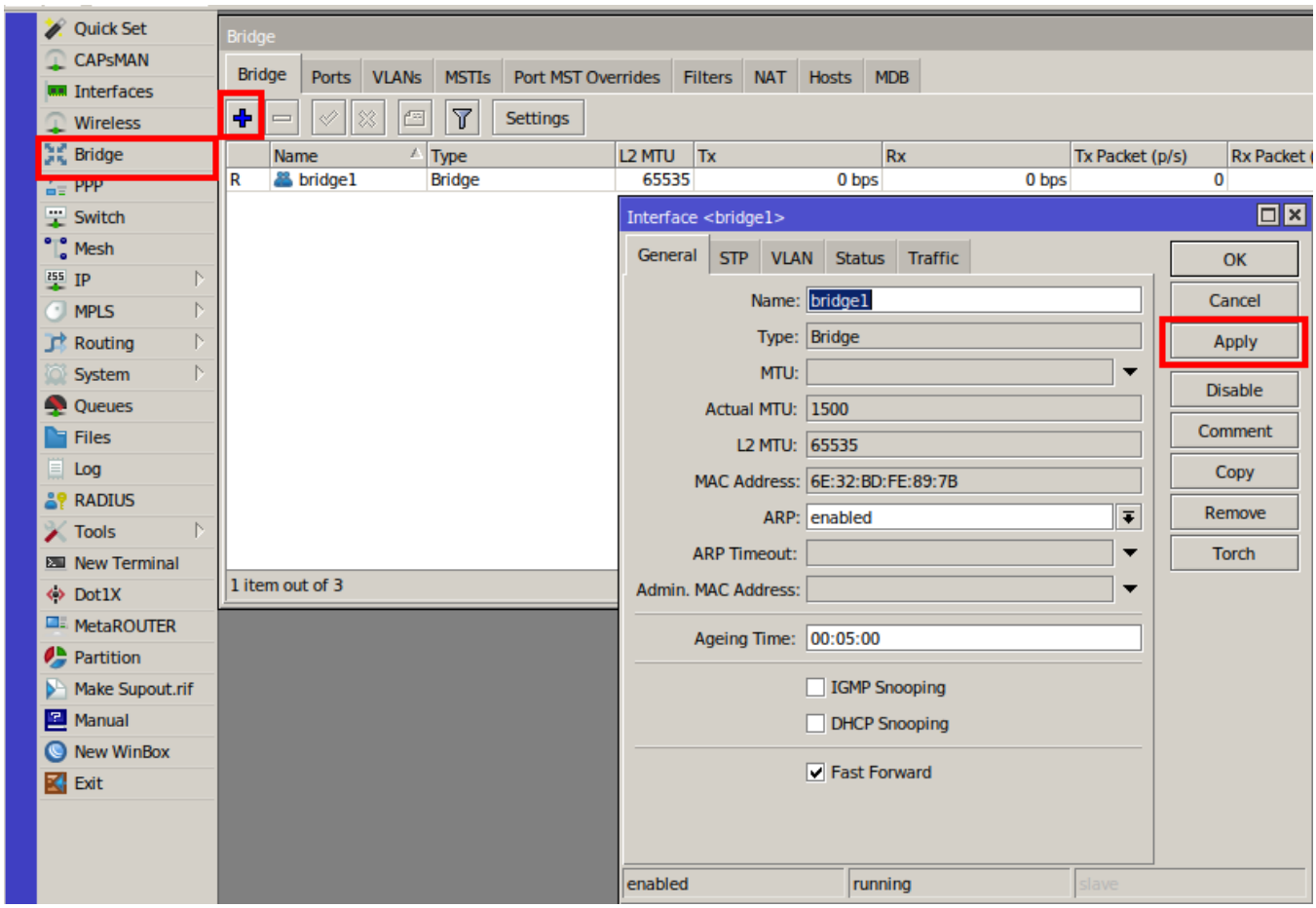
After configuration reset - only mac-telnet is possible. In main WinBox screen press on Neighbours, choose your devices MAC address and press Connect:



Configure bridge

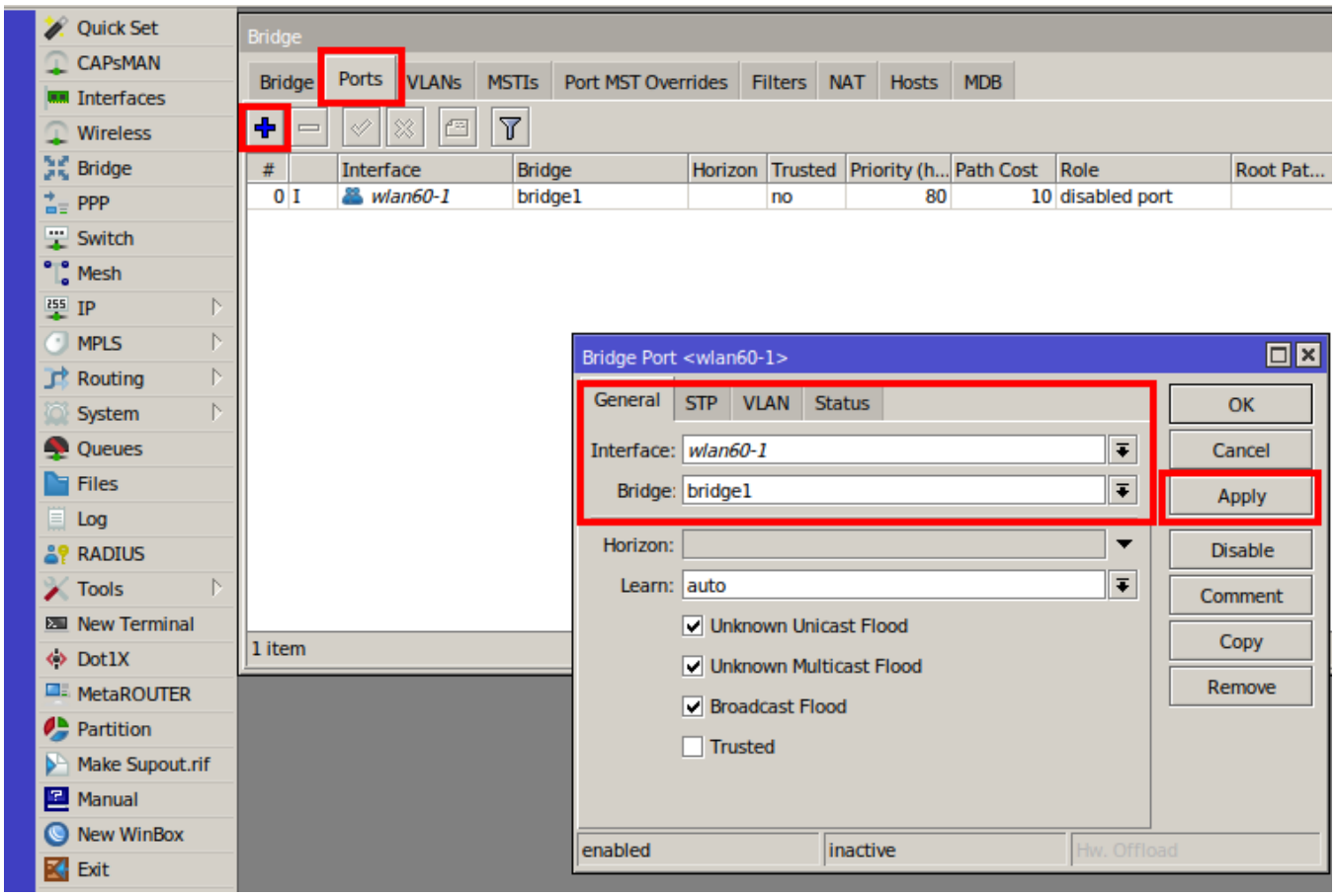
Add new bridge and assign bridge members to it. This will allow to pass traffic from from Ethernet to W60G interface without routing.

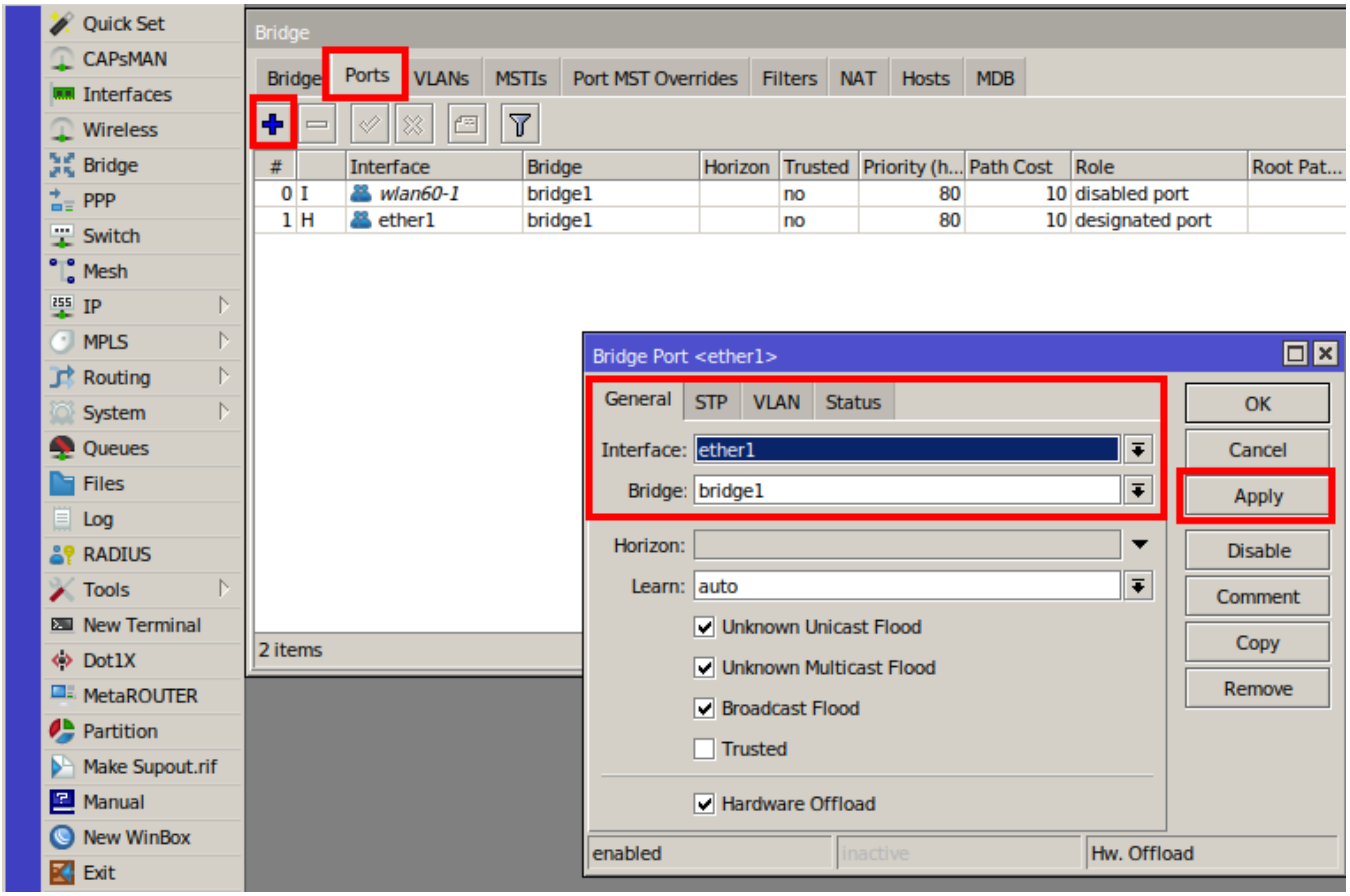
1. Open Bridge sub-menu;
2. Press on "+" to add new bridge;
3. Apply your changes.



Add interface members (ether1 and wlan60-1) to newly created bridge.

1. Press on Ports sub-menu;
2. Add new member with "+";
3. Select correct interfaces;
4. Apply the settings.


Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Tools
New Terminal
Dot1X
MetaROUTER
Partition
Make Supout.rif
Manual
New WinBox
Exit
Bridge
Bridge
Ports
VLANs
MSTIs
Port MST Overrides
Filters
NAT
Hosts
MDB
Interface Bridge Horizon Trusted Priority (h... Path Cost Role Root Pat...
0 I wlan60-1 bridge1 no 80 10 disabled port
1 item
Bridge Port <wlan60-1>
General
STP
VLAN
Status
Interface: wlan60-1
Bridge: bridge1
Horizon:
Learn: auto
Unknown Unicast Flood
Unknown Multicast Flood
Broadcast Flood
Trusted
OK
Cancel
Apply
Disable
Comment
Copy
Remove
enabled
inactive
Hw. Offload

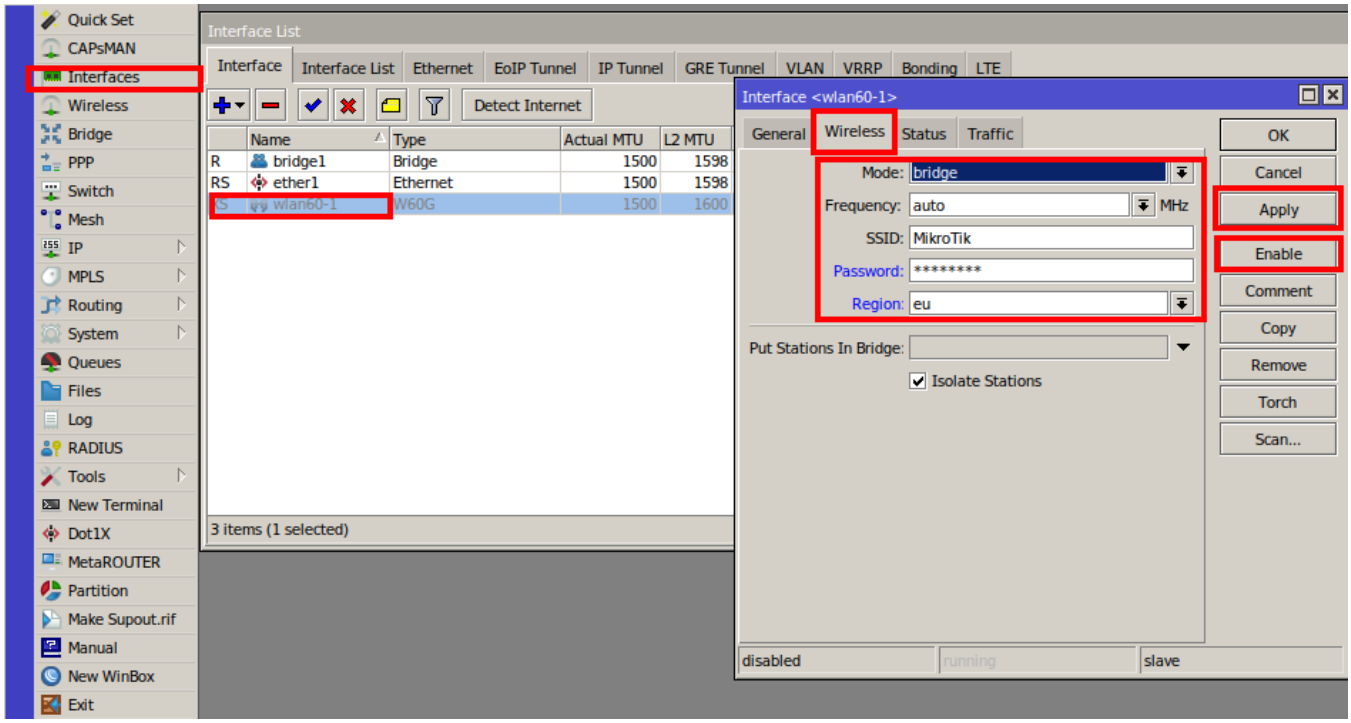

Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Tools
New Terminal
Dot1X
MetaROUTER
Partition
Make Supout.rif
Manual
New WinBox
Exit
Bridge
Bridge
Ports
VLANs
MSTIs
Port MST Overrides
Filters
NAT
Hosts
MDB
Interface Bridge Horizon Trusted Priority (h... Path Cost Role Root Pat...
0 I wlan60-1 bridge1 no 80 10 disabled port
1 H ether1 bridge1 no 80 10 designated port
2 items
Bridge Port <ether1>
General
STP
VLAN
Status
Interface: ether1
Bridge: bridge1
Horizon:
Learn: auto
Unknown Unicast Flood
Unknown Multicast Flood
Broadcast Flood
Trusted
Hardware Offload
OK
Cancel
Apply
Disable
Comment
Copy
Remove
enabled
inactive
Hw. Offload

Set up wireless connection

All previously explained steps are identical to **bridge** and **station** devices. Different modes needs to be used when configuring wireless interfaces.

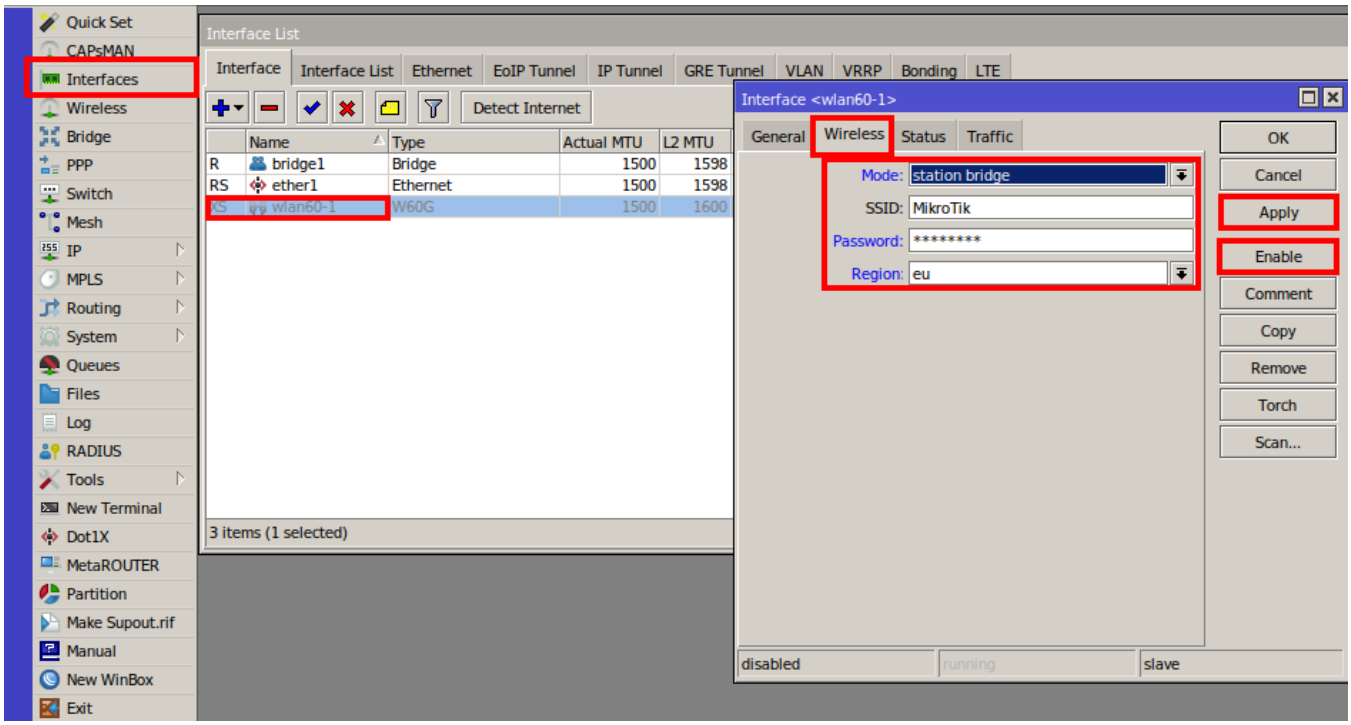
Configure **bridge** device as follows:

1. Open Interface menu;
2. Double click on wlan60-1 interface;
3. Press on Wireless sub-menu and set mode to **bridge;**
4. Set SSID and password and region;
5. Select previously created bridge under "Put Stations In Bridge";
6. Apply your changes;
7. Press enable to start transmitting.



Configure **station** device as follows:

1. Open Interface menu;
2. Double click on wlan60-1 interface;
3. Press on Wireless sub-menu and set mode to **station bridge;**
4. Set SSID and password;
5. Apply your changes;
6. Press enable to start transmitting.



Additional configuration

Interfaces when enabled from greyed out will become active.

Link should be established after all previously explained steps are done. It's recommended to set up administrator password on both devices.

To create point to multi-point setup: On bridge device ap-bridge must be set and station-bridge for stations.

CAPsMAN

CAPsMAN AAA

Settings to configure CAPsMAN AAA functionality are found in the **/caps-man aaa** menu:

Property	Description
mac-format (*string*; Default: **XX:XX:XX:XX:XX:XX**)	Controls how the MAC address of the client is encoded by Access Point in the User-Name attribute of the MAC authentication and MAC accounting RADIUS requests.
mac-mode (*as-username | as-username-and-password*; Default: **as username**)	By default Access Point uses an empty password, when sending Access-Request during MAC authentication. When this property is set to as-username-and-password, Access Point will use the same value for the User-Password attribute as for the User-Name attribute.
mac-caching (*disabled | time-interval*; Default: **disabled**)	If this value is set to a time interval, the Access Point will cache RADIUS MAC authentication responses for a specified time, and will not contact the RADIUS server if matching cache entry already exists. The value disabled will disable the cache, Access Point will always contact the RADIUS server.
interim-update (*disabled | time-interval*; Default: **disabled**)	When RADIUS accounting is used, Access Point periodically sends accounting information updates to the RADIUS server. This property specifies the default update interval that can be overridden by the RADIUS server using the Acct-Interim-Interval attribute.
called-format (*mac | mac:ssid | ssid*; Default: **mac:ssid**)	Format of how the "called-id" identifier will be passed to RADIUS. When configuring radius server clients, you can specify "called-id" in order to separate multiple entires.

Example

Assuming that rest of the settings are already configured and only the "Security" part has been left.

Radius authentication with one server

1. Create CAPsMAN security configuration

2. Configure Radius server client

3. Assign the configuration to your master profile (or directly to CAP itself)

```
/caps-man security add authentication-types=wpa2-eap eap-methods=passthrough encryption=aes-ccm group-
encryption=aes-ccm name=radius
/radius add address=x.x.x.x secret=SecretUserPass service=wireless
/caps-man configuration set security=radius
```

Radius authentication with different radius servers for each SSID

1. Create CAPsMAN security configuration

2. Configure AAA settings

3. Configure Radius server clients

4. Assign the configuration to your master profile (or directly to CAP itself)

```
/caps-man security add authentication-types=wpa2-eap eap-methods=passthrough encryption=aes-ccm group-
encryption=aes-ccm name=radius
/caps-man aaa set called-format=ssid
/radius add address=x.x.x.x secret=SecretUserPass service=wireless called-id=SSID1
/radius add address=y.y.y.y secret=SecretUserPass service=wireless called-id=SSID2
/caps-man configuration set security=radius
```

Now everyone connecting to CAP's with ssid=**SSID1** will have their radius authentication requests sent to **x.x.x.x** and everyone connecting to CAP's with ssid=**SSID2** will have their radius authentication requests sent to **y.y.y.y**

CAPsMAN Access-list

Access list on CAPsMAN is an ordered list of rules that is used to allow/deny clients to connect to any CAP under CAPsMAN control. When a client attempts to connect to a CAP that is controlled by CAPsMAN, CAP forwards that request to CAPsMAN. As a part of the registration process, CAPsMAN consults an access list to determine if a client should be allowed to connect. The default behavior of the access list is to allow a connection.

Access list rules are processed one by one until a matching rule is found. Then the action in the matching rule is executed. If action specifies that the client should be accepted, the client is accepted, potentially overriding its default connection parameters with ones specified in access-list rule.

An access list is configured in the **/caps-man access-list** menu. There are the following parameters for access-list rules:

- client matching parameters:
  - address - MAC address of the client
  - mask - MAC address mask to apply when comparing client address
  - interface - optional interface to compare with an interface to which client actually connects to
  - time - a time of day and days when rule matches
  - signal-range - range in which client signal must fit for a rule to match
  - allow-signal-out-of-range - an option that permits the client's signal to be out of the range always or for some time interval
- action parameter - specifies an action to take when client matches:
  - accept - accept client
  - reject - reject client
  - query-radius - query RADIUS server if a particular client is allowed to connect
- connection parameters:
  - ap-tx-limit - tx speed limit in direction to client
  - client-tx-limit - tx speed limit in direction to AP (applies to RouterOS clients only)
  - client-to-client-forwarding - specifies whether to allow forwarding data received from this client to other clients connected to the same interface
  - private-passphrase - PSK passphrase to use for this client if some PSK authentication algorithm is used
  - radius-accounting - specifies if RADIUS traffic accounting should be used if RADIUS authentication gets done for this client
  - vlan-mode - VLAN tagging mode specifies if traffic coming from a client should get tagged (and untagged when going to a client).
  - vlan-id - VLAN ID to use if doing VLAN tagging.

CAPsMAN channel

Channel group settings allow for the configuration of lists of radio channel related settings, such as radio band, frequency, Tx Power extension channel, and width.

Channel group settings are configured in the Channels profile menu **/caps-man channels**

Property	Description
band (*2ghz-b* | *2ghz-b/g* | *2ghz-b/g/n* | *2ghz-onlyg* | *2ghz-onlyn* | *5ghz-a* | *5ghz-a/n* | *5ghz-onlyn*; Default:)	Define operational radio frequency band and mode taken from hardware capability of wireless card
comment (*string*; Default:)	Short description of the Channel Group profile
extension-channel (*Ce* | *Ceee* | *eC* | *eCee* | *eeCe* | *eeeC* | *disabled*; Default:)	Extension channel configuration. (E.g. Ce = extension channel is above Control channel, eC = extension channel is below Control channel)
frequency (*integer [0..4294967295]*; Default:)	Channel frequency value in MHz on which AP will operate.
name (*string*; Default:)	A descriptive name for the Channel Group Profile
tx-power (*integer [-30..40]*; Default:)	TX Power for CAP interface (for the whole interface not for individual chains) in dBm. It is not possible to set higher than allowed by country regulations or interface. By default max allowed by country or interface is used.
width (; Default:)	Sets Channel Width in MHz. (E.g. 20, 40)

save-selected (; Default: **yes**)	Saves selected channel for the CAP Radio - will select this channel after the CAP reconnects to CAPsMAN and use it till the channel Re-optimize is done for this CAP.

CAPsMAN configuration

Configuration profiles permit pre-defined 'top-level' master settings to be applied to CAP radios being provisioned.

Configuration Profiles are configured in **/caps-man configuration** menu:

Property	Description
channel (*list*; Default:)	User defined list taken from Channel names (**/caps-man channels**)
channel.band (*2ghz-b | 2ghz-b/g | 2ghz-b/g/n | 2ghz-onlyg | 2ghz-onlyn | 5ghz-a | 5ghz-a/n | 5ghz-onlyn | 5ghz-a/n/ac | 5ghz-only-ac*; Default:)	Defines set of used channels.
channel.control-channel-width (*40mhz-turbo | 20mhz | 10mhz | 5mhz*; Default:)	Defines set of used channel widths.
channel.extension-channel (*Ce | Ceee | eC | eCee | eeCe | eeeC | xx | xxxx | disabled*; Default:)	Extension channel configuration. (E.g. Ce = extension channel is above Control channel, eC = extension channel is below Control channel)
channel.frequency (*integer [0..4294967295]*; Default:)	Channel frequency value in MHz on which AP will operate. If left blank, CAPsMAN will automatically determine the best frequency that is least occupied.
channel.reselect-interval (*time [00:00:00]; [00:00:00..00:00:00]*; Default:)	The interval after which the least occupied frequency is chosen, can be defined as a random interval, ex. as "30m..60m". Works only if **channel.frequency** is left blank.
channel.save-selected (*yes | no*; Default: **no**)	If channel frequency is chosen automatically and **channel.reselect-interval** is used, then saves the last picked frequency.
channel.secondary-frequency (*integer [0..4294967295]*; Default: **auto**)	Specifies the second frequency that will be used for 80+80MHz configuration. Set it to **Disabled** in order to disable 80+80MHz capability.
channel.skip-dfs-channels (*yes | no*; Default: **no**)	If **channel.frequency** is left blank, the selection will skip DFS channels
channel.tx-power (*integer [-30..40]*; Default:)	TX Power for CAP interface (for the whole interface not for individual chains) in dBm. It is not possible to set higher than allowed by country regulations or interface. By default max allowed by country or interface is used.
channel.width (; Default:)	Sets Channel Width in MHz.
comment (*string*; Default:)	Short description of the Configuration profile
country (*name of the country | no_country_set*; Default: **no_country_set**)	Limits available bands, frequencies and maximum transmit power for each frequency. Also specifies default value of **scan-list**. Value *no_country_set* is an FCC compliant set of channels.
datapath (*list*; Default:)	User defined list taken from Datapath names (**/caps-man datapath**)
datapath.bridge (*list*; Default:)	Bridge to which particular interface should be automatically added as port. Required only when local-forwarding is not used.
datapath.bridge-cost (*integer [1..200000000]*; Default:)	bridge port cost to use when adding as bridge port
datapath.bridge-horizon (*integer [0..4294967295]*; Default:)	bridge horizon to use when adding as bridge port

datapath.client-to-client-forwarding (*yes* / *no*; Default: **no**)	controls if client-to-client forwarding between wireless clients connected to interface should be allowed, in local forwarding mode this function is performed by CAP, otherwise it is performed by CAPsMAN
datapath.interface-list (; Default:)	
datapath.l2mtu (; Default:)	set Layer2 MTU size
datapath.local-forwarding (*yes* / *no*; Default: **no**)	Controls forwarding mode. If disabled, all L2 and L3 data will be forwarded to CAPsMAN, and further forwarding decisions will be made only then. **Note**, if disabled, make sure that each CAP interface MAC Address that participates in the same broadcast domain is unique (including local MAC's, like Bridge-MAC).
datapath.mtu (; Default:)	set MTU size
datapath.openflow-switch (; Default:)	OpenFlow switch port (when enabled) to add interface to
datapath.vlan-id (*integer [1..4095]*; Default:)	VLAN ID to assign to interface if vlan-mode enables use of VLAN tagging
datapath.vlan-mode (*use-service-tag* / *use-tag*; Default:)	Enables and specifies the type of VLAN tag to be assigned to the interface (causes all received data to get tagged with VLAN tag and allows the interface to only send out data tagged with given tag)
disconnect-timeout (; Default:)	
distance (; Default:)	
frame-lifetime (; Default:)	
guard-interval (*any* / *long*; Default: **any**)	Whether to allow the use of short guard interval (refer to 802.11n MCS specification to see how this may affect throughput). "any" will use either short or long, depending on data rate, "long" will use long only.
hide-ssid (*yes* / *no*; Default:)	• *yes* - AP does not include SSID in the beacon frames and does not reply to probe requests that have broadcast SSID. • *no* - AP includes SSID in the beacon frames and replies to probe requests that have broadcast SSID. This property has effect only in AP mode. Setting it to *yes* can remove this network from the list of wireless networks that are shown by some client software. Changing this setting does not improve the security of the wireless network, because SSID is included in other frames sent by the AP.
hw-protection-mode (; Default:)	
hw-retries (; Default:)	
installation (*any* / *indoor* / *outdoor*; Default: **any**)	
keepalive-frames (*enabled* / *disabled*; Default: **enabled**)	
load-balancing-group (*string*; Default:)	Tags the interface to the load balancing group. For a client to connect to interface in this group, the interface should have the same number of already connected clients as all other interfaces in the group or smaller. Useful in setups where ranges of CAPs mostly overlap.
max-sta-count (*integer [1..2007]*; Default:)	Maximum number of associated clients.
mode (; Default: **ap**)	Set operational mode. Only ap currently supported.

multicast-helper (*default* | *disabled* | *full*; Default: **default**)	When set to full multicast packets will be sent with unicast destination MAC address, resolving multicast problem on a wireless link. This option should be enabled only on the access point, clients should be configured in **station-bridge** mode. Available starting from v5.15. • disabled - disables the helper and sends multicast packets with multicast destination MAC addresses • full - all multicast packet mac address are changed to unicast mac addresses prior sending them out • default - default choice that currently is set to *disabled*. Value can be changed in future releases.
name (*string*; Default:)	Descriptive name for the Configuration Profile
rates (; Default:)	User defined list taken from Rates names (**/caps-man rates**)
rates.basic (*1Mbps* | *2Mbps* | *5.5Mbps* | *6Mbps* | *11Mbps* | *11Mbps* | *12Mbps* | *18Mbps* | *24Mbps* | *36Mbps* | *48Mbps* | *54Mbps*; Default:)	
rates.supported (*1Mbps* | *2Mbps* | *5.5Mbps* | *6Mbps* | *11Mbps* | *11Mbps* | *12Mbps* | *18Mbps* | *24Mbps* | *36Mbps* | *48Mbps* | *54Mbps*; Default:)	
rates.ht-basic-mcs (*list of (mcs-0* | *mcs-1* | *mcs-2* | *mcs-3* | *mcs-4* | *mcs-5* | *mcs-6* | *mcs-7* | *mcs-8* | *mcs-9* | *mcs-10* | *mcs-11* | *mcs-12* | *mcs-13* | *mcs-14* | *mcs-15* | *mcs-16* | *mcs-17* | *mcs-18* | *mcs-19* | *mcs-20* | *mcs-21* | *mcs-22* | *mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11n for MCS specification.
rates.ht-supported-mcs (*list of (mcs-0* | *mcs-1* | *mcs-2* | *mcs-3* | *mcs-4* | *mcs-5* | *mcs-6* | *mcs-7* | *mcs-8* | *mcs-9* | *mcs-10* | *mcs-11* | *mcs-12* | *mcs-13* | *mcs-14* | *mcs-15* | *mcs-16* | *mcs-17* | *mcs-18* | *mcs-19* | *mcs-20* | *mcs-21* | *mcs-22* | *mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7; mcs-8; mcs-9; mcs-10; mcs-11; mcs-12; mcs-13; mcs-14; mcs-15; mcs-16; mcs-17; mcs-18; mcs-19; mcs-20; mcs-21; mcs-22; mcs-23**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11n for MCS specification.
rates.vht-basic-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **none**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - client must support MCS-0 to MCS-7 • *MCS 0-8* - client must support MCS-0 to MCS-8 • *MCS 0-9* - client must support MCS-0 to MCS-9
rates.vht-supported-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **none**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - devices will advertise as supported MCS-0 to MCS-7 • *MCS 0-8* - devices will advertise as supported MCS-0 to MCS-8 • *MCS 0-9* - devices will advertise as supported MCS-0 to MCS-9
rx-chains (*list of integer [0..3]*; Default: **0**)	Which antennas to use for receive.
security (*string*; Default: **none**)	Name of security configuration from **/caps-man security**
security.authentication-types (*list of string*; Default: **none**)	Specify the type of Authentication from **wpa-psk**, **wpa2-psk**, **wpa-eap** or **wpa2-eap**
security.disable-pmkid (; Default:)	

security.eap-methods (*eap-tls* | *passthrough*; Default: **none**)	• eap-tls - Use built-in EAP TLS authentication. • passthrough - Access point will relay authentication process to the RADIUS server.
security.eap-radius-accounting (; Default:)	specifies if RADIUS traffic accounting should be used if RADIUS authentication gets done for this client
security.encryption (*aes-ccm* | *tkip*; Default:)	Set type of unicast encryption algorithm used
security.group-encryption (*aes-ccm* | *tkip*; Default: **aes-ccm**)	Access Point advertises one of these ciphers, multiple values can be selected. Access Point uses it to encrypt all broadcast and multicast frames. Client attempts connection only to Access Points that use one of the specified group ciphers. • tkip - Temporal Key Integrity Protocol - encryption protocol, compatible with legacy WEP equipment, but enhanced to correct some of the WEP flaws. • aes-ccm - more secure WPA encryption protocol, based on the reliable AES (Advanced Encryption Standard). Networks free of WEP legacy should use only this cipher.
security.group-key-update (*time: 30s..1h*; Default: **5m**)	Controls how often Access Point updates the group key. This key is used to encrypt all broadcast and multicast frames. property only has effect for Access Points.
security.passphrase (*string*; Default:)	WPA or WPA2 pre-shared key
security.tls-certificate (*none* | *name*; Default:)	Access Point always needs a certificate when **security.tls-mode** is set to value other than **no-certificates**.
security.tls-mode (*verify-certificate* | *dont-verify-certificate* | *no-certificates* | *verify-certificate-with-crl*; Default:)	This property has effect only when **security.eap-methods** contains *eap-tls*. • verify-certificate - Require remote device to have valid certificate. Check that it is signed by known certificate authority. No additional identity verification is done. Certificate may include information about time period during which it is valid. If router has incorrect time and date, it may reject valid certificate because router's clock is outside that period. See also the Certificates configuration. • dont-verify-certificate - Do not check certificate of the remote device. Access Point will not require client to provide certificate. • no-certificates - Do not use certificates. TLS session is established using 2048 bit anonymous Diffie-Hellman key exchange. • verify-certificate-with-crl - Same as verify-certificate but also checks if the certificate is valid by checking the Certificate Revocation List.
ssid (*string (0..32 chars)*; Default:)	SSID (service set identifier) is a name broadcast in the beacons that identifies wireless network.
tx-chains (*list of integer [0..3]*; Default: **0**)	Which antennas to use for transmit.

CAPsMAN datapath

Datapath settings control data forwarding related aspects. On CAPsMAN datapath settings are configured in the datapath profile menu **/caps-man datapath** or directly in a configuration profile or interface menu as settings with **datapath.** prefix.

There are 2 major forwarding modes:

- local forwarding mode, where CAP is locally forwarding data to and from wireless interface

- manager forwarding mode, where CAP sends to CAPsMAN all data received over wireless and only sends out the wireless data received from CAPsMAN. In this mode, even client-to-client forwarding is controlled and performed by CAPsMAN.

Forwarding mode is configured on a per-interface basis - so if one CAP provides 2 radio interfaces, one can be configured to operate in local forwarding mode and the other in manager forwarding mode. The same applies to Virtual-AP interfaces - each can have different forwarding mode from master interface or other Virtual-AP interfaces.

Most of the datapath settings are used only when in manager forwarding mode, because in local forwarding mode CAPsMAN does not have control over data forwarding.

There are the following datapath settings:

- bridge -- bridge interface to add interface to, as a bridge port, when enabled
- bridge-cost -- bridge port cost to use when adding as bridge port
- bridge-horizon -- bridge horizon to use when adding as bridge port
- client-to-client-forwarding -- controls if client-to-client forwarding between wireless clients connected to interface should be allowed, in local forwarding mode this function is performed by CAP, otherwise it is performed by CAPsMAN.
- local-forwarding -- controls forwarding mode
- openflow-switch -- OpenFlow switch to add interface to, as port when enabled
- vlan-id -- VLAN ID to assign to interface if vlan-mode enables use of VLAN tagging
- vlan-mode -- VLAN tagging mode specifies if VLAN tag should be assigned to interface (causes all received data to get tagged with VLAN tag and allows interface to only send out data tagged with given tag)

CAPsMAN interface

CAPsMAN interfaces are managed in **/caps-man interface** menu:

```
[admin@CM] > /caps-man interface print
Flags: M - master, D - dynamic, B - bound, X - disabled, I - inactive, R - running
# NAME RADIO-MAC MASTER-INTERFACE
0 M BR cap2 00:0C:42:1B:4E:F5 none
1 B cap3 00:00:00:00:00:00 cap2
```

CAPsMAN manager

Property	Description
enabled (*yes* | *no*; Default: **no**)	Disable or enable CAPsMAN functionality
certificate (*auto* | *certificate name* | *none*; Default: **none**)	Device certificate
ca-certificate (*auto* | *certificate name* | *none*; Default: **none**)	Device CA certificate
require-peer-certificate (*yes* | *no*; Default: **no**)	Require all connecting CAPs to have a valid certificate
package-path (*string* |; Default:)	Folder location for the RouterOS packages. For example, use "/upgrade" to specify the upgrade folder from the files section. If empty string is set, CAPsMAN can use built-in RouterOS packages, note that in this case only CAPs with the same architecture as CAPsMAN will be upgraded.
upgrade-policy (*none* | *require-same-version* | *suggest-same-upgrade*; Default: **none**)	Upgrade policy options • none - do not perform upgrade • require-same-version - CAPsMAN suggest to upgrade the CAP RouterOS version and if it fails it will not provision the CAP. (Manual provision is still possible) • suggest-same-version - CAPsMAN suggests to upgrade the CAP RouterOS version and if it fails it will still be provisioned

CAPsMAN provisioning

CAPsMAN distinguishes between CAPs based on a common-name identifier. The identifier is generated based on the following rules:

- if CAP provided a certificate, the identifier is set to the Common Name field in the certificate
- otherwise, an identifier is based on Base-MAC provided by CAP in the form: '[XX:XX:XX:XX:XX:XX]'.

When the DTLS connection with CAP is successfully established (which means that CAP identifier is known and valid), CAPsMAN makes sure there is no stale connection with CAP using the same identifier. Currently connected CAPs are listed in **/caps-man remote-cap** menu:

```
[admin@CM] /caps-man> remote-cap print
# ADDRESS IDENT STATE RADIOS 0 00:0C:42:00:C0:32/27044 MT-000C4200C032 Run 1
```

CAPsMAN distinguishes between actual wireless interfaces (radios) based on their built-in MAC address (radio-mac). This implies that it is impossible to manage two radios with the same MAC address on one CAPsMAN. Radios currently managed by CAPsMAN (provided by connected CAPs) are listed in **/caps-man radio** menu:

```
[admin@CM] /caps-man> radio print
Flags: L - local, P - provisioned
# RADIO-MAC INTERFACE REMOTE-AP-IDENT
0 P 00:03:7F:48:CC:07 cap1 MT-000C4200C032
```

When CAP connects, CAPsMAN at first tries to bind each CAP radio to CAPsMAN master interface based on radio-mac. If an appropriate interface is found, radio gets set up using master interface configuration and configuration of slave interfaces that refer to a particular master interface. At this moment interfaces (both master and slaves) are considered bound to radio and radio is considered provisioned.

If no matching master interface for radio is found, CAPsMAN executes 'provisioning rules'. Provisioning rules is an ordered list of rules that contain settings that specify which radio to match and settings that specify what action to take if a radio matches.

Provisioning rules for matching radios are configured in **/caps-man provisioning** menu:

Property	Description
action (*create-disabled* | *create-enabled* | *create-dynamic-enabled* | *none*; Default: **none**)	Action to take if rule matches are specified by the following settings: • **create-disabled** - create disabled static interfaces for radio. I.e., the interfaces will be bound to the radio, but the radio will not be operational until the interface is manually enabled; • **create-enabled** - create enabled static interfaces. I.e., the interfaces will be bound to the radio and the radio will be operational; • **create-dynamic-enabled** - create enabled dynamic interfaces. I.e., the interfaces will be bound to the radio, and the radio will be operational; • **none** - do nothing, leaves radio in the non-provisioned state;
comment (*string*; Default:)	Short description of the Provisioning rule
common-name-regexp (*string*; Default:)	Regular expression to match radios by common name. Each CAP's common name identifier can be found under "/caps-man radio" as value "REMOTE-CAP-NAME"
hw-supported-modes (*a|a-turbo|ac|an|b|g|g-turbo|gn*; Default:)	Match radios by supported wireless modes
identity-regexp (*string*; Default:)	Regular expression to match radios by router identity
ip-address-ranges (*IpAddressRange[, IpAddressRanges] max 100x*; Default: **""**)	Match CAPs with IPs within configured address range.
master-configuration (*string*; Default:)	If **action** specifies to create interfaces, then a new master interface with its configuration set to this configuration profile will be created

name-format (*cap* | *identity* | *prefix* | *prefix-identity*; Default: **cap**)	specify the syntax of the CAP interface name creation • cap - default name • identity - CAP boards system identity name • prefix - name from the name-prefix value • prefix-identity - name from the name-prefix value and the CAP boards system identity name
name-prefix (*string*; Default:)	name prefix which can be used in the name-format for creating the CAP interface names
radio-mac (*MAC address*; Default: **00:00:00:00:00:00**)	MAC address of radio to be matched, empty MAC (00:00:00:00:00:00) means match all MAC addresses
slave-configurations (*string*; Default:)	If **action** specifies to create interfaces, then a new slave interface for each configuration profile in this list is created.

⚠ If no rule matches radio, then implicit default rule with action **create-enabled** and no configurations set is executed.

To get the active provisioning matchers:

```
[admin@CM] /caps-man provisioning> print
Flags: X - disabled
0 radio-mac=00:00:00:00:00:00 action=create-enabled master-configuration=main-cfg
slave-configurations=virtual-ap-cfg name-prefix=""
```

For the user's convenience there are commands that allow the re-execution of the provisioning process for some radio or all radios provided by some AP:

```
[admin@CM] > caps-man radio provision 0
```

and

```
[admin@CM] > caps-man remote-cap provision 0
```

CAPsMAN radio

see /caps-man provisioning

CAPsMAN rates

see /caps-man configuration

CAPsMAN registration-table

Registration table contains a list of clients that are connected to radios controlled by CAPsMAN and is available in **/caps-man registration-table** menu:

```
[admin@CM] /caps-man> registration-table print
# INTERFACE MAC-ADDRESS UPTIME RX-SIGNAL
0 cap1 00:03:7F:48:CC:0B 1h38m9s210ms -36
```

CAPsMAN remote-cap

see /caps-man provisioning

CAPsMAN security

Example

Assuming that rest of the settings are already configured and only the "Security" part has been left.

Radius authentication with one server

1. Create CAPsMAN security configuration

2. Configure Radius server client

3. Assign the configuration to your master profile (or directly to CAP itself)

```
/caps-man security add authentication-types=wpa2-eap eap-methods=passthrough encryption=aes-ccm group-
encryption=aes-ccm name=radius
/radius add address=x.x.x.x secret=SecretUserPass service=wireless
/caps-man configuration set security=radius
```

Radius authentication with different radius servers for each SSID

1. Create CAPsMAN security configuration

2. Configure AAA settings

3. Configure Radius server clients

4. Assign the configuration to your master profile (or directly to CAP itself)

```
/caps-man security add authentication-types=wpa2-eap eap-methods=passthrough encryption=aes-ccm group-
encryption=aes-ccm name=radius
/caps-man aaa set called-format=ssid
/radius add address=x.x.x.x secret=SecretUserPass service=wireless called-id=SSID1
/radius add address=y.y.y.y secret=SecretUserPass service=wireless called-id=SSID2
/caps-man configuration set security=radius
```

Now everyone connecting to CAP's with ssid=**SSID1** will have their radius authentication requests sent to **x.x.x.x** and everyone connecting to CAP's with ssid=**SSID2** will have their radius authentication requests sent to **y.y.y.y**

AP Controller (CAPsMAN)

Overview

CAPsMAN allows applying wireless settings to multiple MikroTik AP devices from a central configuration interface.

More specifically, the Controlled Access Point system Manager (CAPsMAN) allows centralization of wireless network management and if necessary, data processing. When using the CAPsMAN feature, the network will consist of a number of 'Controlled Access Points' (CAP) that provide wireless connectivity and a 'system Manager' (CAPsMAN) that manages the configuration of the APs, it also takes care of client authentication and optionally, data forwarding.

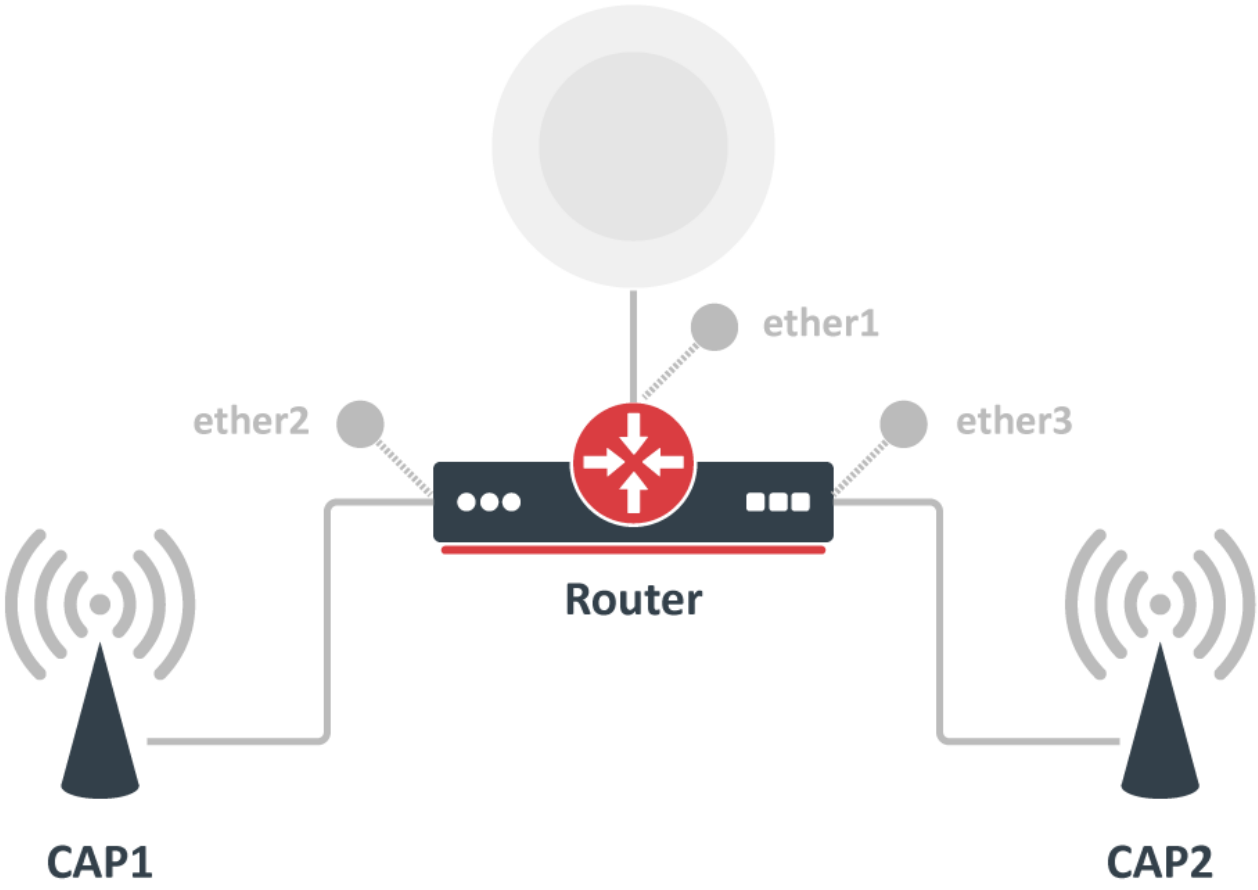
When a CAP is controlled by CAPsMAN it only requires the minimum configuration required to allow it to establish a connection with CAPsMAN. Functions that were conventionally executed by an AP (like access control, client authentication) are now executed by CAPsMAN. The CAP device now only has to provide the wireless link layer encryption/decryption.

Depending on the configuration, data is either forwarded to CAPsMAN for centralized processing (*default*) or forwarded locally at the CAP itself (local forwarding mode).

Requirements

- Any RouterOS device can be a controlled wireless access point (CAP) as long as it has at least a Level 4 RouterOS license
- CAPsMAN server can be installed on any RouterOS device, even if the device itself does not have a wireless interface
- Unlimited CAPs (access points) supported by CAPsMAN
- 32 Radios per CAP maximum
- 32 Virtual interfaces per master radio interface maximum
- Not possible to use Nv2 and NStreme proprietary protocols

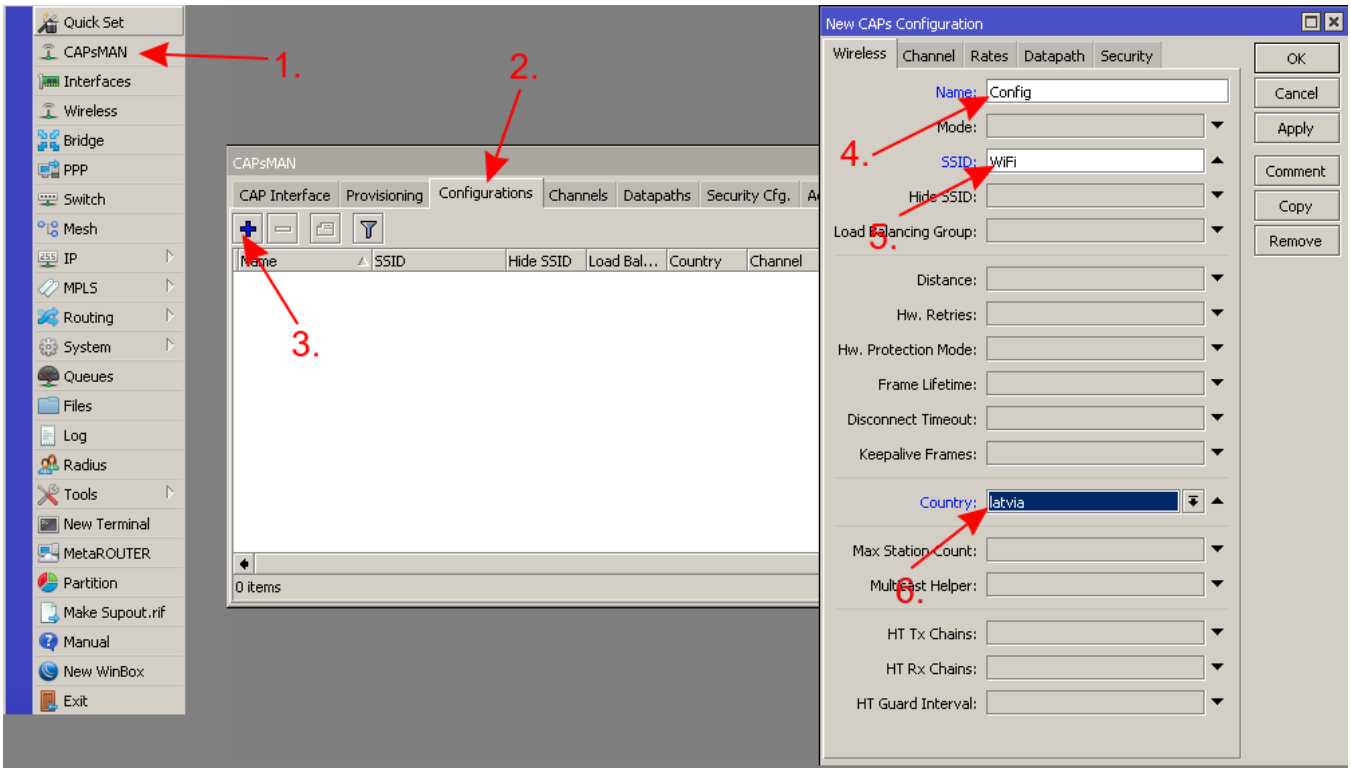
Simple setup of a CAPsMAN system



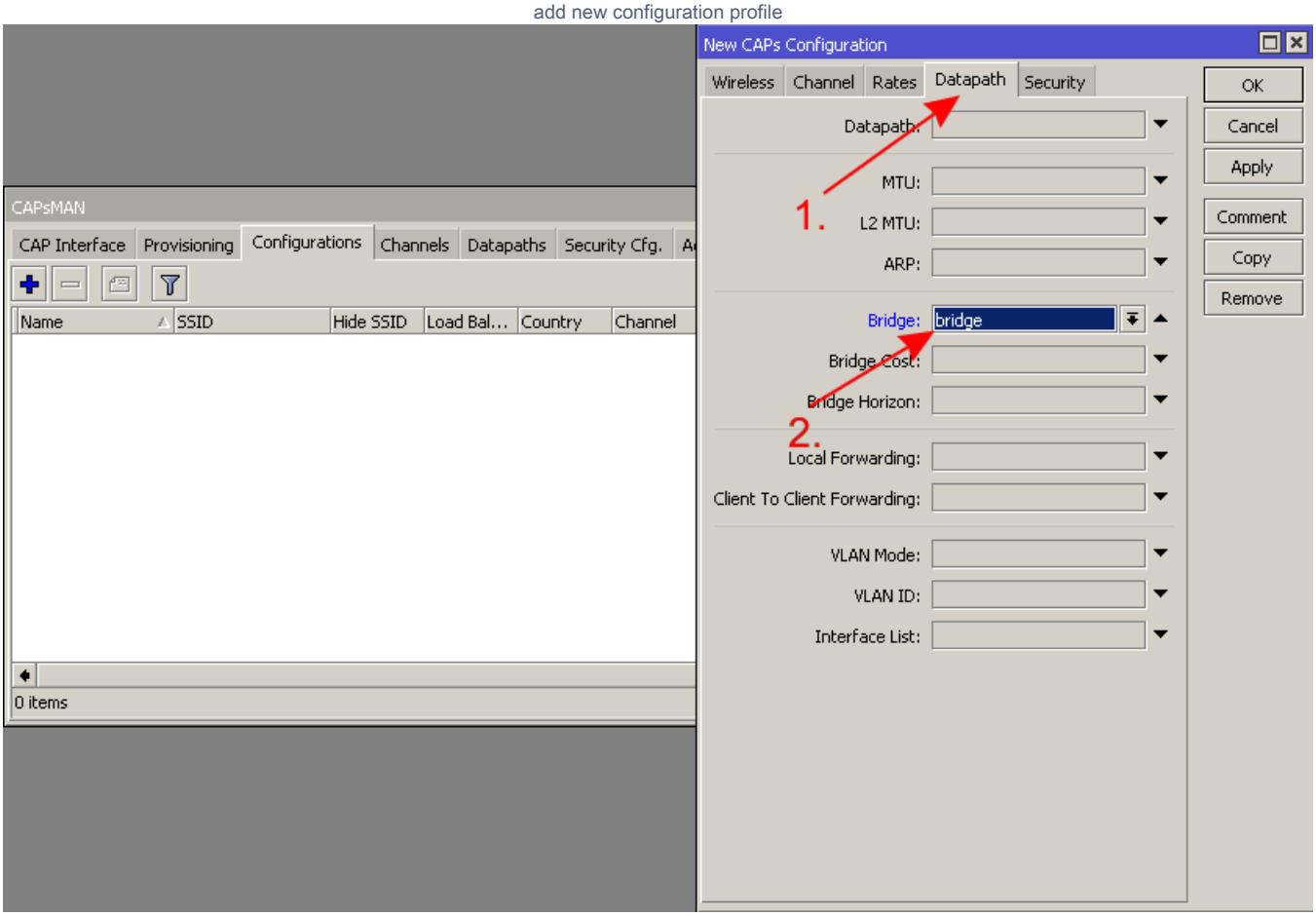
Before deep-diving into the details of CAPsMAN operation, let us quickly illustrate how to set up the most basic system where you have a MikroTik router that manages two MikroTik AP devices. The benefit of CAPsMAN is that the CAP units don't need to be configured, all settings are done in the CAPsMAN server.

The CAPsMAN setup consists of defining configuration templates, which will then be pushed to the controllable AP devices (CAPs). Assuming your main router is already connected to the internet and works fine, you can proceed as follows.

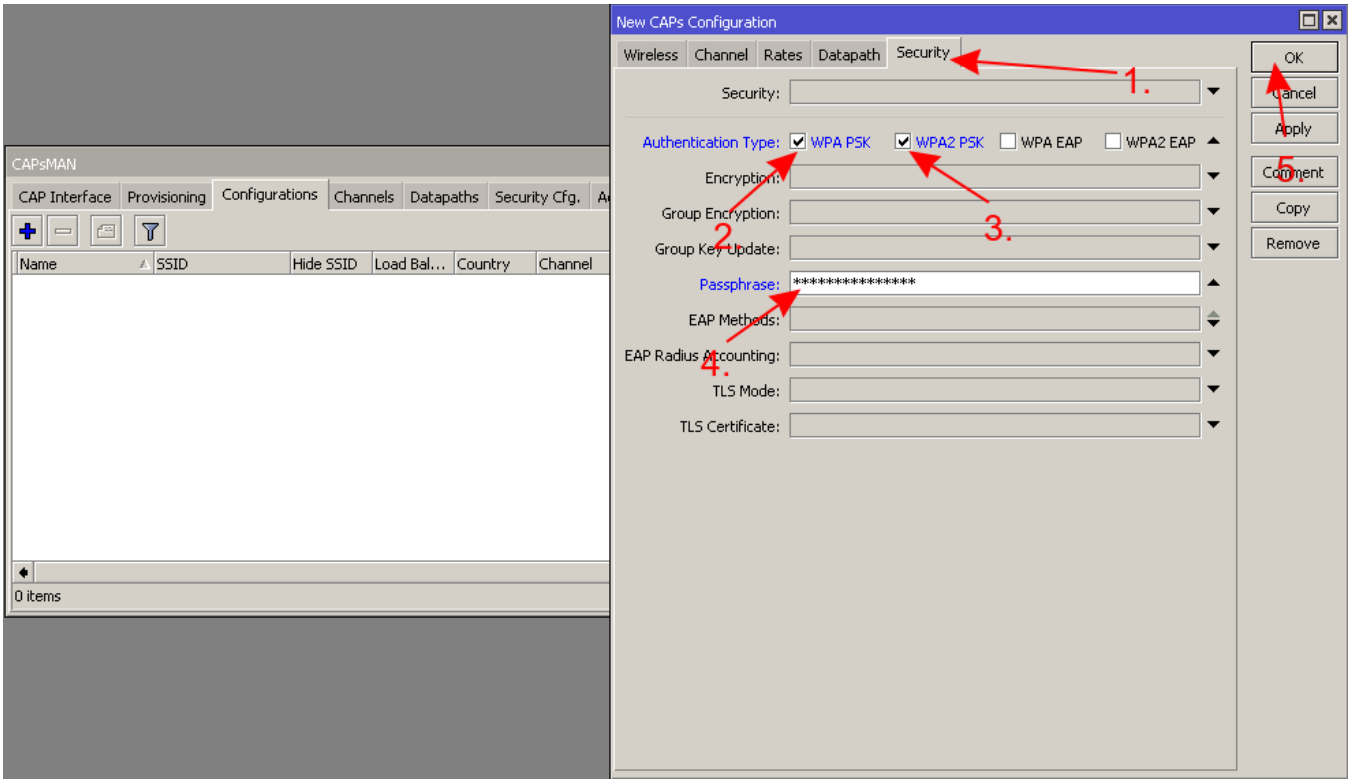
In the central device, which will be your CAPsMAN server, create a new "Configuration" template with only the basic settings (network name, country, the local LAN bridge interface, the wireless password):



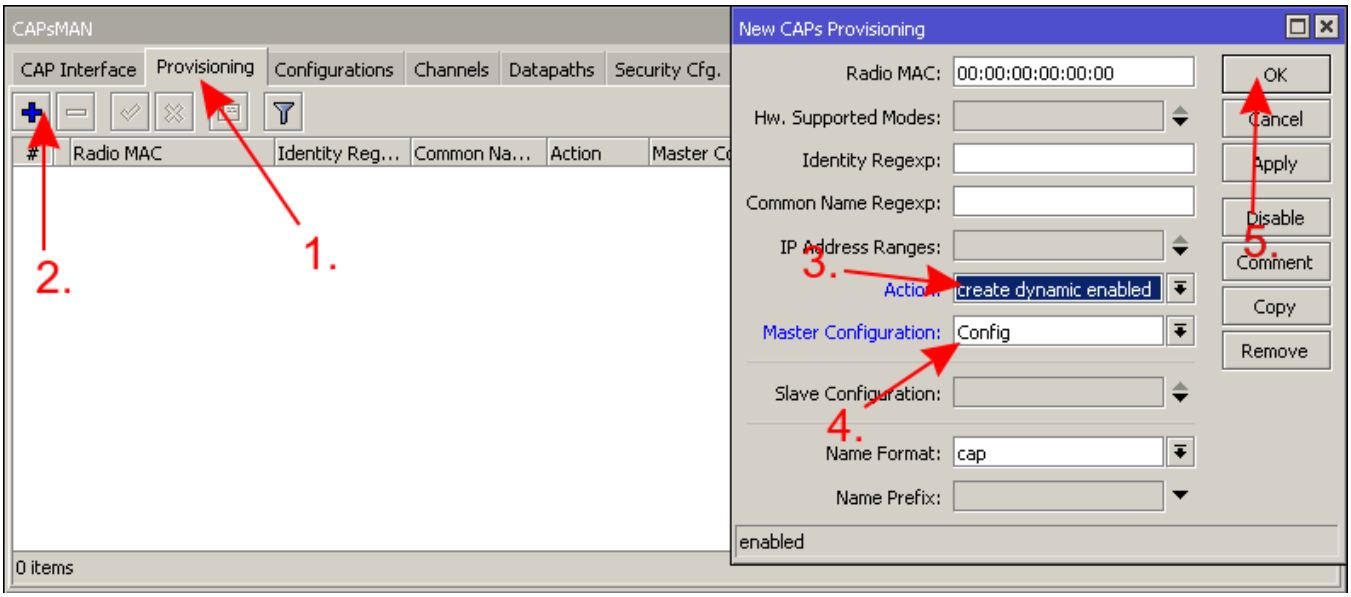
1. add new configuration profile



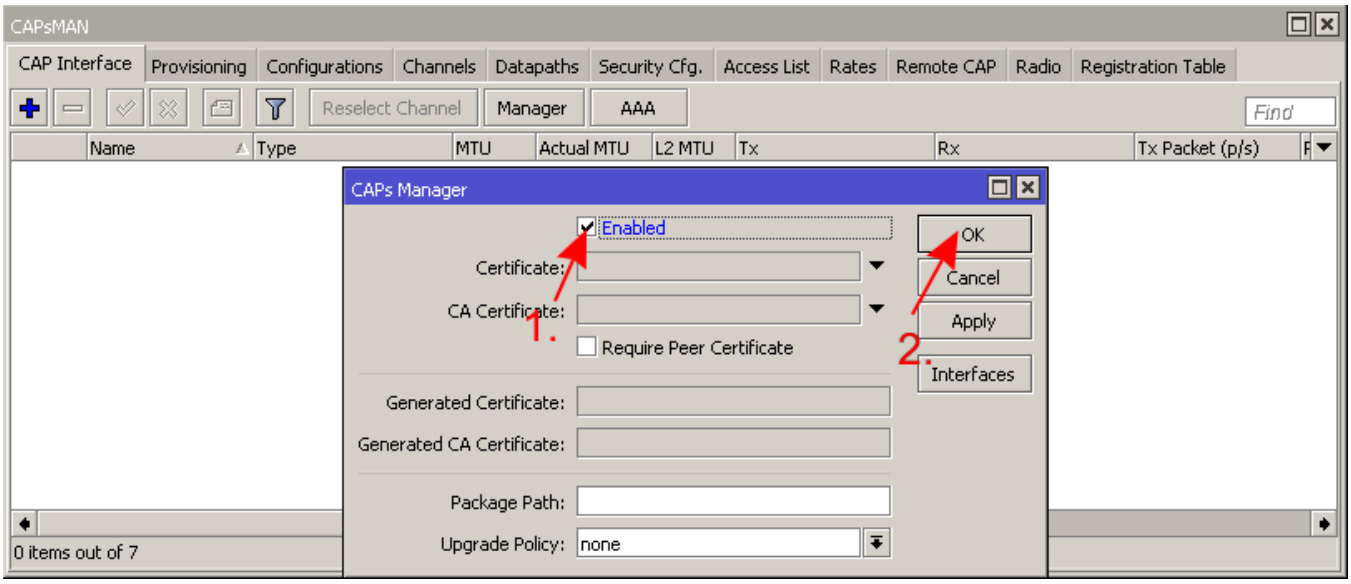
2.

3. 

Then create a new "Provisioning" rule, which will assign the created configuration template to the CAP devices:

4. 

All that remains to do on the CAPsMAN, is to enable it:

5. 

Most MikroTik AP devices already support CAP mode out of the box, all you need to do, is make sure they are on the same network as your CAPsMAN, and then boot them up, while holding the reset button.

So, for example, connect the CAP device to one of the CAPsMAN device LAN ports while it is turned off, then hold the reset button, and power on the CAP device. Keep holding the button until the User LED turns solid, release now to turn on CAP mode. The device will now look for a CAPsMAN server (total time to hold the button, around 10 seconds).

The device will now show up in the CAPsMAN "Remote CAP" menu and will be "provisioned" with the configuration template, as per the provisioning settings. For more details on how to manually adjust all settings, keep reading this document.

CAP to CAPsMAN Connection

For the CAPsMAN system to function and provide wireless connectivity, a CAP must establish management connection with CAPsMAN. A management connection can be established using MAC or IP layer protocols and is secured using 'DTLS'.

A CAP can also pass the client data connection to the Manager, but the data connection is not secured. If this is deemed necessary, then other means of data security needs to be used, e.g. IPSec or encrypted tunnels.

CAP to CAPsMAN connection can be established using 2 transport protocols (via Layer 2 and Layer3).

- MAC layer connection features:
  - no IP configuration necessary on CAP
  - CAP and CAPsMAN must be on the same Layer 2 segment - either physical or virtual (by means of L2 tunnels)
- IP layer (UDP) connection features:
  - can traverse NAT if necessary
  - CAP must be able to reach CAPsMAN using IP protocol
  - if the CAP is not on the same L2 segment as CAPsMAN, it must be provisioned with the CAPsMAN IP address, because IP multicast based discovery does not work over Layer3

In order to establish connection with CAPsMAN, CAP executes a discovery process. During discovery, CAP attempts to contact CAPsMAN and builds an available CAPsMANs list. CAP attempts to contact to an available CAPsMAN using:

- configured list of Manager IP addresses
- list of CAPsMAN IP addresses obtained from DHCP server
- broadcasting on configured interfaces using both - IP and MAC layer protocols.

When the list of available CAPsMANs is built, CAP selects a CAPsMAN based on the following rules:

- if **caps-man-names** parameter specifies allowed manager names (**/system identity** of CAPsMAN), CAP will prefer the CAPsMAN that is earlier in the list, if list is empty it will connect to any available Manager
- suitable Manager with MAC layer connectivity is preferred to Manager with IP connectivity

After Manager is selected, CAP attempts to establish DTLS connection. There are the following authentication modes possible:

- no certificates on CAP and CAPsMAN - no authentication

- only Manager is configured with certificate - CAP checks CAPsMAN certificate, but does not fail if it does not have appropriate trusted CA certificate, CAPsMAN must be configured with **require-peer-certificate=no** in order to establish connection with CAP that does not possess certificate
- CAP and CAPsMAN are configured with certificates - mutual authentication

After DTLS connection is established, CAP can optionally check CommonName field of certificate provided by CAPsMAN. **caps-man-certificate-common-names** parameter contains list of allowed CommonName values. If this list is not empty, CAPsMAN must be configured with certificate. If this list is empty, CAP does not check CommonName field.

If the CAPsMAN or CAP gets disconnected from the network, the loss of connection between CAP and CAPsMAN will be detected in approximately 10-20 seconds.

CAP Auto Locking to CAPsMAN

CAP can be configured to automatically lock to a particular CAPsMAN server. Locking is implemented by recording certificate CommonName of CAPsMAN that CAP is locked to and checking this CommonName for all subsequent connections. As this feature is implemented using certificate CommonName, use of certificates is mandatory for locking to work.

Locking is enabled by the following command:

```
[admin@CAP] > /interface wireless cap set lock-to-caps-man=yes
```

Once CAP connects to suitable CAPsMAN and locks to it, it is reflected like this:

```
[admin@wtp] > /interface wireless cap print
...
        locked-caps-man-common-name: CAPsMAN-000C424C30F3
```

From now on CAP will only connect to CAPsMAN with this CommonName, until locking requirement is cleared, by setting **lock-to-caps-man=no**. This approach needs to be used if it is necessary to force CAP to lock to another CAPsMAN - by at first setting **lock-to-caps-man=no** followed by **lock-to-caps-man=yes**.

Note that CAP can be manually "locked" to CAPsMAN by setting **caps-man-certificate-common-names**.

Auto Certificates

To simplify CAPsMAN and CAP configuration when certificates are required (e.g. for automatic locking feature), CAPsMAN can be configured to generate necessary certificates automatically and CAP can be configured to request certificate from CAPsMAN.

Automatic certificates do not provide full public key infrastructure and are provided for simple setups. If more complicated PKI is necessary - supporting proper certificate validity periods, multiple-level CA certificates, certificate renewal - other means must be used, such as manual certificate distribution or SCEP.

CAPsMAN has the following certificate settings:

- **certificate** - this is CAPsMAN certificate, private key must be available for this certificate. If set to **none**, CAPsMAN will operate in no-certificate mode and none of certificate requiring features will work. If set to **auto**, CAPsMAN will attempt to issue certificate to itself using CA certificate (see **ca-certificate** description). Note that CommonName automatically issued certificate will be "CAPsMAN-<mac address>" and validity period for will be the same as for CA certificate.
- **ca-certificate** - this is CA certificate that CAPsMAN will use when issuing certificate for itself if necessary (see **certificate** description) and when signing certificate requests from CAPs. If set to **none**, CAPsMAN will not be able to issue certificate to itself or sign certificate requests from CAPs. If set to **auto**, CAPsMAN will generate self-signed CA certificate to use as CA certificate. CommonName for this certificate will take form "CAPsMAN-CA-<mac address>" and validity period will be from jan/01/1970 until jan/18/2038.

When CAPsMAN will auto-generate certificates, this will be reflected like this:

```
[admin@CM] /caps-man manager> pr
                  enabled: yes
              certificate: auto
           ca-certificate: auto
  require-peer-certificate: no
     generated-certificate: CAPsMAN-000C424C30F3
  generated-ca-certificate: CAPsMAN-CA-000C424C30F3
```

And certificates:

```
[admin@CM] /certificate> print detail
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key,
A - authority, I - issued, R - revoked, E - expired, T - trusted
 0 K    A T name="CAPsMAN-CA-000C424C30F3" common-name="CAPsMAN-CA-000C424C30F3" key-size=2048
           days-valid=24854 trusted=yes
           key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign
           serial-number="1" fingerprint="69d77bbb45c50afd2d6c1785c2a3d72596b8a5f6"
           invalid-before=jan/01/1970 00:00:01 invalid-after=jan/18/2038 03:14:07

 1 K    I   name="CAPsMAN-000C424C30F3" common-name="CAPsMAN-000C424C30F3" key-size=2048
           days-valid=24854 trusted=no key-usage=digital-signature,key-encipherment
           ca=CAPsMAN-CA-000C424C30F3 serial-number="1"
           fingerprint="e853ddb9d41fc139083a176ab164331bc24bc5ed"
           invalid-before=jan/01/1970 00:00:01 invalid-after=jan/18/2038 03:14:07
```

CAP can be configured to request certificate from CAPsMAN. In order for this to work, CAP must be configured with setting **certificate=request** and CAPsMAN must have CA certificate available (either specified in **ca-certificate** setting or auto-generated).

CAP will initially generate private key and certificate request with CommonName of form "CAP-<mac address>". When CAP will establish connection with CAPsMAN, CAP will request CAPsMAN to sign its certificate request. If this will succeed, CAPsMAN will send CA certificate and newly issued certificate to CAP. CAP will import these certificates in its certificate store:

```
[admin@CAP] > /interface wireless cap print
...
             requested-certificate: cert_2
       locked-caps-man-common-name: CAPsMAN-000C424C30F3
[admin@CAP] > /certificate print detail
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key,
A - authority, I - issued, R - revoked, E - expired, T - trusted
 0       T name="cert_1" issuer=CN=CAPsMAN-CA-000C424C30F3 common-name="CAPsMAN-CA-000C424C30F3"
           key-size=2048 days-valid=24837 trusted=yes
           key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign
           serial-number="1" fingerprint="69d77bbb45c50afd2d6c1785c2a3d72596b8a5f6"
           invalid-before=jan/01/1970 00:00:01 invalid-after=jan/01/2038 03:14:07

 1 K     T name="cert_2" issuer=CN=CAPsMAN-CA-000C424C30F3 common-name="CAP-000C4200C032"
           key-size=2048 days-valid=24837 trusted=yes
           key-usage=digital-signature,key-encipherment serial-number="2"
           fingerprint="2c85bf2fbc9fc0832e47cd2773a6f4b6af35ef65"
           invalid-before=jan/01/1970 00:00:01 invalid-after=jan/01/2038 03:14:07
```

On subsequent connections to CAPsMAN, CAP will use generated certificate.

CAP Configuration

When an AP is configured to be controlled by CAPsMAN, configuration of the managed wireless interfaces on the AP is ignored *(exceptions: antenna-gain, antenna-mode)*. Instead, AP accepts configuration for the managed interfaces from CAPsMAN.

> ⚠ The CAP wireless interfaces that are managed by CAPsMAN and whose traffic is being forwarded to CAPsMAN (ie. they are not in *local forwarding* mode), are shown as *disabled*, with the note **Managed by CAPsMAN**. Those interfaces that are in *local forwarding* mode (traffic is locally managed by CAP, and only management is done by CAPsMAN) are not shown disabled, but the note **Managed by CAPsMAN** is shown

CAP behavior of AP is configured in **/interface wireless cap** menu. From there you can:

- Disable or enable CAP feature on the device
- Set list of wireless interfaces to be controlled by Manager
- Set list of interfaces over which CAP should attempt to discover Manager
- Set list of Manager IP addresses that CAP will attempt to contact during discovery
- Set list of Manager names that CAP will attempt to connect
- Set list of Manager certificate CommonNames that CAP will connect to
- Set bridge to which interfaces should be added when local forwarding mode is used

Each wireless interface on a CAP that is under CAPsMAN control appears as a virtual interface on the CAPsMAN. This provides maximum flexibility in data forwarding control using regular RouterOS features, such as routing, bridging, firewall, etc.CAPsMAN Configuration Concepts

Many wireless interface settings are able to be grouped together into named groups ('profiles') that simplifies the reuse of configuration - for example, common configuration settings can be configured in a 'configuration profile' and multiple interfaces can then refer to that profile. At the same time any profile setting can be overridden directly in an interface configuration for maximum flexibility.

Currently there are the following setting groups:

- channel - channel related settings, such as frequency and width
- datapath - data forwarding related settings, such as bridge to which particular interface should be automatically added as port
- security - security related settings, such as allowed authentication types or passphrase
- configuration - main wireless settings group, includes settings such as SSID, and additionally binds together other setting groups - that is, configuration profile can refer to channel, security, etc. named setting groups. Additionally any setting can be overridden directly in configuration profile.

Interface settings bind together all setting groups, but additionally any setting can be overridden directly in interface settings.

By means of setting groups, configuration is organized in hierarchical structure with interface (actual user of configuration) as the root. In order to figure out the effective value of some setting this structure is consulted in a fashion where a higher level setting value overrides a lower level value.

For example, when WPA2 passphrase to be used by a particular interface needs to be found, the following places are consulted and the first place with WPA2 passphrase configured specifies effective passphrase. "->" denotes referring to setting profile (if configured):

- interface passphrase
- interface->security passphrase
- interface->configuration passphrase
- interface->configuration->security passphrase

There are 2 types of interfaces on CAPsMAN - "master" and "slave". The master interface holds the configuration for an actual wireless interface (radio), while a slave interface links to the master interface and is intended to hold the configuration for a Virtual-AP (multiple SSID support). There are settings that are meaningful only for master interface, i.e. mainly hardware setup related settings such as radio channel settings. Note that in order for a radio to accept clients, it's master interface needs to be enabled. Slave interfaces will become operational only if enabled and the master interface is enabled.

Interfaces on CAPsMAN can be static or dynamic. Static interfaces are stored in RouterOS configuration and will persist across reboots. Dynamic interfaces exist only while a particular CAP is connected to CAPsMAN.

CAPsMAN Global Configuration

Settings to enable CAPsMAN functionality are found in **/caps-man manager** menu:

Property	Description
enabled (*yes* | *no*; Default: **no**)	Disable or enable CAPsMAN functionality
certificate (*auto* | *certificate name* | *none*; Default: **none**)	Device certificate
ca-certificate (*auto* | *certificate name* | *none*; Default: **none**)	Device CA certificate
require-peer-certificate (*yes* | *no*; Default: **no**)	Require all connecting CAPs to have a valid certificate
package-path (*string* |; Default:)	Folder location for the RouterOS packages. For example, use "/upgrade" to specify the upgrade folder from the files section. If empty string is set, CAPsMAN can use built-in RouterOS packages, note that in this case only CAPs with the same architecture as CAPsMAN will be upgraded.
upgrade-policy (*none* | *require-same-version* | *suggest-same-upgrade*; Default: **none**)	Upgrade policy options - none - do not perform upgrade - require-same-version - CAPsMAN suggest to upgrade the CAP RouterOS version and if it fails it will not provision the CAP. (Manual provision is still possible) - suggest-same-version - CAPsMAN suggests to upgrade the CAP RouterOS version and if it fails it will still be provisioned

Radio Provisioning

CAPsMAN distinguishes between CAPs based on an identifier. The identifier is generated based on the following rules:

- if CAP provided a certificate, identifier is set to the Common Name field in the certificate
- otherwise identifier is based on Base-MAC provided by CAP in the form: '[XX:XX:XX:XX:XX:XX]'.

When the DTLS connection with CAP is successfully established (which means that CAP identifier is known and valid), CAPsMAN makes sure there is no stale connection with CAP using the same identifier. Currently connected CAPs are listed in **/caps-man remote-cap** menu:

```
[admin@CM] /caps-man> remote-cap print
 # ADDRESS                                 IDENT           STATE               RADIOS
 0 00:0C:42:00:C0:32/27044                 MT-000C4200C032 Run                      1
```

CAPsMAN distinguishes between actual wireless interfaces (radios) based on their builtin MAC address (radio-mac). This implies that it is impossible to manage two radios with the same MAC address on one CAPsMAN. Radios currently managed by CAPsMAN (provided by connected CAPs) are listed in **/caps-man radio** menu:

```
[admin@CM] /caps-man> radio print
Flags: L - local, P - provisioned
 #    RADIO-MAC         INTERFACE                                 REMOTE-AP-IDENT
 0  P 00:03:7F:48:CC:07 cap1                                      MT-000C4200C032
```

When CAP connects, CAPsMAN at first tries to bind each CAP radio to CAPsMAN master interface based on radio-mac. If an appropriate interface is found, radio gets set up using master interface configuration and configuration of slave interfaces that refer to particular master interface. At this moment interfaces (both master and slaves) are considered bound to radio and radio is considered provisioned.

If no matching master interface for radio is found, CAPsMAN executes 'provisioning rules'. Provisioning rules is an ordered list of rules that contain settings that specify which radio to match and settings that specify what action to take if a radio matches.

Provisioning rules for matching radios are configured in **/caps-man provisioning** menu:

Property	Description
action (*create-disabled* | *create-enabled* | *create-dynamic-enabled* | *none*; Default: **none**)	Action to take if rule matches are specified by the following settings: - **create-disabled** - create disabled static interfaces for radio. I.e., the interfaces will be bound to the radio, but the radio will not be operational until the interface is manually enabled; - **create-enabled** - create enabled static interfaces. I.e., the interfaces will be bound to the radio and the radio will be operational; - **create-dynamic-enabled** - create enabled dynamic interfaces. I.e., the interfaces will be bound to the radio, and the radio will be operational; - **none** - do nothing, leaves radio in non-provisioned state;
comment (*string*; Default:)	Short description of the Provisioning rule
common-name-regexp (*string*; Default:)	Regular expression to match radios by common name
hw-supported-modes (*a|a-turbo|ac|an|b|g|g-turbo|gn*; Default:)	Match radios by supported wireless modes
identity-regexp (*string*; Default:)	Regular expression to match radios by router identity
ip-address-ranges (*IpAddressRange[, IpAddressRanges] max 100x*; Default: **""**)	Match CAPs with IPs within configured address range.
master-configuration (*string*; Default:)	If **action** specifies to create interfaces, then a new master interface with its configuration set to this configuration profile will be created
name-format (*cap* | *identity* | *prefix* | *prefix-identity*; Default: **cap**)	specify the syntax of the CAP interface name creation - cap - default name - identity - CAP boards system identity name - prefix - name from the name-prefix value - prefix-identity - name from the name-prefix value and the CAP boards system identity name
name-prefix (*string*; Default:)	name prefix which can be used in the name-format for creating the CAP interface names

radio-mac (*MAC address*; Default: **00:00:00:00:00:00**)	MAC address of radio to be matched, empty MAC (00:00:00:00:00:00) means match all MAC addresses
slave-configurations (*string*; Default:)	If **action** specifies to create interfaces, then a new slave interface for each configuration profile in this list is created.

If no rule matches radio, then implicit default rule with action **create-enabled** and no configurations set is executed.

To get the active provisioning matchers:

```
[admin@CM] /caps-man provisioning> print
Flags: X - disabled
 0   radio-mac=00:00:00:00:00:00 action=create-enabled master-configuration=main-cfg
     slave-configurations=virtual-ap-cfg name-prefix=""
```

For user's convenience there are commands that allow the re-execution of the provisioning process for some radio or all radios provided by some AP:

```
[admin@CM] > caps-man radio provision 0
```

and

```
[admin@CM] > caps-man remote-cap provision 0
```

Interface Configuration

CAPsMAN interfaces are managed in **/caps-man interface** menu:

```
[admin@CM] > /caps-man interface print
Flags: M - master, D - dynamic, B - bound, X - disabled, I - inactive, R - running
 #      NAME                                RADIO-MAC         MASTER-INTERFACE
 0 M BR cap2                                00:0C:42:1B:4E:F5 none
 1   B  cap3                                00:00:00:00:00:00 cap2
```

Master Configuration Profiles

Configuration profiles permit pre-defined 'top level' master settings to be applied to CAP radios being provisioned.

Configuration Profiles are configured in **/caps-man configuration** menu:

Property	Description
channel (*list*; Default:)	User defined list taken from Channel names (**/caps-man channels**)
channel.band (*2ghz-b | 2ghz-b/g | 2ghz-b/g/n | 2ghz-onlyg | 2ghz-onlyn | 5ghz-a | 5ghz-a/n | 5ghz-onlyn | 5ghz-a/n/ac | 5ghz-only-ac*; Default:)	Defines set of used channels.
channel.control-channel-width (*40mhz-turbo | 20mhz | 10mhz | 5mhz*; Default:)	Defines set of used channel widths.
channel.extension-channel (*Ce | Ceee | eC | eCee | eeCe | eeeC | xx | xxxx | disabled*; Default:)	Extension channel configuration. (E.g. Ce = extension channel is above Control channel, eC = extension channel is below Control channel)
channel.frequency (*integer [0..4294967295]*; Default:)	Channel frequency value in MHz on which AP will operate. If left blank, CAPsMAN will automatically determine the best frequency that is least occupied.
channel.reselect-interval (*time [00:00:00]*; *[00:00:00..00:00:00];* Default:)	The interval after which the least occupied frequency is chosen, can be defined as a random interval, ex. as "30m..60m". Works only if **channel.frequency** is left blank.
channel.save-selected (*yes | no*; Default: **no**)	If channel frequency is chosen automatically and **channel.reselect-interval** is used, then saves the last picked frequency.
channel.secondary-frequency (*integer [0..4294967295]*; Default: **auto**)	Specifies the second frequency that will be used for 80+80MHz configuration. Set it to **Disabled** in order to disable 80+80MHz capability.

channel.skip-dfs-channels (*yes* / *no*; Default: **no**)	If **channel.frequency** is left blank, the selection will skip DFS channels
channel.tx-power (*integer [-30..40]*; Default:)	TX Power for CAP interface (for the whole interface not for individual chains) in dBm. It is not possible to set higher than allowed by country regulations or interface. By default max allowed by country or interface is used.
channel.width (; Default:)	Sets Channel Width in MHz.
comment (*string*; Default:)	Short description of the Configuration profile
country (*name of the country* / *no_country_set*; Default: **no_country_set**)	Limits available bands, frequencies and maximum transmit power for each frequency. Also specifies default value of **scan-list**. Value *no_country_set* is an FCC compliant set of channels.
datapath (*list*; Default:)	User defined list taken from Datapath names (**/caps-man datapath**)
datapath.bridge (*list*; Default:)	Bridge to which particular interface should be automatically added as port
datapath.bridge-cost (*integer [1..200000000]*; Default:)	bridge port cost to use when adding as bridge port
datapath.bridge-horizon (*integer [0..4294967295]*; Default:)	bridge horizon to use when adding as bridge port
datapath.client-to-client-forwarding (*yes* / *no*; Default: **no**)	controls if client-to-client forwarding between wireless clients connected to interface should be allowed, in local forwarding mode this function is performed by CAP, otherwise it is performed by CAPsMAN
datapath.interface-list (; Default:)	
datapath.l2mtu (; Default:)	set Layer2 MTU size
datapath.local-forwarding (*yes* / *no*; Default: **no**)	controls forwarding mode
datapath.mtu (; Default:)	set MTU size
datapath.openflow-switch (; Default:)	OpenFlow switch port (when enabled) to add interface to
datapath.vlan-id (*integer [1..4095]*; Default:)	VLAN ID to assign to interface if vlan-mode enables use of VLAN tagging
datapath.vlan-mode (*use-service-tag* / *use-tag*; Default:)	Enables and specifies the type of VLAN tag to be assigned to the interface (causes all received data to get tagged with VLAN tag and allows the interface to only send out data tagged with given tag)
disconnect-timeout (; Default:)	
distance (; Default:)	
frame-lifetime (; Default:)	
guard-interval (*any* / *long*; Default: **any**)	Whether to allow the use of short guard interval (refer to 802.11n MCS specification to see how this may affect throughput). "any" will use either short or long, depending on data rate, "long" will use long only.
hide-ssid (*yes* / *no*; Default:)	• *yes* - AP does not include SSID in the beacon frames and does not reply to probe requests that have broadcast SSID. • *no* - AP includes SSID in the beacon frames and replies to probe requests that have broadcast SSID. This property has effect only in AP mode. Setting it to *yes* can remove this network from the list of wireless networks that are shown by some client software. Changing this setting does not improve the security of the wireless network, because SSID is included in other frames sent by the AP.
hw-protection-mode (; Default:)	

hw-retries (; Default:)	
installation (*any* | *indoor* | *outdoor*; Default: **any**)	
keepalive-frames (*enabled* | *disabled*; Default: **enabled**)	
load-balancing-group (*string*; Default:)	Tags the interface to the load balancing group. For a client to connect to interface in this group, the interface should have the same number of already connected clients as all other interfaces in the group or smaller. Useful in setups where ranges of CAPs mostly overlap.
max-sta-count (*integer [1..2007]*; Default:)	Maximum number of associated clients.
mode (; Default: **ap**)	Set operational mode. Only ap currently supported.
multicast-helper (*default* | *disabled* | *full*; Default: **default**)	When set to full multicast packets will be sent with unicast destination MAC address, resolving multicast problem on a wireless link. This option should be enabled only on the access point, clients should be configured in **station-bridge** mode. Available starting from v5.15. • disabled - disables the helper and sends multicast packets with multicast destination MAC addresses • full - all multicast packet mac address are changed to unicast mac addresses prior sending them out • default - default choice that currently is set to *disabled*. Value can be changed in future releases.
name (*string*; Default:)	Descriptive name for the Configuration Profile
rates (; Default:)	User defined list taken from Rates names (**/caps-man rates**)
rates.basic (*1Mbps* | *2Mbps* | *5.5Mbps* | *6Mbps* | *11Mbps* | *11Mbps* | *12Mbps* | *18Mbps* | *24Mbps* | *36Mbps* | *48Mbps* | *54Mbps*; Default:)	
rates.supported (*1Mbps* | *2Mbps* | *5.5Mbps* | *6Mbps* | *11Mbps* | *11Mbps* | *12Mbps* | *18Mbps* | *24Mbps* | *36Mbps* | *48Mbps* | *54Mbps*; Default:)	
rates.ht-basic-mcs (*list of (mcs-0 | mcs-1 | mcs-2 | mcs-3 | mcs-4 | mcs-5 | mcs-6 | mcs-7 | mcs-8 | mcs-9 | mcs-10 | mcs-11 | mcs-12 | mcs-13 | mcs-14 | mcs-15 | mcs-16 | mcs-17 | mcs-18 | mcs-19 | mcs-20 | mcs-21 | mcs-22 | mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11n for MCS specification.
rates.ht-supported-mcs (*list of (mcs-0 | mcs-1 | mcs-2 | mcs-3 | mcs-4 | mcs-5 | mcs-6 | mcs-7 | mcs-8 | mcs-9 | mcs-10 | mcs-11 | mcs-12 | mcs-13 | mcs-14 | mcs-15 | mcs-16 | mcs-17 | mcs-18 | mcs-19 | mcs-20 | mcs-21 | mcs-22 | mcs-23)*; Default: **mcs-0; mcs-1; mcs-2; mcs-3; mcs-4; mcs-5; mcs-6; mcs-7; mcs-8; mcs-9; mcs-10; mcs-11; mcs-12; mcs-13; mcs-14; mcs-15; mcs-16; mcs-17; mcs-18; mcs-19; mcs-20; mcs-21; mcs-22; mcs-23**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11n for MCS specification.
rates.vht-basic-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **none**)	Modulation and Coding Schemes that every connecting client must support. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - client must support MCS-0 to MCS-7 • *MCS 0-8* - client must support MCS-0 to MCS-8 • *MCS 0-9* - client must support MCS-0 to MCS-9

rates.vht-supported-mcs (*none* | *MCS 0-7* | *MCS 0-8* | *MCS 0-9*; Default: **none**)	Modulation and Coding Schemes that this device advertises as supported. Refer to 802.11ac for MCS specification. You can set MCS interval for each of Spatial Stream • *none* - will not use selected Spatial Stream • *MCS 0-7* - devices will advertise as supported MCS-0 to MCS-7 • *MCS 0-8* - devices will advertise as supported MCS-0 to MCS-8 • *MCS 0-9* - devices will advertise as supported MCS-0 to MCS-9
rx-chains (*list of integer [0..2]*; Default: **0**)	Which antennas to use for receive.
security (*string*; Default: **none**)	Name of security configuration from **/caps-man security**
security.authentication-types (*list of string*; Default: **none**)	Specify the type of Authentication from **wpa-psk**, **wpa2-psk**, **wpa-eap** or **wpa2-eap**
security.disable-pmkid (; Default:)	
security.eap-methods (*eap-tls* | *passthrough*; Default: **none**)	• eap-tls - Use built-in EAP TLS authentication. • passthrough - Access point will relay authentication process to the RADIUS server.
security.eap-radius-accounting (; Default:)	
security.encryption (*aes-ccm* | *tkip*; Default: *aes-ccm*)	Set type of unicast encryption algorithm used
security.group-encryption (*aes-ccm* | *tkip*; Default: **aes-ccm**)	Access Point advertises one of these ciphers, multiple values can be selected. Access Point uses it to encrypt all broadcast and multicast frames. Client attempts connection only to Access Points that use one of the specified group ciphers. • tkip - Temporal Key Integrity Protocol - encryption protocol, compatible with legacy WEP equipment, but enhanced to correct some of the WEP flaws. • aes-ccm - more secure WPA encryption protocol, based on the reliable AES (Advanced Encryption Standard). Networks free of WEP legacy should use only this cipher.
security.group-key-update (*time: 30s..1h*; Default: **5m**)	Controls how often Access Point updates the group key. This key is used to encrypt all broadcast and multicast frames. property only has effect for Access Points.
security.passphrase (*string*; Default:)	WPA or WPA2 pre-shared key
security.tls-certificate (*none* | *name*; Default:)	Access Point always needs a certificate when configured when **security.tls-mode** is set to *verify-certificate*, or is set to *dont-verify-certificate*.
security.tls-mode (*verify-certificate* | *dont-verify-certificate* | *no-certificates*; Default:)	This property has effect only when **security.eap-methods** contains *eap-tls*. • verify-certificate - Require remote device to have valid certificate. Check that it is signed by known certificate authority. No additional identity verification is done. Certificate may include information about time period during which it is valid. If router has incorrect time and date, it may reject valid certificate because router's clock is outside that period. See also the Certificates configuration. • dont-verify-certificate - Do not check certificate of the remote device. Access Point will not require client to provide certificate. • no-certificates - Do not use certificates. TLS session is established using 2048 bit anonymous Diffie-Hellman key exchange.

ssid (*string (0..32 chars)*; Default:)	SSID (service set identifier) is a name broadcast in the beacons that identifies wireless network.
tx-chains (*list of integer [0..2]*; Default: **0**)	Which antennas to use for transmit.

Channel Groups

Channel group settings allows for the configuration of lists of radio channel related settings, such as radio band, frequency, Tx Power extension channel and width.

Channel group settings are configured in the Channels profile menu **/caps-man channels**

Property	Description
band (*2ghz-b | 2ghz-b/g | 2ghz-b/g/n | 2ghz-onlyg | 2ghz-onlyn | 5ghz-a | 5ghz-a/n | 5ghz-onlyn*; Default:)	Define operational radio frequency band and mode taken from hardware capability of wireless card
comment (*string*; Default:)	Short description of the Channel Group profile
extension-channel (*Ce | Ceee | eC | eCee | eeCe | eeeC | disabled*; Default:)	Extension channel configuration. (E.g. Ce = extension channel is above Control channel, eC = extension channel is below Control channel)
frequency (*integer [0..4294967295]*; Default:)	Channel frequency value in MHz on which AP will operate.
name (*string*; Default:)	Descriptive name for the Channel Group Profile
tx-power (*integer [-30..40]*; Default:)	TX Power for CAP interface (for the whole interface not for individual chains) in dBm. It is not possible to set higher than allowed by country regulations or interface. By default max allowed by country or interface is used.
width (; Default:)	Sets Channel Width in MHz. (E.g. 20, 40)
save-selected (; Default: **yes**)	Saves selected channel for the CAP Radio - will select this channel after the CAP reconnects to CAPsMAN and use it till the channel Re-optimize is done for this CAP.

Datapath Configuration

Datapath settings control data forwarding related aspects. On CAPsMAN datapath settings are configured in datapath profile menu **/caps-man datapath** or directly in a configuration profile or interface menu as settings with **datapath.** prefix.

There are 2 major forwarding modes:

- local forwarding mode, where CAP is locally forwarding data to and from wireless interface
- manager forwarding mode, where CAP sends to CAPsMAN all data received over wireless and only sends out the wireless data received from CAPsMAN. In this mode even client-to-client forwarding is controlled and performed by CAPsMAN.

Forwarding mode is configured on a per-interface basis - so if one CAP provides 2 radio interfaces, one can be configured to operate in local forwarding mode and the other in manager forwarding mode. The same applies to Virtual-AP interfaces - each can have different forwarding mode from master interface or other Virtual-AP interfaces.

Most of the datapath settings are used only when in manager forwarding mode, because in local forwarding mode CAPsMAN does not have control over data forwarding.

There are the following datapath settings:

- bridge -- bridge interface to add interface to, as a bridge port, when enabled
- bridge-cost -- bridge port cost to use when adding as bridge port
- bridge-horizon -- bridge horizon to use when adding as bridge port
- client-to-client-forwarding -- controls if client-to-client forwarding between wireless clients connected to interface should be allowed, in local forwarding mode this function is performed by CAP, otherwise it is performed by CAPsMAN.
- local-forwarding -- controls forwarding mode
- openflow-switch -- OpenFlow switch to add interface to, as port when enabled
- vlan-id -- VLAN ID to assign to interface if vlan-mode enables use of VLAN tagging
- vlan-mode -- VLAN tagging mode specifies if VLAN tag should be assigned to interface (causes all received data to get tagged with VLAN tag and allows interface to only send out data tagged with given tag)

Local Forwarding Mode

In this mode wireless interface on CAP behaves as a normal interface and takes part in normal data forwarding. Wireless interface will accept/pass data to networking stack on CAP. CAPsMAN will not participate in data forwarding and will not process any of data frames, it will only control interface configuration and client association process.

Wireless interface on CAP will change its configuration to 'enabled' and its state and some relevant parameters (e.g. mac-address, arp, mtu) will reflect that of the interface on CAPsMAN. Note that wireless related configuration **will not** reflect actual interface configuration as applied by CAPsMAN:

```
[admin@CAP] /interface wireless> pr
Flags: X - disabled, R - running
 0  R ;;; managed by CAPsMAN
      ;;; channel: 5180/20-Ceee/ac, SSID: master, local forwarding
      name="wlan2" mtu=1500 mac-address=00:03:7F:48:CC:07 arp=enabled
      interface-type=Atheros AR9888 mode=ap-bridge ssid="merlin"
      frequency=5240 band=5ghz-a/n channel-width=20/40mhz-eC scan-list=default
      ...
```

Virtual-AP interfaces in local forwarding mode will appear as enabled and dynamic Virtual-AP interfaces:

```
[admin@CAP] /interface> pr
Flags: D - dynamic, X - disabled, R - running, S - slave
 #     NAME                                TYPE         MTU L2MTU  MAX-L2MTU
 ...
 2  RS ;;; managed by CAPsMAN
       ;;; channel: 5180/20-Ceee/ac, SSID: master, local forwarding
       wlan2                               wlan        1500  1600
 3 DRS ;;; managed by CAPsMAN
       ;;; SSID: slave, local forwarding
       wlan6                               wlan        1500  1600
 ...
[admin@CAP] /interface> wireless pr
Flags: X - disabled, R - running
 ...
 2  R ;;; managed by CAPsMAN
      ;;; SSID: slave, local forwarding
      name="wlan6" mtu=1500 mac-address=00:00:00:00:00:00 arp=enabled
      interface-type=virtual-AP master-interface=wlan2
```

The fact that Virtual-AP interfaces are added as dynamic, somewhat limits static configuration possibilities on CAP for data forwarding, such as assigning addresses to Virtual-AP interface. This does not apply to master wireless interface.

To overcome this it is possible to use the static-virtual setting on the CAP which will create Static Virtual Interfaces instead of Dynamic and allows the possibility to assign IP configuration to those interfaces. MAC address is used to remember each static-interface when applying the configuration from the CAPsMAN. If two or more static interfaces will have the same MAC address the configuration could be applied in random order.

To facilitate data forwarding configuration, CAP can be configured with bridge to which interfaces are automatically added as ports when interfaces are enabled by CAPsMAN. This can be done in **/interface wireless cap** menu.

Manager Forwarding Mode

In this mode CAP sends all data received over wireless to CAPsMAN and only sends out over wireless, data received from CAPsMAN. CAPsMAN has full control over data forwarding including client-to-client forwarding. Wireless interface on CAP is disabled and does not participate in networking:

```
 ...
 1 X  ;;; managed by CAPsMAN
      ;;; channel: 5180/20-Ceee/ac, SSID: master, manager forwarding
      name="wlan2" mtu=1500 mac-address=00:03:7F:48:CC:07 arp=enabled
      interface-type=Atheros AR9888 mode=ap-bridge ssid="merlin"
 ...
```

Virtual-AP interfaces are also created as 'disabled' and do not take part in data forwarding on CAP.

Access List

Access list on CAPsMAN is an ordered list of rules that is used to allow/deny clients to connect to any CAP under CAPsMAN control. When client attempts to connect to a CAP that is controlled by CAPsMAN, CAP forwards that request to CAPsMAN. As a part of registration process, CAPsMAN consults access list to determine if client should be allowed to connect. The default behaviour of the access list is to allow connection.

Access list rules are processed one by one until matching rule is found. Then the action in the matching rule is executed. If action specifies that client should be accepted, client is accepted, potentially overriding it's default connection parameters with ones specified in access list rule.

Access list is configured in **/caps-man access-list** menu. There are the following parameters for access list rules:

- client matching parameters:
  - address - MAC address of client (or, if mask is specified, only those parts will be checked as per the mask, so to match vendor D8 from "D8:1C:79:6E:1E:FE", simply enter a bogus entry, such as "D8:00:00:00:00" and then use the mask as per next line)
  - mask - MAC address mask to apply when comparing client address. For example, use FF:00:00:00:00:00 to match only the first octet of the specified MAC address. In above example, regardless of entered MAC, it will match only first octet. Similarly, entering 00:00:00:00:FF will only match the last octet (FE) of a hypotetical MAC "D8:1C:79:6E:1E:FE"). So in the mac line, you could just enter 00:00:00:00:00:FE, if you would use such a mask.
  - interface - optional interface to compare with interface to which client actually connects to
  - time - time of day and days when rule matches
  - signal-range - range in which client signal must fit for rule to match
- action parameter - specifies action to take when client matches:
  - accept - accept client
  - reject - reject client
  - query-radius - query RADIUS server if particular client is allowed to connect
- connection parameters:
  - ap-tx-limit - tx speed limit in direction to client
  - client-tx-limit - tx speed limit in direction to AP (applies to RouterOS clients only)
  - client-to-client-forwarding - specifies whether to allow forwarding data received from this client to other clients connected to the same interface
  - private-passphrase - PSK passphrase to use for this client if some PSK authentication algorithm is used
  - radius-accounting - specifies if RADIUS traffic accounting should be used if RADIUS authentication gets done for this client
  - vlan-mode - VLAN tagging mode specifies if traffic coming from client should get tagged (and untagged when going to client).
  - vlan-id - VLAN ID to use if doing VLAN tagging.

Registration Table

Registration table contains a list of clients that are connected to radios controlled by CAPsMAN and is available in **/caps-man registration-table** menu:

```
[admin@CM] /caps-man> registration-table print
 # INTERFACE                  MAC-ADDRESS       UPTIME               RX-SIGNAL
 0 cap1                       00:03:7F:48:CC:0B 1h38m9s210ms                -36
```

Examples

Basic configuration with master and slave interface

Create security profile for WPA2 PSK, without specifying passphrase:

```
[admin@CM] /caps-man security>add name="wpa2psk" authentication-types=wpa2-psk encryption=aes-ccm
```

Create configuration profile to be used by master interface

- specify WPA2 passphrase in configuration
- specify channel settings in configuration:

```
[admin@CM] /caps-man configuration> add name=master-cfg ssid=master security=wpa2psk
security.passphrase=12345678 channel.frequency=5180 channel.width=20 channel.band=5ghz-a
```

Create configuration profile to be used by virtual AP interface

- specify different WPA2 passphrase in configuration:

```
[admin@CM] /caps-man configuration> add name=slave-cfg ssid=slave security=wpa2psk
security.passphrase=87654321
```

Create provisioning rule that matches any radio and creates dynamic interfaces using master-cfg and slave-cfg:

```
[admin@CM] /caps-man provisioning> add action=create-dynamic-enabled master-configuration=master-cfg
slave-configurations=slave-cfg
```

Now when AP connects and is provisioned 2 dynamic interfaces (one master and one slave) will get created:

```
[admin@CM] /caps-man interface> print detail
Flags: M - master, D - dynamic, B - bound, X - disabled, I - inactive, R - running
 0 MDB  name="cap1" mtu=1500 l2mtu=2300 radio-mac=00:0C:42:1B:4E:F5 master-interface=none
        configuration=master-cfg

 1  DB  name="cap2" mtu=1500 l2mtu=2300 radio-mac=00:00:00:00:00:00 master-interface=cap1
        configuration=slave-cfg
```

Consider an AP, that does not support configured frequency connects and can not become operational:

```
[admin@CM] /caps-man interface> pr
Flags: M - master, D - dynamic, B - bound, X - disabled, I - inactive, R - running
 #      NAME                                RADIO-MAC         MASTER-INTERFACE
 0 MDB  ;;; unsupported band or channel
        cap3                                00:0C:42:1B:4E:FF none
 ...
```

We can override channel settings for this particular radio in interface settings, without affecting master-cfg profile:

```
[admin@CM] /caps-man interface> set cap3 channel.frequency=2142 channel.band=2ghz-b/g
```

Allow Specific MAC address range to match the Access-list, for example, match all the Apple devices:

```
[admin@CM] /caps-man access-list> add mac-address=18:34:51:00:00:00 mac-address-mask=FF:FF:FF:00:00:00
action=accept
```

Configuring DHCP Server Option 138 for setting the CAPsMAN address on the CAP boards

```
[admin@CM] /ip dhcp-server network set <network-id> caps-manager=<capsman-server-ip>
```

DHCP client this CAPsMAN IP will see in "/ip dhcp-client print detail"

Configuration with certificates

You would want to configure certificates in your CAPsMAN to use options as *Require Peer Certificate* and *Lock To Caps Man*. These options increase security and in some cases stability of your CAPsMAN network. CAPs won't connect to CAPsMAN without a specific certificate and vice versa.

Fast and easy configuration

This is a basic configuration for using certificates in your CAPsMAN setup. This example assumes that you already have basic configuration on your CAPsMAN and CAP. It is best to use this configuration in CAPsMAN networks which are not constantly growing. For more details read about CAP to CAPsMAN Connection.

CAPsMAN device:

In CAPsMAN Manager menu set *Certificate* and *CA Certificate* to *auto*:

```
/caps-man manager
set ca-certificate=auto certificate=auto
```

Print output:

```
[admin@CAPsMAN] /caps-man manager print
                  enabled: yes
              certificate: auto
           ca-certificate: auto
             package-path:
           upgrade-policy: none
  require-peer-certificate: no
     generated-certificate: CAPsMAN-D4CA6D987C26
  generated-ca-certificate: CAPsMAN-CA-D4CA6D987C26
```

CAPsMAN device first will generate *CA-Certificate* and then it will generate *Certificate* which depends on *CA-Certificate*.

CAP device:

Set in CAP configuration to *request* certificate:

```
/interface wireless cap
set certificate=request
```

CAP will connect to CAPsMAN and request certificate. CAP will receive *CA-Certificate* form CAPsMAN and another certificate will be created for use on CAP.

In Result

On CAP device in CAP menu *Requested Certificate* is set:

```
[admin@CAP] /interface wireless cap print
                           enabled: yes
                        interfaces: wlan1
                       certificate: request
                  lock-to-caps-man: no
               discovery-interfaces: ether1
                 caps-man-addresses:
                    caps-man-names:
  caps-man-certificate-common-names:
                            bridge: none
                    static-virtual: no
         -->  requested-certificate: CAP-D4CA6D7F45BA  <--
```

Also, two certificates are gained and are seen in *Certificate* menu:

```
[admin@CAP] > /certificate print
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked, E -
expired, T - trusted
 #          NAME              COMMON-NAME              SUBJECT-ALT-NAME
FINGERPRINT
 0     A  T _0                CAPsMAN-CA-D4CA6D987C26
383e63d7b...
 1 K        CAP-D4CA6D7F45BA  CAP-D4CA6D7F45BA
d495d1a94...
```

On CAPsMAN device in Certificate menu three certificates are created. CAPsMAN and CAPsMAN-CA certificates, as well as a certificate which is issued to CAP:

```
[admin@CAPsMAN] > /certificate print
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked, E -
expired, T - trusted
 #          NAME                     COMMON-NAME              SUBJECT-ALT-NAME
FINGERPRINT
 0 K   A  T CAPsMAN-CA-D4CA6D987C26  CAPsMAN-CA-D4CA6D987C26
383e63d7b...
 1 K    I   CAPsMAN-D4CA6D987C26     CAPsMAN-D4CA6D987C26
02b0f7ff4...
 2      I   issued_1                 CAP-D4CA6D7F45BA
d495d1a94...
```

Additionally

If you want to allow only CAPs with a valid certificate to connect to this CAPsMAN you can set *Require Peer Certificate* to *yes* on CAPsMAN device:

```
/caps-man manager
set require-peer-certificate=yes
```

However, when you will want to add new CAP devices to your CAPsMAN network you will have to set this option to *no* and then back to *yes* after CAP has gained certificates. Every time you change this option CAPsMAN will drop all dynamic interfaces and CAPs will try to connect again.

If you want to lock CAP to specific CAPsMAN and be sure it won't connect to other CAPsMANs you should set option *Lock To CAPsMAN* to *yes*. Additionally, you can specify CAPsMAN to lock to by setting *CAPsMAN Certificate Common Names* on CAP device:

```
/interface wireless cap
set lock-to-caps-man=yes
set caps-man-certificate-common-names=CAPsMAN-D4CA6D987C26
```

Manual certificates and issuing with SCEP

With this example, you can create your own certificates for CAPsMAN and take control over issuing certificates to CAPs. This configuration can be useful in big, growing CAPsMAN networks. Many segments of this example can be done differently depending on your situation and needs. At this point, some knowledge about Certificates and their application can be useful.

CAPsMAN device:

In *Certificate* menu add certificate templates for CA certificate and CAPsMAN server certificate:

```
/certificate
add name=CA-temp common-name=CA
add name=CAPsMAN-temp common-name=CAPsMAN
```

Now *Sign* the certifiace templates. First *Sign* the CA certificate and use CAPsMAN device IP as *CA CRL Host*:

```
/certificate
sign CA-temp ca-crl-host=10.5.138.157 name=CA
sign CAPsMAN-temp ca=CA name=CAPsMAN
```

Alternatively, previous two steps can be done with auto setting in *Certificate* and *CA-Certificate* option in CAPsMAN Manager menu, see the Fast and easy configuration.

Export CA certificate. You will have to *Import* it on CAP device. You can use *Download -> Drag&Drop* to CAP device, in this example *fetch* command is used later from CAP device. Using long passphrase is advisable - longer passphrase will take longer to crack if it gets into the wrong hands:

```
/certificate
export-certificate CA export-passphrase=thelongerthebetterpassphrase
```

Create *SCEP server* which will be used to issue and grant certificates to CAP devices:

```
/certificate scep-server
add ca-cert=CA path=/scep/CAPsMAN
```

Set certificates in CAPsMAN Manager menu and set *Require Peer Certificate* to yes:

```
/caps-man manager
set ca-certificate=CA certificate=CAPsMAN
set require-peer-certificate=yes
```

At this point, only CAPs with a valid certificate will be able to connect.

CAP device

Download export of CA certificate from CAPsMAN device to CAP device. In this example *fetch* is used, however, there are multiple other ways:

```
/tool fetch address=10.5.138.157 src-path=cert_export_CA.crt user=admin password="123" mode=ftp
```

Import CA certificate from CAPsMAN device in *Certificate* menu:

```
/certificate> import file-name=cert_export_CA.crt passphrase=thelongerthebetterpassphrase
```

Add certificate template for CAP:

```
/certificate
add name=CAP1 common-name=CAP1
```

Ask CAPsMAN device to grant this certificate with a key using SCEP:

```
/certificate
add-scep template=CAP1 scep-url="https://10.5.138.157/scep/CAPsMAN"
```

You will have to return to CAPsMAN device to grant key to this certificate.

In CAP menu set just created certificate:

```
/interface wireless cap
set certificate=CAP1
```

CAPsMAN device:

Return to CAPsMAN device to grant a key to CAP certificate in *Certificate Request* menu:

```
/certificate scep-server requests
grant numbers=0
```

In Result

Now CAP should be able to connect to CAPsMAN, see in *CAPsMAN interfaces* if it connects. In CAPsMAN device *Certificate* menu three certificates can be seen: CA, CAPsMAN, and the one which is issued to CAP:

```
[admin@CAPsMAN] /certificate print
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked,
E - expired, T - trusted
 #          NAME        COMMON-NAME      SUBJECT-ALT-NAME                                    FINGERPRINT
 0 K L A  T CA          CA                                                                   752775b457a37...
 1 K   A    CAPsMAN     CAPsMAN                                                              12911ba445b3b...
 2      I   issued_1    CAP1                                                                 5b9a52b6ce3fb...
```

In CAP devices *Certificate* menu two acquired certificates can be seen:

```
[admin@CAP1] /interface wireless> /certificate print
Flags: K - private-key, D - dsa, L - crl, C - smart-card-key, A - authority, I - issued, R - revoked,
E - expired, T - trusted
 #          NAME        COMMON-NAME      SUBJECT-ALT-NAME                                    FINGERPRINT
 0   L A  T cert_exp... CA                                                                   752775b457a37...
 1 K      T CAP1        CAP1                                                                 5b9a52b6ce3fb...
```

HWMPplus (Mesh)

Summary

```
/interface mesh
```

HWMP+ is a MikroTik specific layer-2 routing protocol for wireless mesh networks. It is based on Hybrid Wireless Mesh Protocol (HWMP) from IEEE 802.11s draft standard. It can be used instead of (Rapid) Spanning Tree protocols in mesh setups to ensure loop-free optimal routing.

The HWMP+ protocol however is not compatible with HWMP from IEEE 802.11s draft standard.

Note that the distribution system you use for your network need not be a Wireless Distribution System (WDS). HWMP+ mesh routing supports not only WDS interfaces but also Ethernet interfaces inside the mesh. So you can use a simple Ethernet-based distribution system, or you can combine both WDS and Ethernet links!

HWMPplus is not supported on Wifi interfaces, but can be used on Wireless interfaces.

Properties

Mesh

Property	Description
admin-mac (*MAC address*; **Default: 00:00:00:00:00:00)**	Administratively assigned MAC address, used when the **auto-mac** setting is disabled
arp (*disabled | enabled | proxy-arp | reply-only*; Default: **enabled**)	Address Resolution Protocol setting
auto-mac (*boolean*; Default: **no**)	If disabled, then the value from **admin-mac** will be used as the MAC address of the mesh interface; else address of some port will be used if ports are present
hwmp-default-hoplimit (*integer: 1..255*; Default:)	Maximum hop count for generated routing protocol packets; after an HWMP+ packet is forwarded "hoplimit" times, it is dropped
hwmp-prep-lifetime (*time*; Default: **5m**)	Lifetime for routes created from received PREP or PREQ messages
hwmp-preq-destination-only (*boolean*; Default: **yes**)	Whether the only destination can respond to HWMP+ PREQ message
hwmp-preq-reply-and-forward (*boolean*; Default: **yes**)	Whether intermediate nodes should forward HWMP+ PREQ message after responding to it. Useful only when **hwmp-preq-destination-only** is disabled
hwmp-preq-retries (*integer*; Default: **2**)	How many times to retry a route discovery to a specific MAC address before the address is considered unreachable
hwmp-preq-waiting-time (*time*; Default: **4s**)	How long to wait for a response to the first PREQ message. Note that for subsequent PREQs the waiting time is increased exponentially
hwmp-rann-interval (*time*; Default: **10s**)	How often to send out HWMP+ RANN messages
hwmp-rann-lifetime (*time*; Default: **1s**)	Lifetime for routes created from received RANN messages
hwmp-rann-propagation-delay (*number*; Default: **0.5**)	How long to wait before propagating a RANN message. Value in seconds

mesh-portal (*boolean*; Default: **no**)	Whether this interface is a portal in the mesh network
mtu (*number*; Default: **1500**)	Maximum transmission unit size
name (*string*; Default:)	Interface name
reoptimize-paths (*boolean*; Default: **no**)	Whether to send out periodic PREQ messages asking for known MAC addresses. Turning on this setting is useful if the network topology is changing often. Note that if no reply is received to a re-optimization PREQ, the existing path is kept anyway (until it timeouts itself)

Port

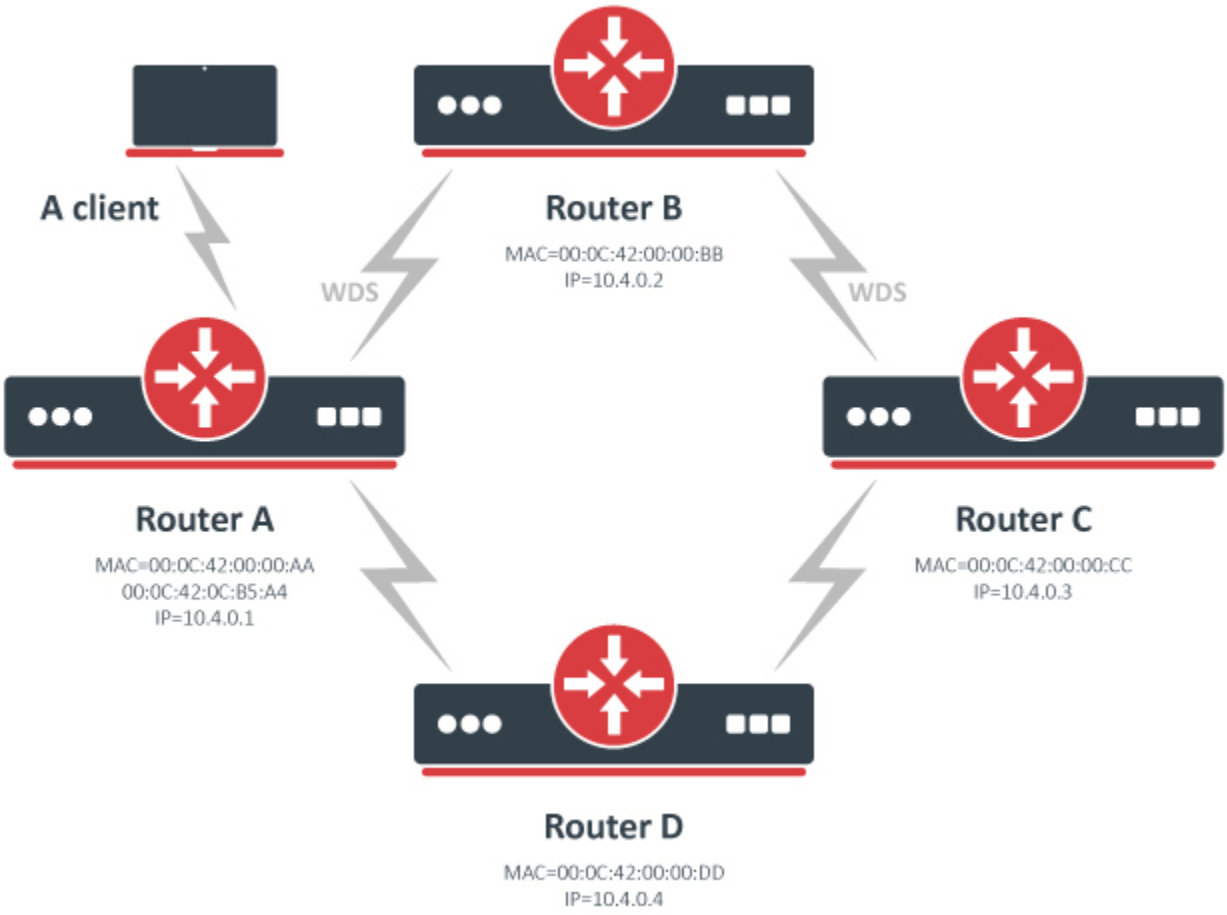
Property	Description
active-port-type (*read-only: wireless | WDS | ethernet-mesh | ethernet-bridge | ethernet-mixed*; Default:)	port type and state actually used
hello-interval (*time*; Default: **10s**)	the maximum interval between sending out HWMP+ Hello messages. Used only for Ethernet type ports
interface (*interface name*; Default:)	interface name, which is to be included in a mesh
mesh (*interface name*; Default:)	mesh interface this port belongs to
path-cost (*integer: 0..65535*; Default: **10**)	path cost to the interface, used by routing protocol to determine the 'best' path
port-type (*WDS | auto | ethernet | wireless*; Default:)	port type to use • auto - port type is determined automatically based on the underlying interface's type • WDS - a Wireless Distribution System interface. Remote MAC address is learned from wireless connection data • ethernet - Remote MAC addresses are learned either from HWMP+ Hello messages or from source MAC addresses in received or forwarded traffic • wireless - Remote MAC addresses are learned from wireless connection data

FDB Status

Property	Description
mac-address (*MAC address*)	MAC address corresponding for this FDB entry
seq-number (*integer*)	sequence number used in routing protocol to avoid loops
type (*integer*)	sequence number used in routing protocol to avoid loops
interface (*local | outsider | direct | mesh | neighbor | larval | unknown*)	type of this FDB entry • local -- MAC address belongs to the local router itself • outsider -- MAC address belongs to a device external to the mesh network • direct -- MAC address belongs to a wireless client on an interface that is in the mesh network • mesh -- MAC address belongs to a device reachable over the mesh network; it can be either internal or external to the mesh network • neighbor -- MAC address belongs to a mesh router that is a direct neighbor to this router • larval -- MAC address belongs to an unknown device that is reachable over the mesh network • unknown -- MAC address belongs to an unknown device
mesh (*interface name*)	the mesh interface this FDB entry belongs to
on-interface (*interface name*)	mesh port used for traffic forwarding, kind of a next-hop value
lifetime (*time*)	time remaining to live if this entry is not used for traffic forwarding

age (*time*)	age of this FDB entry
metric (*integer*)	a metric value used by routing protocol to determine the 'best' path

Example



This example uses static WDS links that are dynamically added as mesh ports when they become active. Two different frequencies are used: one for AP interconnections, and one for client connections to APs, so the AP must have at least two wireless interfaces. Of course, the same frequency for all connections also could be used, but that might not work as well because of potential interference issues.

Repeat this configuration on all APs:

```
/interface mesh add disabled=no
/interface mesh port add interface=wlan1 mesh=mesh1
/interface mesh port add interface=wlan2 mesh=mesh1

 # interface used for AP interconnections
/interface wireless set wlan1 disabled=no ssid=mesh frequency=2437 band=2.4ghz-b/g/n mode=ap-bridge \
 wds-mode=static-mesh wds-default-bridge=mesh1

# interface used for client connections
 /interface wireless set wlan2 disabled=no ssid=mesh-clients frequency=5180 band=5ghz-a/n/ac mode=ap-bridge

# a static WDS interface for each AP you want to connect to
/interface wireless wds add disabled=no master-interface=wlan1 name=<descriptive name of remote end> \
 wds-address=<MAC address of remote end>
```

Here WDS interface is added manually because static WDS mode is used. If you are using **wds-mode=dynamic-mesh**, all WDS interfaces will be created automatically. The **frequency** and **band** parameters are specified here only to produce valid example configuration; mesh protocol operations are by no means limited to or optimized for, these particular values.

> ⚠ You may want to increase the **disconnect-timeout** wireless interface option to make the protocol more stable.

In real-world setups you also should take care of securing the wireless connections, using **/interface wireless security-profile**. For simplicity, that configuration is not shown here.

Results on router A (there is one client connected to wlan2):

```
[admin@A] > /interface mesh print
Flags: X - disabled, R - running
0 R name="mesh1" mtu=1500 arp=enabled mac-address=00:0C:42:0C:B5:A4 auto-mac=yes
admin-mac=00:00:00:00:00:00 mesh-portal=no hwmp-default-hoplimit=32
hwmp-preq-waiting-time=4s hwmp-preq-retries=2 hwmp-preq-destination-only=yes
hwmp-preq-reply-and-forward=yes hwmp-prep-lifetime=5m hwmp-rann-interval=10s
hwmp-rann-propagation-delay=1s hwmp-rann-lifetime=22s

[admin@A] > /interface mesh port print detail
Flags: X - disabled, I - inactive, D - dynamic
0 interface=wlan1 mesh=mesh1 path-cost=10 hello-interval=10s port-type=auto port-type-used=wireless
1 interface=wlan2 mesh=mesh1 path-cost=10 hello-interval=10s port-type=auto port-type-used=wireless
2 D interface=router_B mesh=mesh1 path-cost=105 hello-interval=10s port-type=auto port-type-used=WDS
3 D interface=router_D mesh=mesh1 path-cost=76 hello-interval=10s port-type=auto port-type-used=WDS
```

The FDB (Forwarding Database) at the moment contains information only about local MAC addresses, non-mesh nodes reachable through a local interface, and direct mesh neighbors:

```
[admin@A] /interface mesh fdb print
Flags: A - active, R - root
MESH TYPE MAC-ADDRESS ON-INTERFACE LIFETIME AGE
A mesh1 local 00:0C:42:00:00:AA 3m17s
A mesh1 neighbor 00:0C:42:00:00:BB router_B 1m2s
A mesh1 neighbor 00:0C:42:00:00:DD router_D 3m16s
A mesh1 direct 00:0C:42:0C:7A:2B wlan2 2m56s
A mesh1 local 00:0C:42:0C:B5:A4 2m56s

[admin@A] /interface mesh fdb print detail
Flags: A - active, R - root
A mac-address=00:0C:42:00:00:AA type=local age=3m21s mesh=mesh1 metric=0 seqnum=4294967196
A mac-address=00:0C:42:00:00:BB type=neighbor on-interface=router_B age=1m6s mesh=mesh1 metric=132
seqnum=4294967196
A mac-address=00:0C:42:00:00:DD type=neighbor on-interface=router_D age=3m20s mesh=mesh1 metric=79
seqnum=4294967196
A mac-address=00:0C:42:0C:7A:2B type=direct on-interface=wlan2 age=3m mesh=mesh1 metric=10 seqnum=0
A mac-address=00:0C:42:0C:B5:A4 type=local age=3m mesh=mesh1 metric=0 seqnum=0
```

Test if ping works:

```
[admin@A] > /ping 00:0C:42:00:00:CC
00:0C:42:00:00:CC 64 byte ping time=108 ms
00:0C:42:00:00:CC 64 byte ping time=51 ms
00:0C:42:00:00:CC 64 byte ping time=39 ms
00:0C:42:00:00:CC 64 byte ping time=43 ms
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 39/60.2/108 ms
```

Router A had to discover a path to Router C first, hence the slightly larger time for the first ping. Now the FDB also contains an entry for 00:0C:42:00:00:CC, with type "mesh".

Also, test that ARP resolving works and so does IP level ping:

```
[admin@A] > /ping 10.4.0.3
10.4.0.3 64 byte ping: ttl=64 time=163 ms
10.4.0.3 64 byte ping: ttl=64 time=46 ms
10.4.0.3 64 byte ping: ttl=64 time=48 ms
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 46/85.6/163 ms
```

Mesh traceroute

There is also a mesh traceroute command, that can help you to determine which paths are used for routing.

For example, for this network:

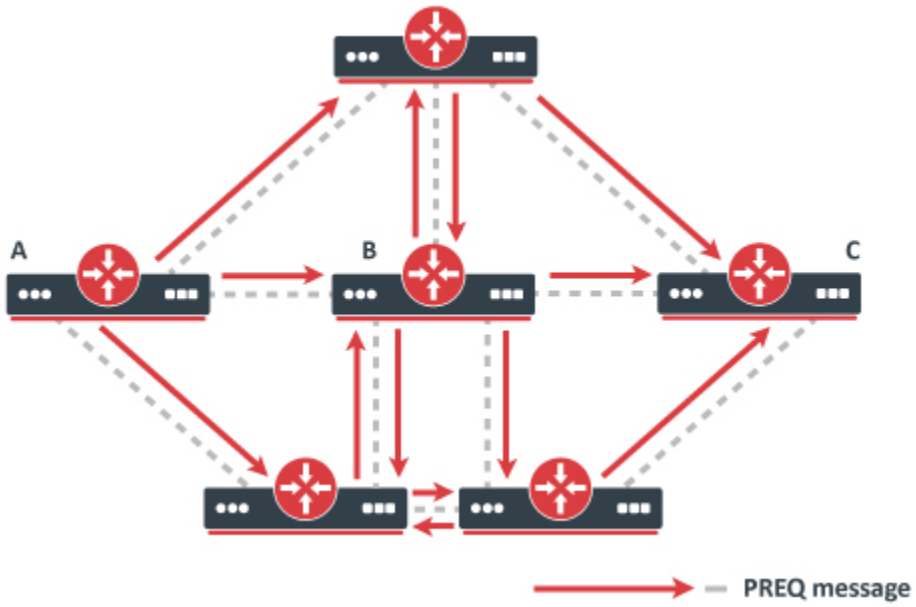
```
[admin@1] /interface mesh fdb print
Flags: A - active, R - root
MESH TYPE MAC-ADDRESS ON-INTERFACE LIFETIME AGE
A mesh1 local 00:0C:42:00:00:01 7m1s
A mesh1 mesh 00:0C:42:00:00:02 wds4 17s 4s
A mesh1 mesh 00:0C:42:00:00:12 wds4 4m58s 1s
A mesh1 mesh 00:0C:42:00:00:13 wds4 19s 2s
A mesh1 neighbor 00:0C:42:00:00:16 wds4 7m1s
A mesh1 mesh 00:0C:42:00:00:24 wds4 18s 3s
```

Traceroute to 00:0C:42:00:00:12 shows:

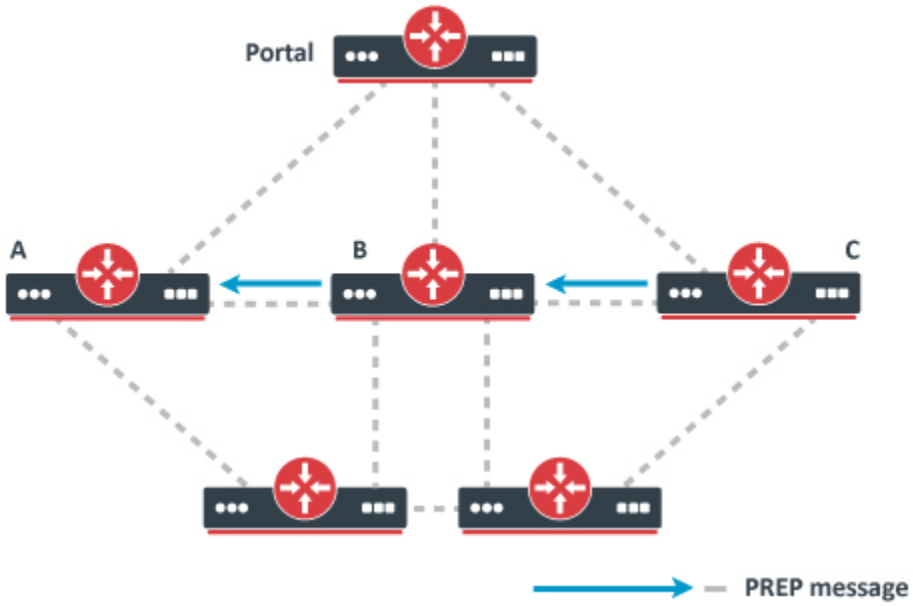
```
[admin@1] /interface mesh traceroute mesh1 00:0C:42:00:00:12
ADDRESS TIME STATUS
00:0C:42:00:00:16 1ms ttl-exceeded
00:0C:42:00:00:02 2ms ttl-exceeded
00:0C:42:00:00:24 4ms ttl-exceeded
00:0C:42:00:00:13 6ms ttl-exceeded
00:0C:42:00:00:12 6ms success
```

Protocol description

Reactive mode



Router A wants to discover a path to C

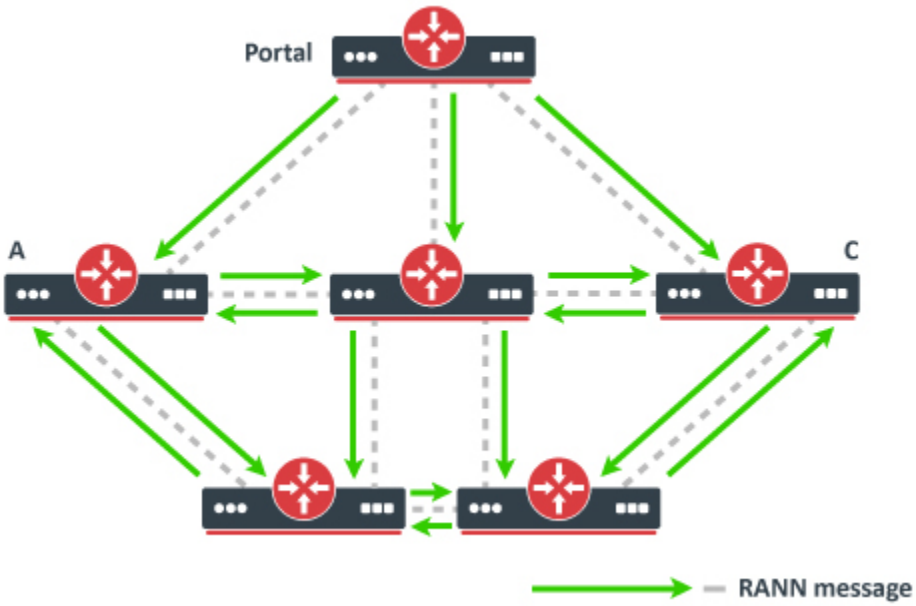


Router C sends a unicast response to A

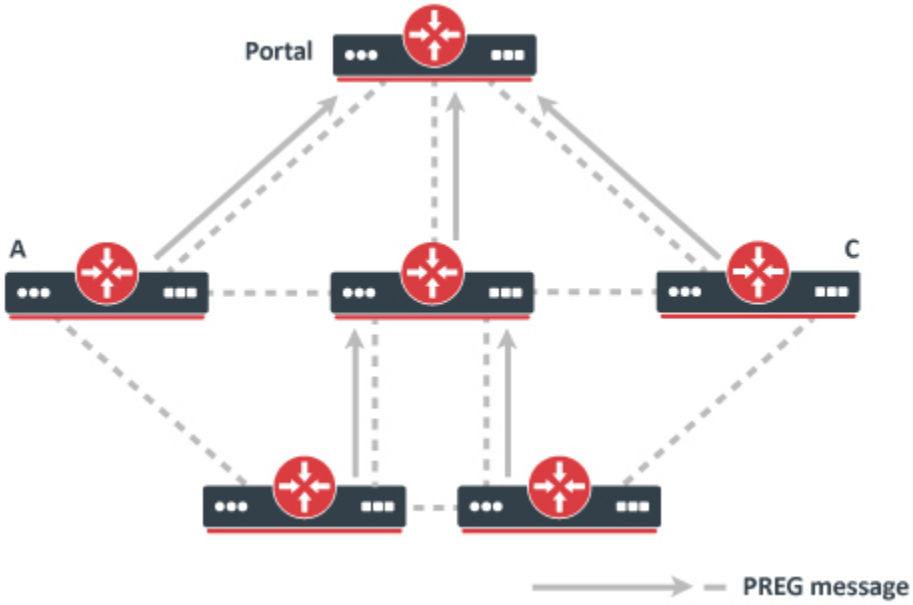
In reactive mode, HWMP+ is very much like AODV (Ad-hoc On-demand Distance Vector). All paths are discovered on-demand, by flooding Path Request (PREQ) message in the network. The destination node or some router that has a path to the destination will reply with a Path Response (PREP). Note that if the destination address belongs to a client, the AP this client is connected to will serve as a proxy for him (i.e. reply to PREQs on his behalf).

This mode is best suited for mobile networks, and/or when most of the communication happens between intra-mesh nodes.

Proactive mode



The root announces itself by flooding RANN



Internal nodes respond with PREGs

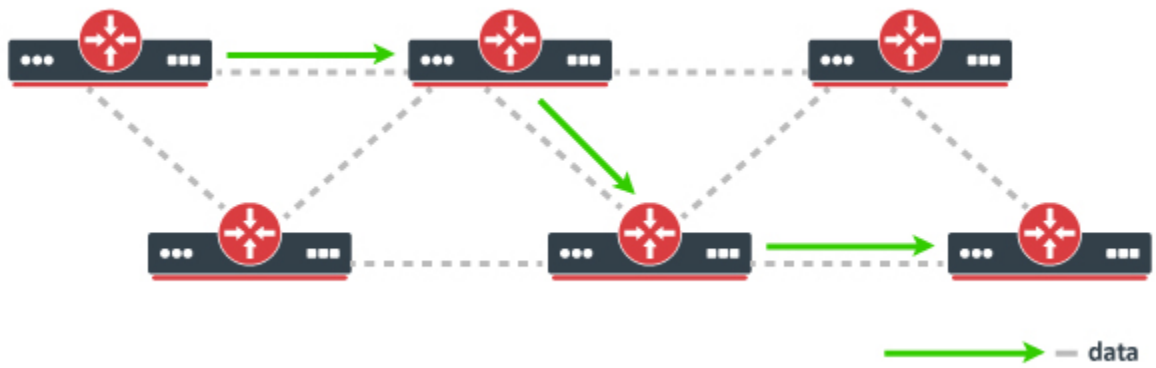
In proactive mode, there are some routers configured as portals. In general, being a portal means that the router has interfaces to some other network, i.e. it is an entry/exit point to the mesh network.

The portals will announce their presence by flooding the Root Announcement (RANN) message in the network. Internal nodes will reply with a Path Registration (PREG) message. The result of this process will be routing trees with roots in the portal.

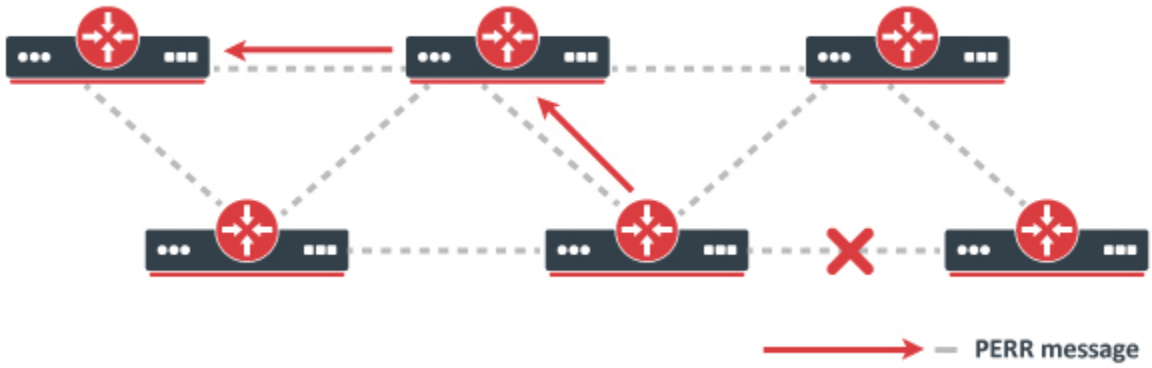
Routes to portals will serve as a kind of default route. If an internal router does not know the path to a particular destination, it will forward all data to its closest portal. The portal will then discover the path on behalf of the router if needed. The data afterward will flow through the portal. This may lead to sub-optimal routing unless the data is addressed to the portal itself or some external network the portals have interfaces to.

A proactive mode is best suited when most of the traffic goes between internal mesh nodes and a few portal nodes.

Topology change detection



Data flow path



After the link disappears, an error is propagated upstream

HWMP+ uses Path Error (PERR) message to notify that a link has disappeared. The message is propagated to all upstream nodes up to the data source. The source on PERR reception restarts the path discovery process.

FAQ

Q. How is this better than RSTP?

A. It gives you optimal routing. RSTP is only for loop prevention.

Q. How the route selection is done?

A. The route with the best metric is always selected after the discovery process. There is also a configuration option to periodically reoptimize already known routes.

Route metric is calculated as the sum of individual link metrics.

Link metric is calculated in the same way as for (R)STP protocols:

- For Ethernet links the metric is configured statically (same as for OSPF, for example).
- For WDS links the metric is updated dynamically depending on actual link bandwidth, which in turn is influenced by wireless signal strength, and the selected data transfer rate.

Currently, the protocol does not take into account the amount of bandwidth being used on a link, but that might be also used in the future.

Q. How is this better than OSPF/RIP/layer-3 routing in general?

A. WDS networks usually are bridged, not routed. The ability to self-configure is important for mesh networks, and routing generally requires much more configuration than bridging. Of course, you can always run any L3 routing protocol over a bridged network, but for mesh networks that usually makes little sense.

⚠ Since optimized layer-2 multicast forwarding is not included in the mesh protocol, it is better to avoid forwarding any multicast traffic (including OSPF) over meshed networks. If you need OSPF, then you have to configure OSPF NBMA neighbors that use unicast mode instead.

Q. What about performance/CPU requirements?

A. The protocol itself, when properly configured, will take much fewer resources than OSPF (for example) would. Data forwarding performance on an individual router should be close to that of bridging.

Q. How does it work together with existing mesh setups that are using RSTP?

A. The internal structure of an RSTP network is transparent to the mesh protocol (because mesh hello packets are forwarded inside the RSTP network). The mesh will see the path between two entry points in the RSTP network as a single segment. On the other hand, a mesh network is not transparent to the RSTP, since RSTP hello packets are not be forwarded inside the mesh network. *(This is the behavior since v3.26)*

⛔ Routing loops are possible if a mesh network is attached to an RSTP network in two or more points!

Note that if you have a WDS link between two access points, then both ends must have the same configuration (either as ports in a mesh on both ends or as ports in a bridge interface on both ends).

You can also put a bridge interface as a mesh port (to be able to use a bridge firewall, for example).

Q. Can I have multiple entry/exit points to the network?

A. If the entry/exit points are configured as portals (i.e. proactive mode is used), each router inside the mesh network will select its closest portal and forward all data to it. The portal will then discover a path on behalf of the router if needed.

Q. How to control or filter mesh traffic?

A. At the moment the only way is to use a bridge firewall. Create a bridge interface, put the WDS interfaces and/or Ethernets in that bridge, and put that bridge in a mesh interface. Then configure bridge firewall rules.

To match MAC protocol used for mesh traffic encapsulation, use MAC protocol number 0x9AAA, and to match mesh routing traffic, use MAC protocol number 0x9AAB. Example:

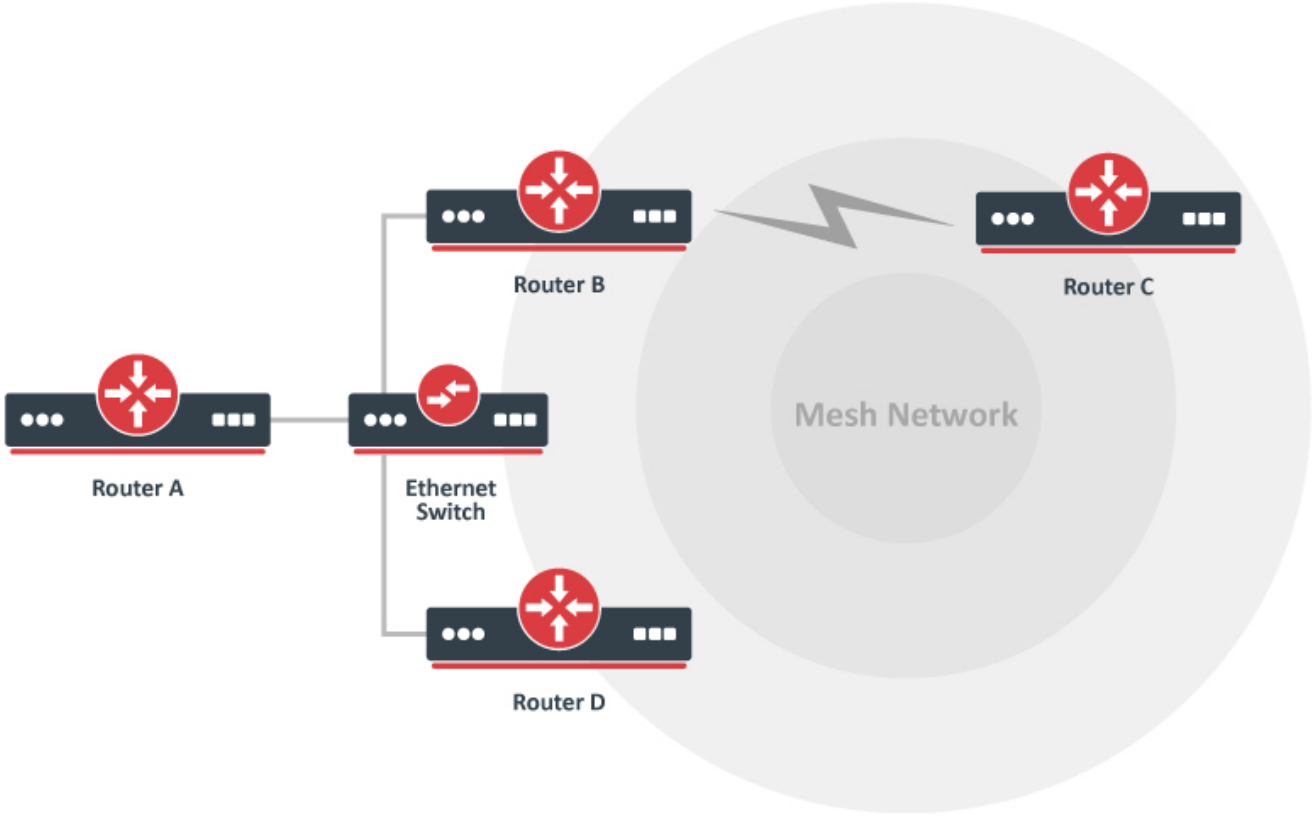
```
interface bridge settings set use-ip-firewall=yes
interface bridge filter add chain=input action=log mac-protocol=0x9aaa
interface bridge filter add chain=input action=log mac-protocol=0x9aab
```

⚠ It is perfectly possible to create mixed mesh/bridge setups that will not work (e.g. *Problematic example 1* with bridge instead of a switch). The recommended fail-safe way that will always work is to create a separate bridge interface per each of the physical interfaces; then add all these bridge interfaces as mesh ports.

Advanced topics

We all know that it's easy to make problematic layer-2 bridging or routing setups and it can be hard to debug them. (Compared to layer-3 routing setups.) So here are a few bad configuration examples that could create problems for you. Avoid them!

Problematic example 1: Ethernet switch inside a mesh



Router A is outside the mesh, all the rest of the routers are inside. For routers B, C, D all interfaces are added as mesh ports.

Router A will not be able to communicate reliably with router C. The problem manifests itself when D is the designated router for Ethernet; if B takes this role, everything is OK. The main cause of the problem is MAC address learning on Ethernet switch.

Consider what happens when router A wants to send something to C. We suppose router A either knows or floods data to all interfaces. Either way, data arrives at the switch. The switch, not knowing anything about the destination's MAC address, forwards the data to both B and D.

What happens now:

1. B receives the packet on a mesh interface. Since the MAC address is not local for B and B knows that he is not the designated router for the Ethernet network, he simply ignores the packet.
2. D receives the packet on a mesh interface. Since the MAC address is not local for B and D is the designated router for the Ethernet network, he initiates the path discovery process to C.

After path discovery is completed, D has information that C is reachable over B. Now D encapsulates the packet and forwards it back to the Ethernet network. The encapsulated packet is forwarded by the switch, received and forwarded by B, and received by C. So far everything is good.

Now C is likely to respond to the packet. Since B already knows where A is, he will decapsulate and forward the reply packet. But now the switch will learn that the MAC address of C is reachable through B! That means, next time when something arrives from A addressed to C, the switch will forward data *only* t o B (and B, of course, will silently ignore the packet)!

In contrast, if B took up the role of a designated router, everything would be OK, because traffic would not have to go through the Ethernet switch twice.

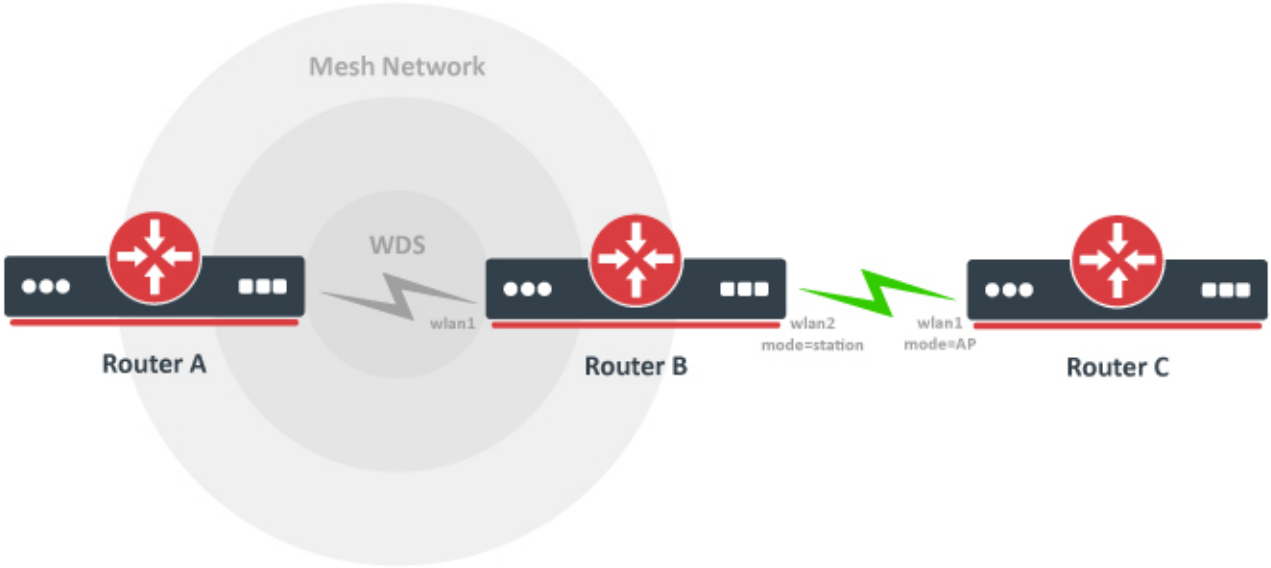
Troubleshooting: either avoid such setup or disable MAC address learning on the switch. Note that on many switches that is not possible.

Also note that there will be no problem, if either:

- router A supports and is configured to use HWMP+;
- or Ethernet switch is replaced with a router that supports HWMP+ and has Ethernet interfaces added as mesh ports.

Problematic example 2: wireless modes

Consider this (invalid) setup example:



Routers A and B are inside the mesh, router C: outside. For routers A and B all interfaces are added as mesh ports.

It is not possible to bridge wlan1 and wlan2 on router B now. The reason for this is pretty obvious if you understand how WDS works. For WDS communications four address frames are used. This is because for wireless multihop forwarding you need to know both the addresses of the intermediate hops, as well as the original sender and final receiver. In contrast, non-WDS 802.11 communication includes only three MAC addresses in a frame. That's why it's not possible to do multi-hop forwarding in station mode.

Troubleshooting: depends on what you want to achieve:

1. If you want router C to act as a repeater either for wireless or Ethernet traffic, configure the WDS link between router B and router C, and run mesh routing protocol on all nodes.
2. In other cases configure wlan2 on router B in AP mode and WLAN on router C in station mode.

Nv2

Overview

Nv2 protocol is a proprietary wireless protocol developed by MikroTik for use with Atheros 802.11 wireless chips. Nv2 is based on TDMA (Time Division Multiple Access) media access technology instead of CSMA (Carrier Sense Multiple Access) media access technology used in regular 802.11 devices.

TDMA media access technology solves hidden node problem and improves media usage, thus improving throughput and latency, especially in PtMP networks.

Nv2 is supported for Atheros 802.11n chips and legacy 802.11a/b/g chips starting from AR5212, but not supported on older AR5211 and AR5210 chips. This means that both - 11n and legacy devices can participate in the same network and it is not required to upgrade hardware to implement Nv2 in network.

Media access in Nv2 network is controlled by Nv2 Access Point. Nv2 AP divides time into fixed size "periods" which are dynamically divided into downlink (data sent from AP to clients) and uplink (data sent from clients to AP) portions, based on the queue state on AP and clients. Uplink time is further divided between connected clients based on their requirements for bandwidth. At the beginning of each period, AP broadcasts a schedule that tells clients when they should transmit and the amount of time they can use.

In order to allow new clients to connect, Nv2 AP periodically assigns uplink time for "unspecified" client - this time interval is then used by a fresh client to initiate registration to AP. Then AP estimates propagation delay between AP and client and starts periodically scheduling uplink time for this client in order to complete registration and receive data from client.

Nv2 implements dynamic rate selection on a per-client basis and ARQ for data transmissions. This enables reliable communications across Nv2 links.

For QoS Nv2 implements variable number of priority queues with built-in default QoS scheduler that can be accompanied with fine-grained QoS policy based on firewall rules or priority information propagated across network using VLAN priority or MPLS EXP bits.

Nv2 protocol limit is 511 clients per interface.

Nv2 protocol implementation status

Nv2 has the following features:

- TDMA media access
- WDS support
- QoS support with variable number or priority queues
- data encryption
- RADIUS authentication features
- statistics fields
- Fixed Downlink mode support
- Uplink/Downlink ratio support
- Nv2 AP synchronization experimental support

Compatibility and coexistence with other wireless protocols

Nv2 protocol is not compatible to or based on any other available wireless protocols or implementations, either TDMA based or any other kind. This implies that **only Nv2 supporting and enabled devices can participate in Nv2 network**.

Regular 802.11 devices will not recognize and will not be able to connect to Nv2 AP. RouterOS devices that have Nv2 support (that is - have RouterOS version 5.0rc1 or higher) will see Nv2 APs when issuing scan command, but will only connect to Nv2 AP if properly configured.

As Nv2 does not use CSMA technology it may disturb any other network in the same channel. In the same way other networks may disturb Nv2 network, because every other signal is considered noise.

The key points regarding compatibility and coexistence:

- only RouterOS devices will be able to participate in Nv2 network
- only RouterOS devices will see Nv2 AP when scanning
- Nv2 network will disturb other networks in the same channel
- Nv2 network may be affected by any (Nv2 or not) other networks in the same channel
- Nv2 enabled device will not connect to any other TDMA based network

How Nv2 compares with Nstreme and 802.11

Nv2 vs 802.11

The key differences between Nv2 and 802.11:

- Media access is scheduled by AP - this eliminates hidden node problem and allows to implement centralized media access policy - AP controls how much time is used by every client and can assign time to clients according to some policy instead of every device contending for media access.
- Reduced propagation delay overhead - There are no per-frame ACKs in Nv2 - this significantly improves throughput, especially on long-distance links where data frame and following ACK frame propagation delay significantly reduces the effectiveness of media usage.
- Reduced per frame overhead - Nv2 implements frame aggregation and fragmentation to maximize assigned media usage and reduce per-frame overhead (interframe spaces, preambles).

Nv2 vs Nstreme

The key differences between Nv2 and Nstreme:

- Reduced polling overhead - instead of polling each client, Nv2 AP broadcasts an uplink schedule that assigns time to multiple clients, this can be considered "group polling" - no time is wasted for polling each client individually, leaving more time for actual data transmission. This improves throughput, especially in PtMP configurations.
- Reduced propagation delay overhead - Nv2 must not poll each client individually, this allows to create uplink schedule based on estimated distance (propagation delay) to clients such that media usage is most effective. This improves throughput, especially in PtMP configurations.
- More control over latency - reduced overhead, adjustable period size and QoS features allows for more control over latency in the network.

Configuring Nv2

wireless-protocol setting controls which wireless protocol selects and uses. Note that the meaning of this setting depends on the interface role (either it is AP or client) that depends on interface **mode** setting. Find possible values of **wireless-protocol** and their meaning in table below.

value	AP	client
unspecified	establish nstreme or 802.11 network based on old **nstreme** setting	connect to nstreme or 802.11 network based on old **nstreme** setting
any	same as **unspecified**	scan for all matching networks, no matter what protocol, connect using protocol of chosen network
802.11	establish 802.11 network	connect to 802.11 networks only
nstreme	establish Nstreme network	connect to Nstreme networks only
Nv2	establish Nv2 network	connect to Nv2 networks only

Nv2-nstreme-802.11	establish Nv2 network	scan for Nv2 networks, if suitable network found - connect, otherwise scan for Nstreme networks, if suitable network found - connect, otherwise scan for 802.11 network and if suitable network found - connect.
Nv2-nstreme	establish Nv2 network	scan for Nv2 networks, if suitable network found - connect, otherwise scan for Nstreme networks and if suitable network found - connect

Note that **wireless-protocol** values **Nv2-nstreme-802.11** and **Nv2-nstreme DO NOT** specify some hybrid or special kind of protocol - these values are implemented to simplify client configuration when protocol of network that client must connect to can change. Using these values can help in migrating network to Nv2 protocol.

Most of Nv2 settings are significant only to Nv2 AP - Nv2 client automatically adapts necessary settings from AP. The following settings are relevant to Nv2 AP:

- **Nv2-queue-count** - specifies how many priority queues are used in Nv2 network. For more details see QoS in Nv2 network
- **Nv2-qos** - controls frame to priority queue mapping policy. For more details see QoS in Nv2 network
- **Nv2-cell-radius** - specifies distance to farthest client in Nv2 network in km. This setting affects the size of contention time slot that AP allocates for clients to initiate connection and also size of time slots used for estimating distance to client. If this setting is too small, clients that are farther away may have trouble connecting and/or disconnect with "ranging timeout" error. Although during normal operation the effect of this setting should be negligible, in order to maintain maximum performance, it is advised to not increase this setting if not necessary, so AP is not reserving time that is actually never used, but instead allocates it for actual data transfer.
- **tdma-period-size** - specifies size in ms of time periods that Nv2 AP uses for media access scheduling. Smaller period can potentially decrease latency (because AP can assign time for client sooner), but will increase protocol overhead and therefore decrease throughput. On the other hand - increasing period will increase throughput but also increase latency. It may be required to increase this value for especially long links to get acceptable throughput. This necessity can be caused by the fact that there is "propagation gap" between downlink (from AP to clients) and uplink (from clients to AP) data during which no data transfer is happening. This gap is necessary because client must receive last frame from AP - this happens after propagation delay after AP's transmission, and only then client can transmit - as a result frame from client arrives at AP after propagation delay after client's transmission (so the gap is propagation delay times two). The longer the distance, the bigger is necessary propagation gap in every period. If propagation gap takes significant portion of period, actual throughput may become unacceptable and period size should get increased at the expense of increased latency. Basically value of this setting must be carefully chosen to maximize throughput but also to keep latency at acceptable levels.
- **Nv2-mode** - specifies to use dynamic or fixed downlink/uplink ratio. Default value is "dynamic-downlink";

"sync-master" - works as nv2-mode=fixed-downlink (so uses nv2-downlink-ratio), but allows slaves to sync to this master; "sync-slave" - tries to sync to master (or already synced slave) and adapt period-size and downlink ratio settings from master.

- **Nv2-downlink-ratio** - specifies the Nv2 downlink ratio. Uplink ratio is automatically calculated from the downlink-ratio value. When using dynamic-downlink mode the downlink-ratio is also used when link get fully saturated. Minimum value is 20 and maximum 80. Default value is 50.

The follwing settings are significant on both - Nv2 AP and Nv2 client:

- **Nv2-security** - specifies Nv2 security mode, for more details see Security in Nv2 network
- **Nv2-preshared-key** - specifies preshared key to be used, for more details see Security in Nv2 network
- **nv2-sync-secret** - specifies secret key for use in the Nv2 synchronization. Secret should match on Master and Slave devices in order to establish the synced state.

Migrating to Nv2

Using **wireless-protocol** setting aids in migration or evaluating Nv2 protocol in existing networks really simple and reduce downtime as much as possible. These are the recommended steps:

- upgrade AP to version that supports Nv2, but do not enable Nv2 on AP yet.
- upgrade clients to version that supports Nv2
- configure all clients with **wireless-protocol=Nv2-nstreme-802.11**. Clients will still connect to AP using protocol that was used previously, because AP is not changed over to Nv2 yet
- configure Nv2 related settings on AP
- if it is necessary to use data encryption and secure authentication, configure Nv2 security related settings on AP and clients (refer to Security in Nv2 network).
- set **wireless-protocol=Nv2** on AP. This will make AP to change to Nv2 protocol. Clients should now connect using Nv2 protocol.
- in case of some trouble you can easily switch back to previous protocol by simply changing it back to whatever was used before on AP.
- fine tune Nv2 related settings to get acceptable latency and throughput
- implement QoS policy for maximum performance.

The basic troubleshooting guide:

- clients have trouble connecting or disconnect with "ranging timeout" error - check that **Nv2-cell-radius** setting is set appropriately
- unexpectedly low throughput on long distance links although signal and rate is fine - try to increase **tdma-period-size** setting

Nv2 AP Synchronization

This feature will let multiple MikroTik Nv2 APs on the same location to coexist in a better fashion by reducing the interference between each other. This feature will synchronize the transmit/receive time windows of APs in the same frequency, so that all synced MikroTik Nv2 APs transmits/receives at the same time. That allows to reuse the same wireless frequency on the location for multiple APs giving more flexibility in frequency planning.

To make Nv2 synced setup:

- For Nv2 Synchronization a Master Nv2 AP should be chosen and "nv2-mode=sync-master" should be specified together with "nv2-sync-secret".
- For Nv2 Slave APs the same wireless frequency as Master AP should be used and "nv2-mode=sync-slave" should be specified with the same "nv2-sync-secret" as the in Master AP configuration.
- When Master AP is enabled Slave APs will try start searching for Master AP by matching it against specified "nv2-sync-secret".
- After Master AP is found the Slave AP will calculate the distance to the Master AP as it is possible that Master AP is located not on the same location.
- Then Slave AP starts operating as AP and it adapts the period size and downlink ratio from the synced Master AP.
- In addition after the Slave AP is operational other Slave APs can use this Slave AP to sync with.
- Slave AP periodically listens for the Master AP and checks if the "nv2-sync-secret" still matches and adapts the parameters again. If Master AP interface is disabled/enabled all the Slaves will be also disabled and will start the synchronization process from the beginning.
- If Master AP stops working Slave APs also will stop working as they do not have sync information.

Configuration example

Master AP:

```
 /interface wireless set wlan1 mode=ap-bridge ssid=Sector1 frequency=5220 nv2-mode=sync-master nv2-preshared-
key=clients1 nv2-sync-secret=Tower1
```

Slave AP:

```
 /interface wireless set wlan1 mode=ap-bridge ssid=Sector2 frequency=5220 nv2-mode=sync-slave nv2-preshared-
key=clients2 nv2-sync-secret=Tower1
```

Monitor interface on the Slave AP:

```
[admin@SlaveAP] /interface wireless> monitor wlan1
                  status: running-ap
                 channel: 5220/20/an
       wireless-protocol: nv2
             noise-floor: -110dBm
      registered-clients: 1
    authenticated-clients: 1
          nv2-sync-state: synced
         nv2-sync-master: 4C:5E:0C:57:84:38
       nv2-sync-distance: 1
    nv2-sync-period-size: 2
  nv2-sync-downlink-ratio: 50
```

Debug logs on the Master AP:

```
 09:22:08 wireless,debug wlan1: 4C:5E:0C:57:85:BE attempts to sync
```

Debug logs on the Slave AP:

```
09:22:08 wireless,debug wlan1: attempting to sync to 4C:5E:0C:57:84:38
09:22:09 wireless,debug wlan1: synced to 4C:5E:0C:57:84:38
```

QoS in Nv2 network

QoS in Nv2 is implemented by means of variable number of priority queues. Queue is considered for transmission based on rule recommended by 802.1D-2004 - only if all higher priority queues are empty. In practice this means that at first all frames from queue with higher priority will be sent, and only then next queue is considered. Therefore QoS policy must be designed with care so that higher priority queues do not make lower priority queues starve.

QoS policy in Nv2 network is controlled by AP, clients adapt policy from AP. On AP QoS policy is configured with **Nv2-queue-count** and **Nv2-qos** parameters. **Nv2-queue-count** parameter specifies number of priority queues used. Mapping of frames to queues is controlled by **Nv2-qos** parameter.

Nv2-qos=default

In this mode outgoing frame at first is inspected by built-in QoS policy algorithm that selects queue based on packet type and size. If built-in rules do not match, queue is selected based on frame priority field, as in **Nv2-qos=frame-priority** mode.

Nv2-qos=frame-priority

In this mode QoS queue is selected based on frame priority field. Note that frame priority field is not some field in headers and therefore it is valid only while packet is processed by given device. Frame priority field must be set either explicitly by firewall rules or implicitly from ingress priority by frame forwarding process, for example, from MPLS EXP bits. For more information on frame priority field see:

- EXP bit behaviour
- WMM and VLAN priority

Queue is selected based on frame priority according to 802.1D recommended user priority to traffic class mapping. Mapping depends on number of available queues (**Nv2-queue-count** parameter). For example, if number of queues is 4, mapping is as follows (pay attention how this mapping resembles mapping used by WMM):

- priority 0,3 -> queue 0
- priority 1,2 -> queue 1
- priority 4,5 -> queue 2
- priority 6,7 -> queue 3

If number of queues is 2 (default), mapping is as follows:

- priority 0,1,2,3 -> queue 0
- priority 4,5,6,7 -> queue 1

If number of queues is 8 (maximum possible), mapping is as follows:

- priority 1 -> queue 0
- priority 2 -> queue 1
- priority 0 -> queue 2
- priority 3 -> queue 3
- priority 4 -> queue 4
- priority 5 -> queue 5
- priority 6 -> queue 6
- priority 7 -> queue 7

For other mappings, discussion on rationale for these mappings and recommended practices please see 802.1D-2004.

Security in Nv2 network

Nv2 security implementation has the following features:

- hardware accelerated data encryption using AES-CCM with 128 bit keys;
- 4-way handshake for key management (similar to that of 802.11i);
- preshared key authentication method (similar to that of 802.11i);
- periodically updated group keys (used for broadcast and multicast data).

Being proprietary protocol Nv2 does not use security mechanisms of 802.11, therefore security configuration is different. Interface using Nv2 protocol ignores **security-profile** setting. Instead, security is configured by the following interface settings:

- **Nv2-security** - this setting enables/disables use of security in Nv2 network. Note that when security is enabled on AP, it will not accept clients with disabled security. In the same way clients with enabled security will not connect to unsecure APs.

- **Nv2-preshared-key** - preshared key to use for authentication. Data encryption keys are derived from preshared key during 4-way handshake. Preshared key must be the same in order for 2 devices to establish connection. If preshared key will differ, connection will time out because remote party will not be able to correctly interpret key exchange messages.

Interworking Profiles

Interworking

Interworking is the occurrence of two or more things working together. For a better Wireless network experience information about the network must be exchanged between Access Points and Wireless client devices, the information that can be found in basic Wireless beacons and probe requests is limited. For this reason, the IEEE 802.11u™-2011 (Interworking with External Networks) standard was created, that specifies how devices should exchange information between each other. Network discovery and Access Point selection process can be enhanced with the interworking service. Wireless client devices can have more criteria upon which they can choose the network with which to associate.

Hotspot 2.0

Hotspot 2.0 is a specification developed and owned by the Wi-Fi Alliance. It was designed to enable a more cellular-like experience when connecting to Wi-Fi networks. In the attempt to increase Wireless network security Hotspot 2.0 access points use mandatory WPA2 authentication. Hotspot 2.0 relies on Interworking as well as adds some of its own properties and procedures.

Interworking profiles are implemented according to IEEE 802.11u and Hotspot 2.0 Release 1 specifications.

> This manual page describes the configuration of the regular wireless package, the same parameters are available in the WifiWave2 package as well.

Configuration Properties

Sub-menu: `/interface wireless interworking-profiles`

Information elements in beacon and probe response

Some information can be added to beacon and probe response packets with a Interworking element. Following parameters of a Interworking element can be configured:

Property	Description
asra (*yes* | *no*; Default: **no**)	Additional Steps Required for Access. Set to `yes`, if a user should take additional steps to access the internet, like the walled garden.
esr (*yes* | *no*; Default: **no**)	Emergency services reachable (ESR). Set to `yes` in order to indicate that emergency services are reachable through the access point.
hessid (*MAC address*; Default:)	Homogenous extended service set identifier (HESSID). Devices that provide access to same external networks are in one homogenous extended service set. This service set can be identified by HESSID that is the same on all access points in this set. 6-byte value of HESSID is represented as MAC address. It should be globally unique, therefore it is advised to use one of the MAC address of access point in the service set.

internet (*yes* / *no*; Default: **yes**)	Whether the internet is available through this connection or not. This information is included in the Interworking element.
network-type (*emergency-only* / *personal-device* / *private* / *private-with-guest* / *public-chargeable* / *public-free* / *test* / *wildcard*; Default: **wildcard**)	Information about network access type. • `emergency-only` - a network dedicated and limited to accessing emergency services; • `personal-device` - a network of personal devices. An example of this type of network is a camera that is attached to a printer, thereby forming a network for the purpose of printing pictures; • `private` - network for users with user accounts. Usually used in enterprises for employees, not guests; • `private-with-guest` - same as private, but guest accounts are available; • `public-chargeable` - a network that is available to anyone willing to pay. For example, a subscription to Hotspot 2.0 service or in-room internet access in a hotel; • `public-free` - network is available to anyone without any fee. For example, municipal network in city or airport Hotspot; • `test` - network used for testing and experimental uses. Not used in production; • `wildcard` - is used on Wireless clients. Sending probe request with a wildcard as network type value will make all Interworking Access Points respond despite their actual network-type setting. A client sends a probe request frame with network-type set to value it is interested in. It will receive replies only from access points with the same value (except the case of wildcard).
uesa (*yes* / *no*; Default: **no**)	Unauthenticated emergency service accessible (UESA). • `no` - indicates that no unauthenticated emergency services are reachable through this Access Point; • `yes` - indicates that higher layer unauthenticated emergency services are reachable through this Access Point.

venue (*venue*; Default: **unspecified**)	Specify the venue in which the Access Point is located. Choose the value from available ones. Some examples:
	`venue=business-bank`
	`venue=mercantile-shopping-mall`
	`venue=educational-university-or-college`

ANQP elements

Access network query protocol (ANQP). Not all necessary information is included in probe response and beacon frames. For client device to get more information before choosing access point to associate with ANQP is used. The Access Point can have stored information in multiple ANQP elements. Client device will use ANQP to query only for the information it is interested in. This reduces the time needed before association.

Property	Description
3gpp-raw (*octet string in hex*; Default:)	Cellular network advertisement information - country and network codes. This helps Hotspot 2.0 clients in the selection of an Access Point to access 3GPP network. Please see 3GPP TS 24.302. (Annex H) for a format of this field. This value is sent ANQP response if queried.
3gpp-info (*number/number*; Default:)	Cellular network advertisement information - country and network codes. This helps Hotspot 2.0 clients in the selection of an Access Point to access 3GPP network. Written as "mcc /mnc". Usage is identical to "3gpp-raw", but without using hex. Multiple mcc/mnc pairs can be defined, by separating them with a comma.
authentication-types (*dns-redirection:`url` / https-redirection:`url` / online-enrollment:`url` / terms-and-conditions:`url`*; Default:)	This property is only effective when asra is set to `yes`. Value of `url` is optional and not needed if `dns-redirection` or `online-enrollment` is selected. To set the value of `url` to empty string use double quotes. For example: `authentication-types=online-enrollment:""`

connection-capabilities (*number:number:closed|open|unknown*; Default:)	This option allows to provide information about the allowed IP protocols and ports. This information can be provided in ANQP response. The first number represents the IP protocol number, the second number represents a port number. • `closed` - set if protocol and port combination is not allowed; • `open` - set if protocol and port combination is allowed; • `unknown` - set if protocol and port combination is either open or closed. Example: `connection-capabilities=6:80:open,17:5060:closed` Setting such a value on an Access Point informs the Wireless client, which is connecting to the Access Point, that HTTP (6 - TCP, 80 - HTTP) is allowed and VoIP (17 - UDP; 5060 - VoIP) is not allowed. This property does not restrict or allow usage of these protocols and ports, it only gives information to station device which is connecting to Access Point.
domain-names (*list of strings*; Default:)	None or more fully qualified domain names (FQDN) that indicate the entity operating the Hotspot. A station that is connecting to the Access Point can request this AQNP property and check if there is a suffix match with any of the domain names it has credentials to.
ipv4-availability (*double-nated | not-available | port-restricted | port-restricted-double-nated | port-restricted-single-nated | public | single-nated | unknown*; Default: **not-available**)	Information about what IPv4 address and access are available. • `not-available` - Address type not available; • `public` - public IPv4 address available; • `port-restricted` - port-restricted IPv4 address available; • `single-nated` - single NATed private IPv4 address available; • `double-nated` - double NATed private IPv4 address available; • `port-restricted-single-nated` -port-restricted IPv4 address and single NATed IPv4 address available; • `port-restricted-double-nated` - port-restricted IPv4 address and double NATed IPv4 address available; • `unknown` - availability of the address type is not known.
ipv6-availability (*available | not-available | unknown*; Default: **not-available**)	Information about what IPv6 address and access are available. • `not-available` - Address type not available; • `available` - address type available; • `unknown` - availability of the address type is not known.
realms (*string:eap-sim|eap-aka|eap-tls|not-specified*; Default:)	Information about supported realms and the corresponding EAP method. `realms=example.com:eap-tls,foo.ba:not-specified`
realms-raw (*octet string in hex*; Default:)	Set NAI Realm ANQP-element manually.
roaming-ois (*octet string in hex*; Default:)	Organization identifier (OI) usually are 24-bit is unique identifiers like organizationally unique identifier (OUI) or company identifier (CID). In some cases, OI is longer for example OUI-36. A subscription service provider (SSP) can be specified by its OI. roaming-ois property can contain zero or more SSPs OIs whose networks are accessible via this AP. Length of OI should be specified before OI itself. For example, to set E4-8D-8C and 6C-3B-6B: `roaming-ois=03E48D8C036C3B6B`
venue-names (*string:lang*; Default:)	Venue name can be used to provide additional info on the venue. It can help the client to choose a proper Access Point. Venue-names parameter consists of zero or more duple that contain Venue Name and Language Code: `venue-names=CoffeeShop:eng,TiendaDeCafe:es` The Language Code field value is a two or three-character 8 language code selected from ISO-639.

Realms raw

realms-raw - list of strings with hex values. Each string specifies contents of "NAI Realm Tuple", excluding "NAI Realm Data Field Length" field.

Each hex encoded string must consist of the following fields:

```
- NAI Realm Encoding (1 byte)
- NAI Realm Length (1 byte)
- NAI Realm (variable)
- EAP Method Count (1 byte)
- EAP Method Tuples (variable)
```

For example, value "000454657374010020d00" decodes as:

```
- NAI Realm Encoding: 0 (rfc4282)
- NAI Realm Length: 4
- NAI Realm: Test
- EAP Method Count: 1
- EAP Method Length: 2
- EAP Method Tuple: TLS, no EAP method parameters
```

Note, that setting "realms-raw=000454657374010020d00" produces the same advertisement contents as setting "realms=Test:eap-tls".

Refer to 802.11-2016, section 9.4.5.10 for full NAI Realm encoding.

Hotspot 2.0 ANQP elements

Hotspot 2.0 specification introduced some additional ANQP elements. These elements use an ANQP vendor specific element ID. Here are available properties to change these elements.

Property	Description
hotspot20 (*yes* | *no*; Default: **yes**)	Indicate Hotspot 2.0 capability of the Access Point.
hotspot20-dgaf (*yes* | *no*; Default: **yes**)	Downstream Group-Addressed Forwarding (DGAF). Sets value of DGAF bit to indicate whether multicast and broadcast frames to clients are disabled or enabled. • `yes` - multicast and broadcast frames to clients are enabled; • `no` - multicast and broadcast frames to clients are disabled. To disable multicast and broadcast frames set `multicast-helper=full`.
operational-classes (*list of numbers*; Default:)	Information about other available bands of the same ESS.
operator-names (*string:lang*; Default:)	Set operator name. Language must be specified for each operator name entry. Operator-names parameter consists of zero or more duple that contain Operator Name and Language Code: `operator-names=BestOperator:eng,MejorOperador:es` The Language Code field value is a two or three-character 8 language code selected from ISO-639.
wan-at-capacity (*yes* | *no*; Default: **no**)	Whether the Access Point or the network is at its max capacity. If set to `yes` no additional mobile devices will be permitted to associate to the AP.
wan-downlink (*number*; Default: **0**)	The downlink speed of the WAN connection set in kbps. If the downlink speed is not known, set to 0.
wan-downlink-load (*number*; Default: **0**)	The downlink load of the WAN connection measured over `wan-measurement-duration`. Values from 0 to 255. • `0` - unknown; • `255` - 100%.
wan-measurement-duration (*number*; Default: **0**)	Duration during which wan-downlink-load and `wan-uplink-load` are measured. Value is a numeric value from 0 to 65535 representing tenths of seconds. • `0` - not measured; • `10` - 1 second; • `65535` - 1 hour 49 minutes or more.

wan-status (*down* / *reserved* / *test* / *up*; Default: **reserved**)	Information about the status of the Access Point's WAN connection. The value `reserved` is not used.
wan-symmetric (*yes* / *no*; Default: **no**)	Weather the WAN link is symmetric (upload and download speeds are the same) or not.
wan-uplink (*number*; Default: **0**)	The uplink speed of the WAN connection set in kbps. If the uplink speed is not known set to 0.
wan-uplink-load (*number*; Default: **0**)	The uplink load of th WAN connection measured over wan-measurement-duration. Values from 0 to 255. • `0` - unknown; • 255 - 100%.

Other Properties

Property	Description
comment (*string*; Default:)	Short description of the profile
name (*string*; Default:)	Name of the Interworking profile.

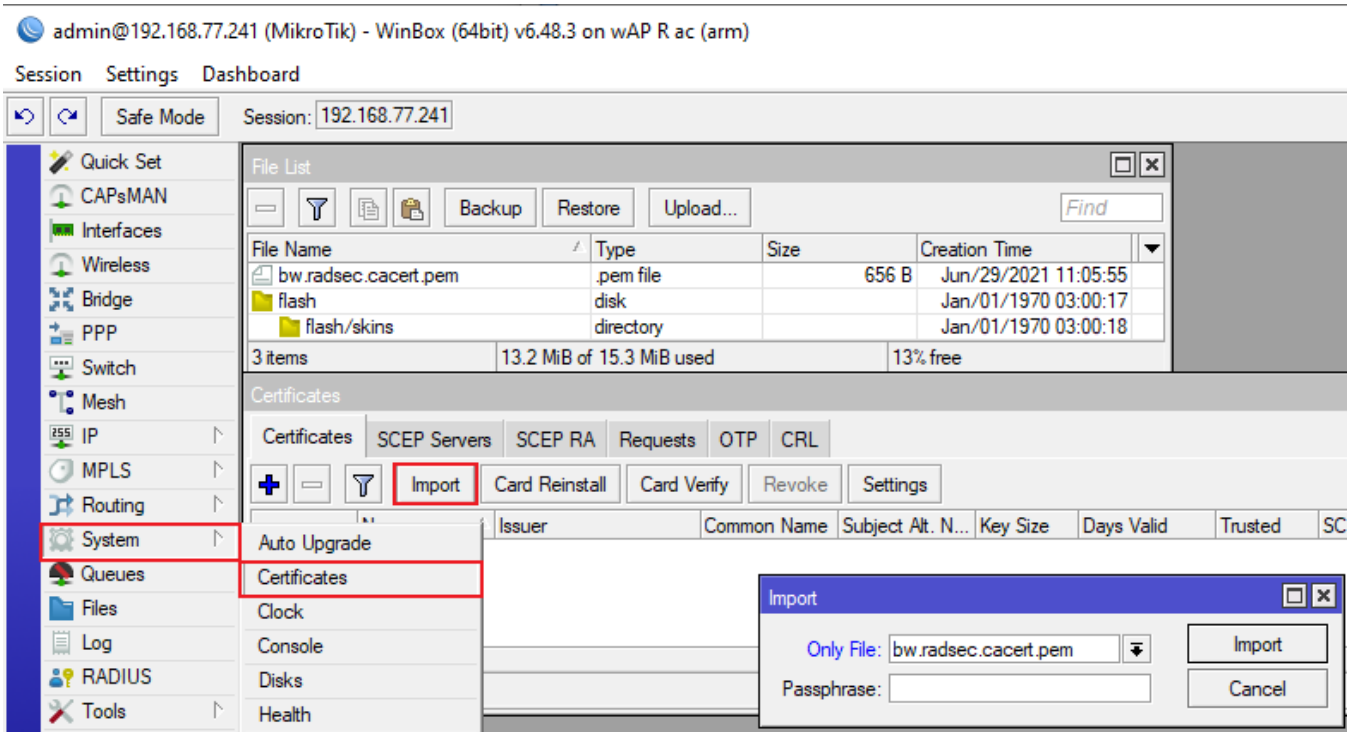
Configuration guide using native RadSec and Orion Wifi:

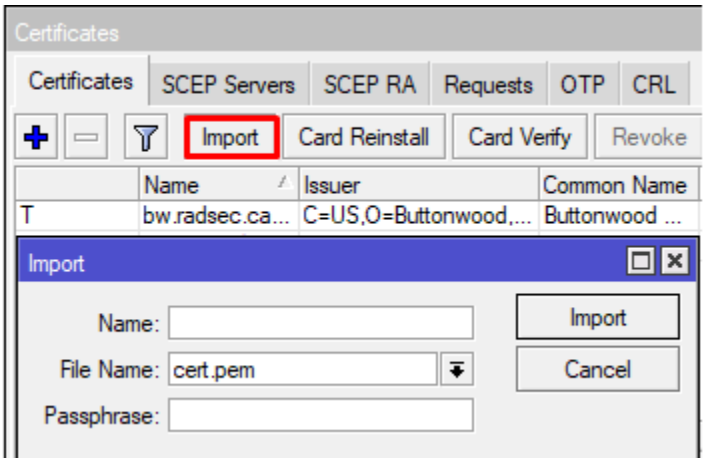
This guide describes how to set up your MikroTik devices so you can use them with RadSec proxy and Orion Wifi, though the main configuration steps remain the same and will work with different providers as well:
Make sure to use the latest long-term or stable RouterOS releases.

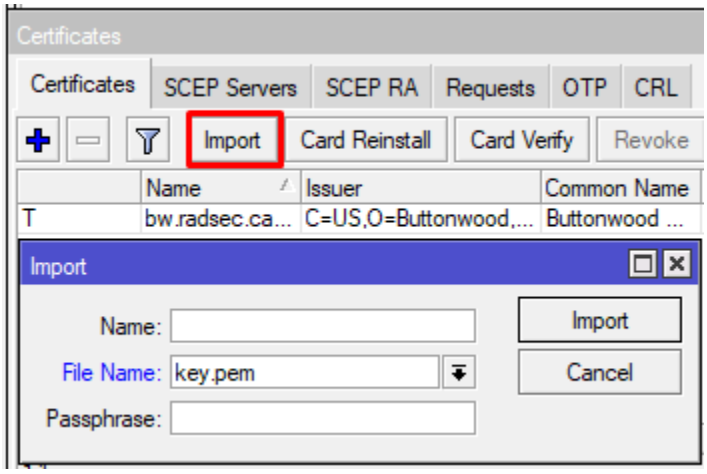
It is important to set up a secure RADIUS connection between the wireless LAN controller and Orion Wifi.
Orion Wifi uses RADIUS over TLS (RadSec) to ensure end-to-end encryption of AAA traffic.

1) Import RadSec certificates you have downloaded from the Orion:

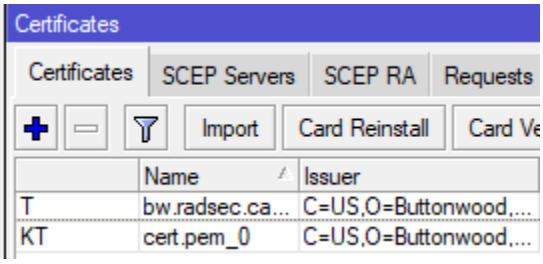
Drag and drop certificate in WinBox, and then use the import function for it, which can be found under /system certificates in WinBox, command line equivalent is :"/certificate import file-name=bw.radsec.cacert.pem passphrase="""", "/certificate import file-name=cert.pem passphrase="""", "/certificate import file-name=key.pem passphrase=""""



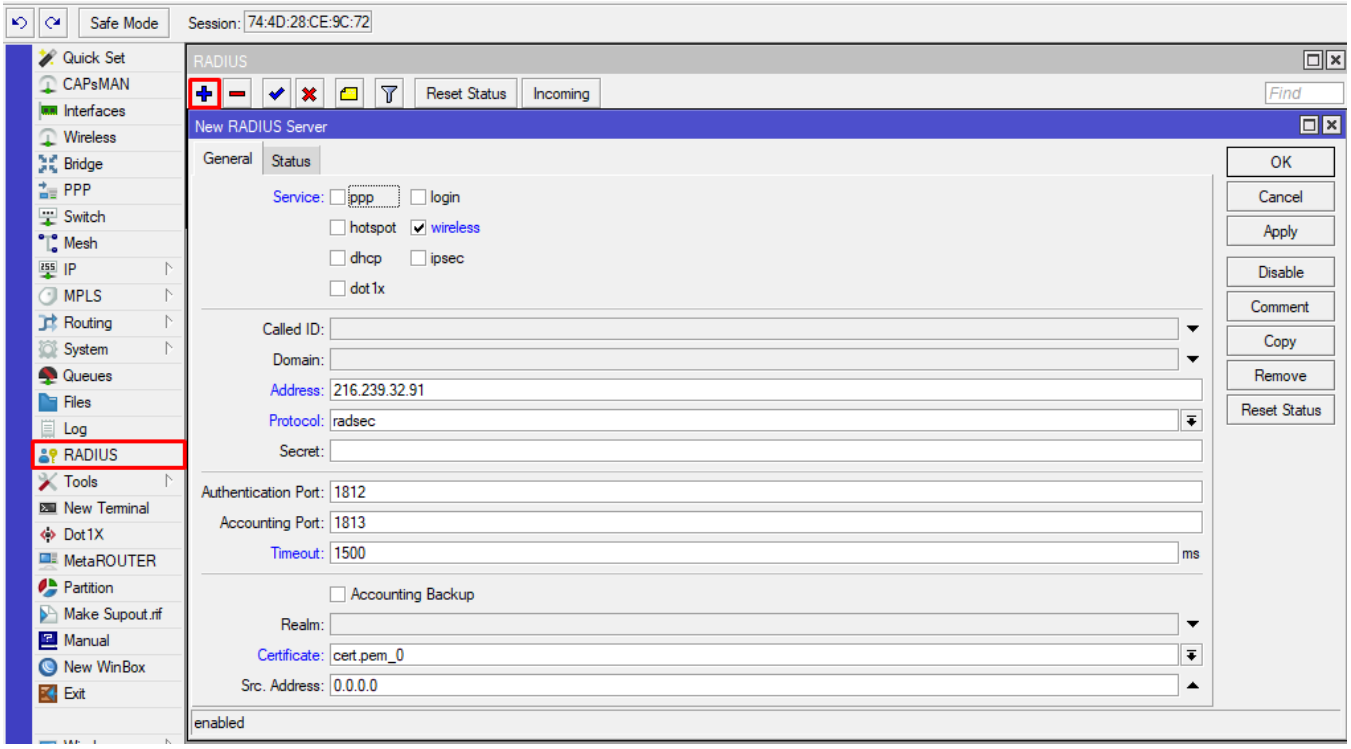




Once certificates are imported, they should look like this:

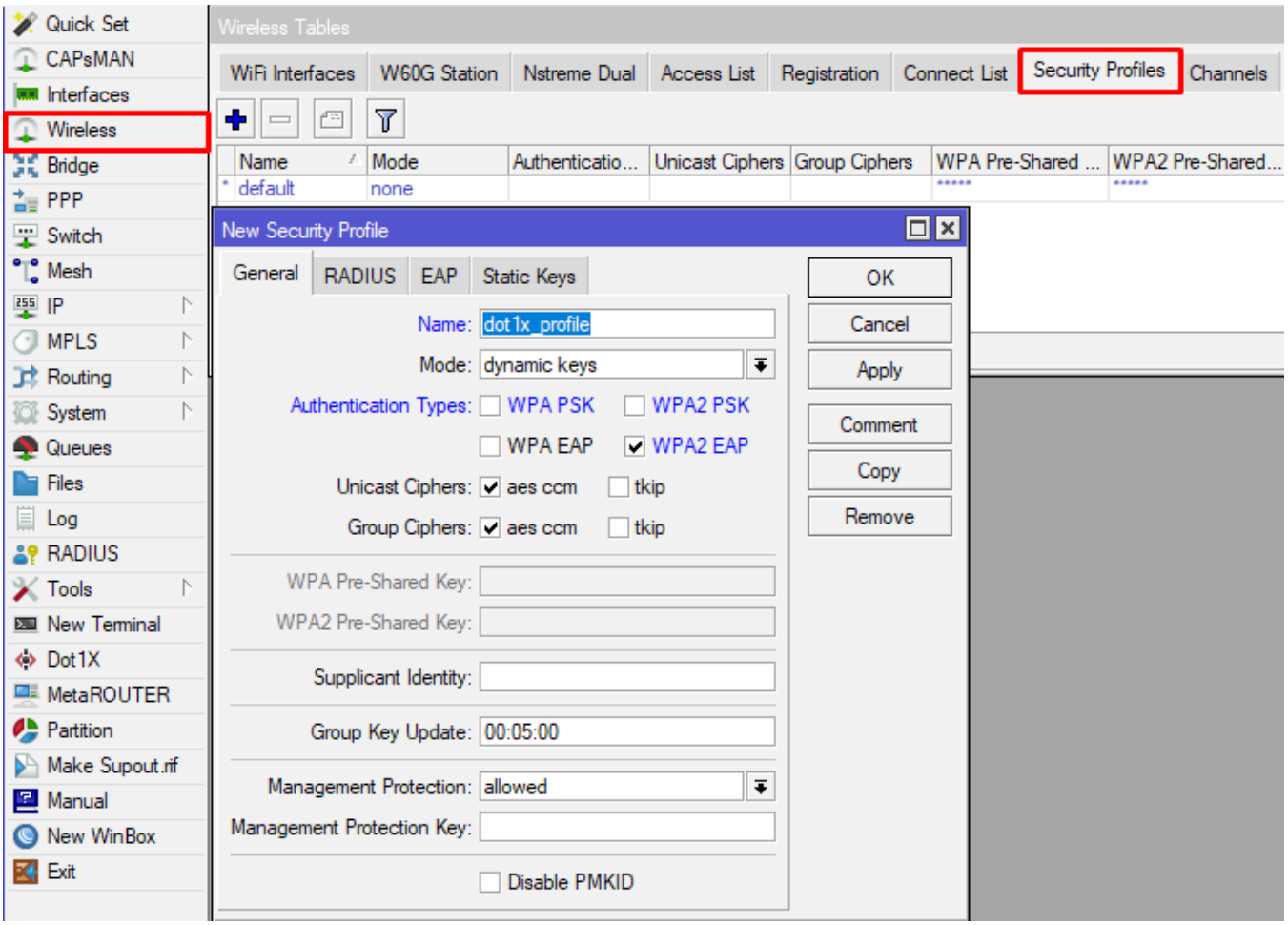


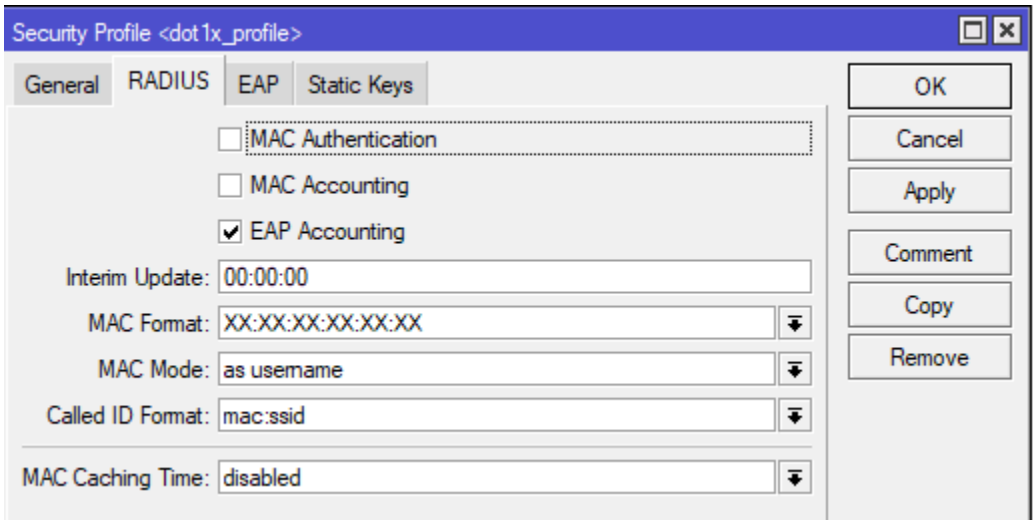
2) Configure the Radius client

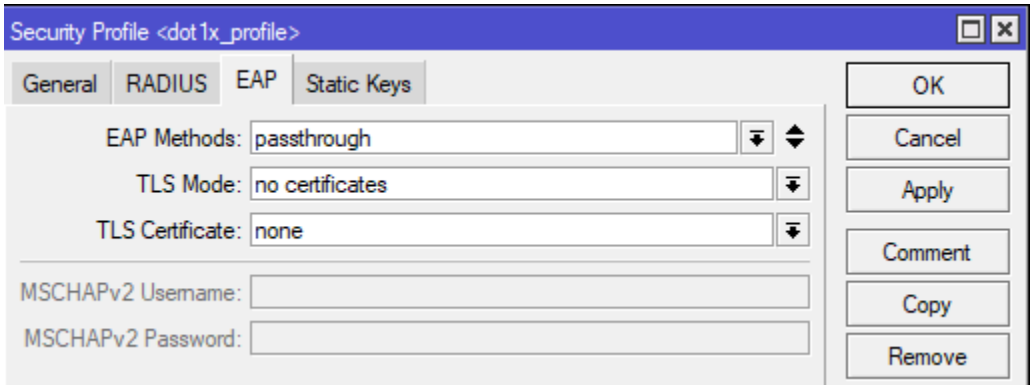


Command line equivalent: "/radius add address=216.239.32.91 certificate=cert.pem_0 protocol=radsec service=wireless timeout=1s500ms"

3) Create a wireless security profile that would perform 802.1x authentication

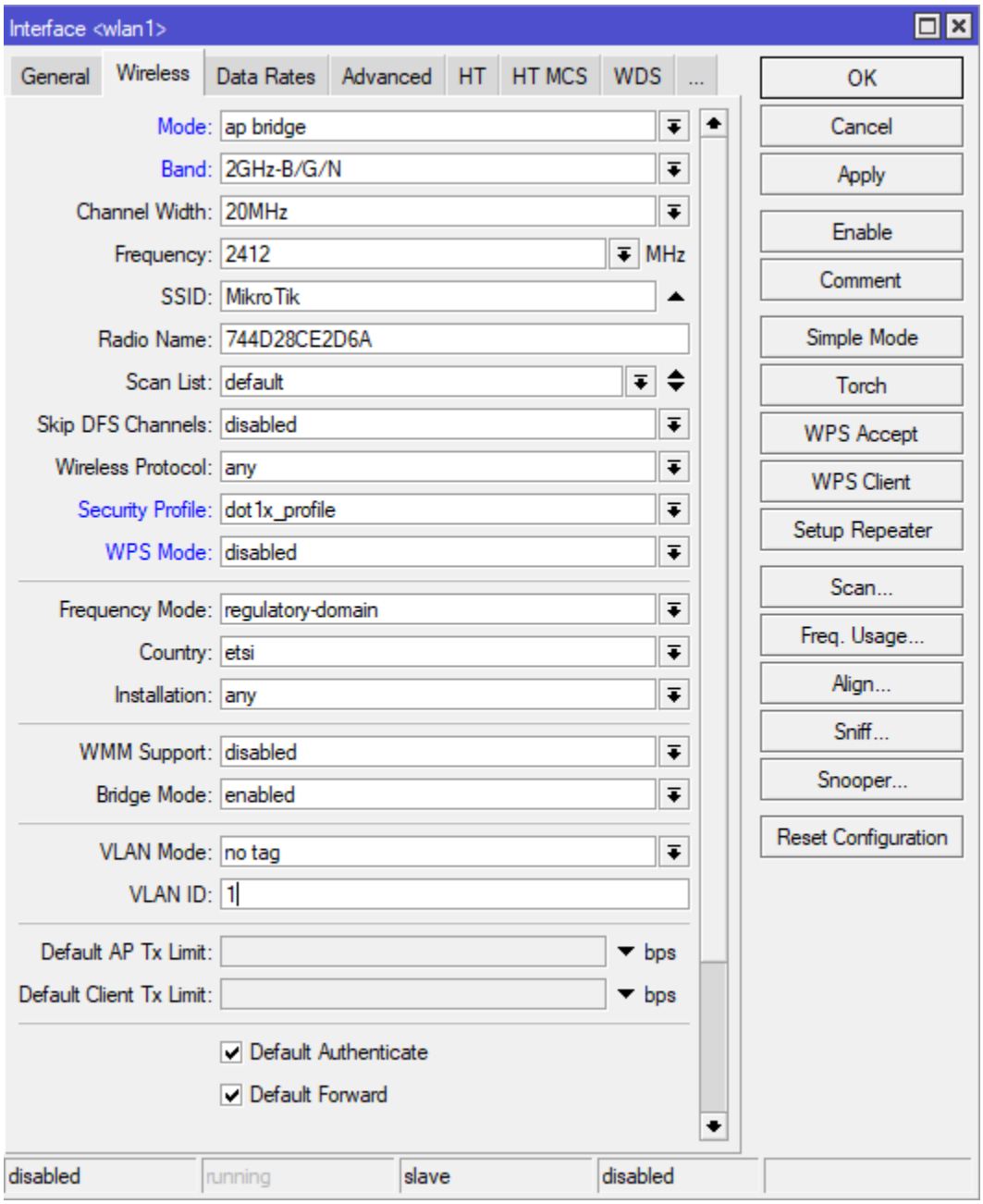






Command line equivalent is *"/interface wireless security-profiles add authentication-types=wpa2-eap management-protection=allowed mode=dynamic-keys name=dot1x_profile supplicant-identity="" radius-eap-accounting=yes eap-methods=passthrough*".

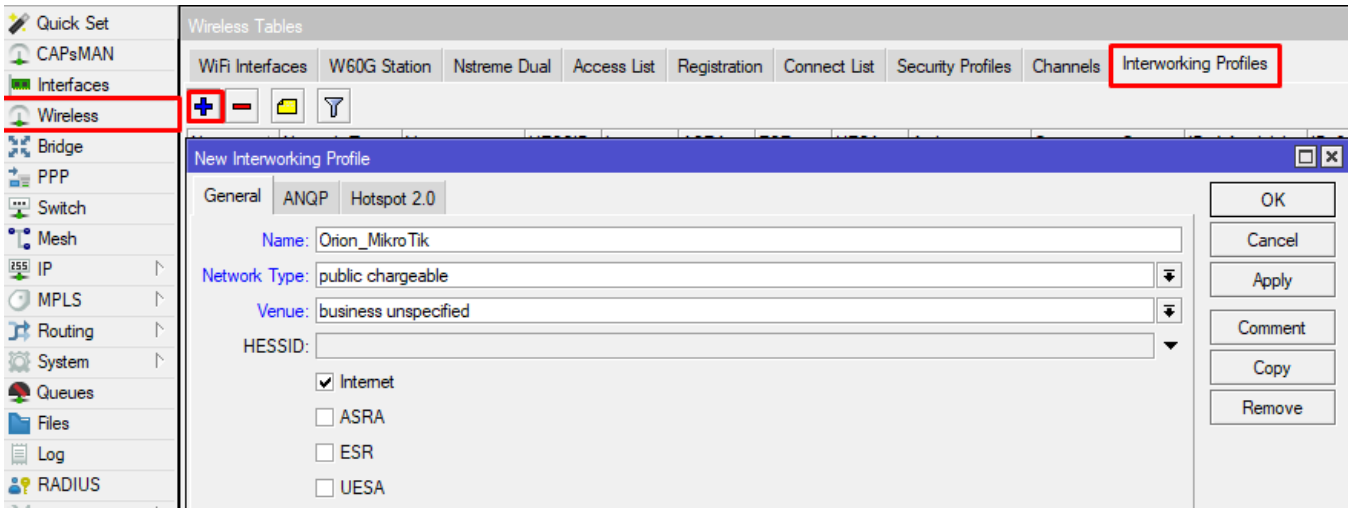
4) The next step is configuring the wireless interface and assigning the created security profile. Press "Advanced mode" to see all the options.

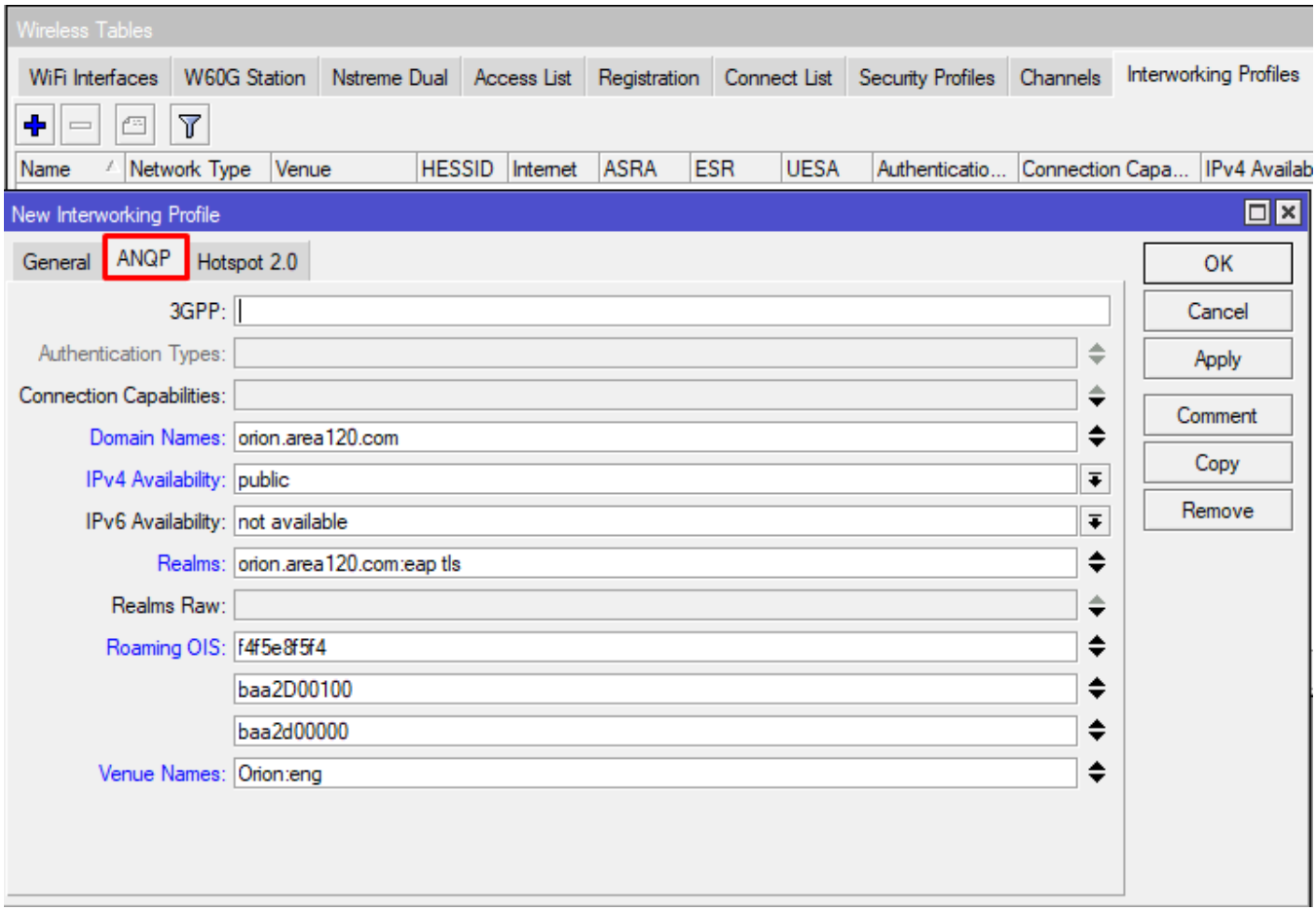


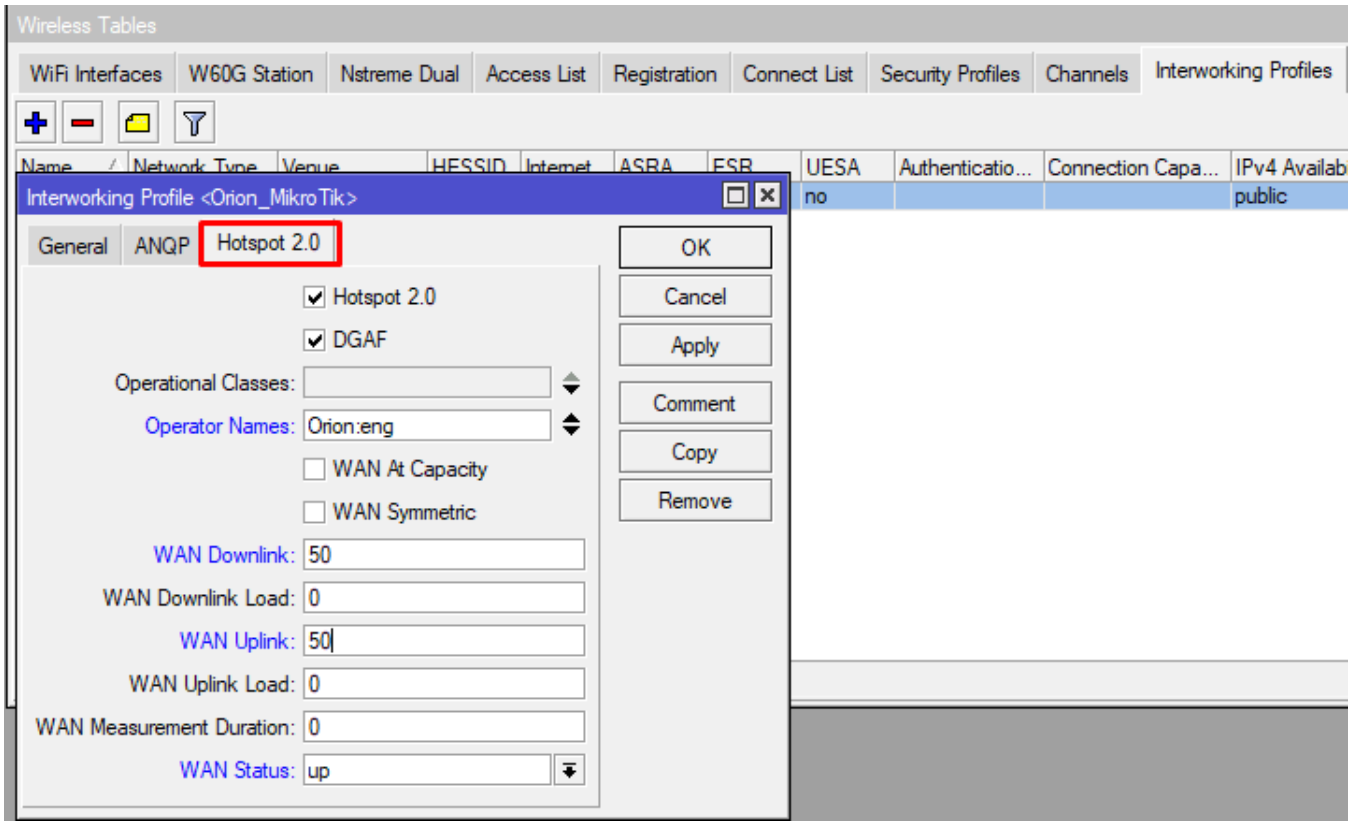
Command line equivalent is: "*/interface wireless set [find default-name=wlan1] mode=ap-bridge security-profile=dot1x_profile wps-mode=disabled*".

Make sure the correct country profile is configured. In this example, we are using “wlan1”, but the same command would work with other interfaces, or as “*/interface wireless set wlan1*”.

5) Configure interworking settings (hotspot 2.0).


Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Interworking Profiles
New Interworking Profile
General
ANQP
Hotspot 2.0
Name: Orion_MikroTik
Network Type: public chargeable
Venue: business unspecified
HESSID:
Internet
ASRA
ESR
UESA
OK
Cancel
Apply
Comment
Copy
Remove

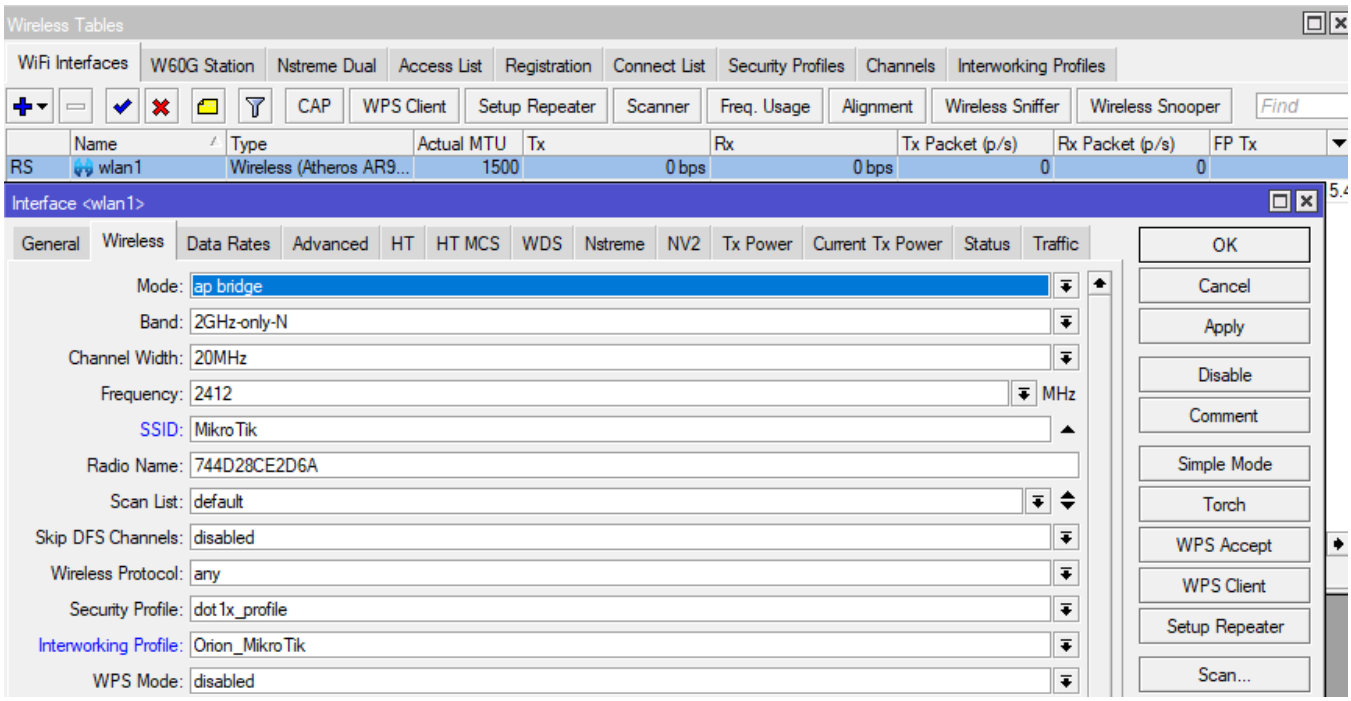

Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Interworking Profiles
Name
Network Type
Venue
HESSID
Internet
ASRA
ESR
UESA
Authenticatio...
Connection Capa...
IPv4 Availab
New Interworking Profile
General
ANQP
Hotspot 2.0
3GPP:
Authentication Types:
Connection Capabilities:
Domain Names: orion.area120.com
IPv4 Availability: public
IPv6 Availability: not available
Realms: orion.area120.com:eap tls
Realms Raw:
Roaming OIS: f4f5e8f5f4
baa2D00100
baa2d00000
Venue Names: Orion:eng
OK
Cancel
Apply
Comment
Copy
Remove



Command line equivalent: *"/interface wireless interworking-profile add domain-names=orion.area120.com ipv4-availability=public name=Orion_MikroTik network-type=public-chargeable operator-names=Orion:eng realms=orion.area120.com:eap-tls roaming-ois=f4f5e8f5f4,baa2D00100,baa2d00000 venue=business-unspecified venue-names=Orion:eng wan-downlink=50 wan-uplink=50 wan-status=up*".

> ⚠ Pay special attention to "wan-downlink" and "wan-uplink", in this scenario value of "50" is used as a placeholder, make sure to adjust the values according to your setup, some client devices use it to evaluate, if they should join the network. Set "venue" – venue type, "venue-names" and other attributes as applicable. "domain-names" should be of hotspot 2.0 Operator.

6) Assign the interworking profile to the interface.



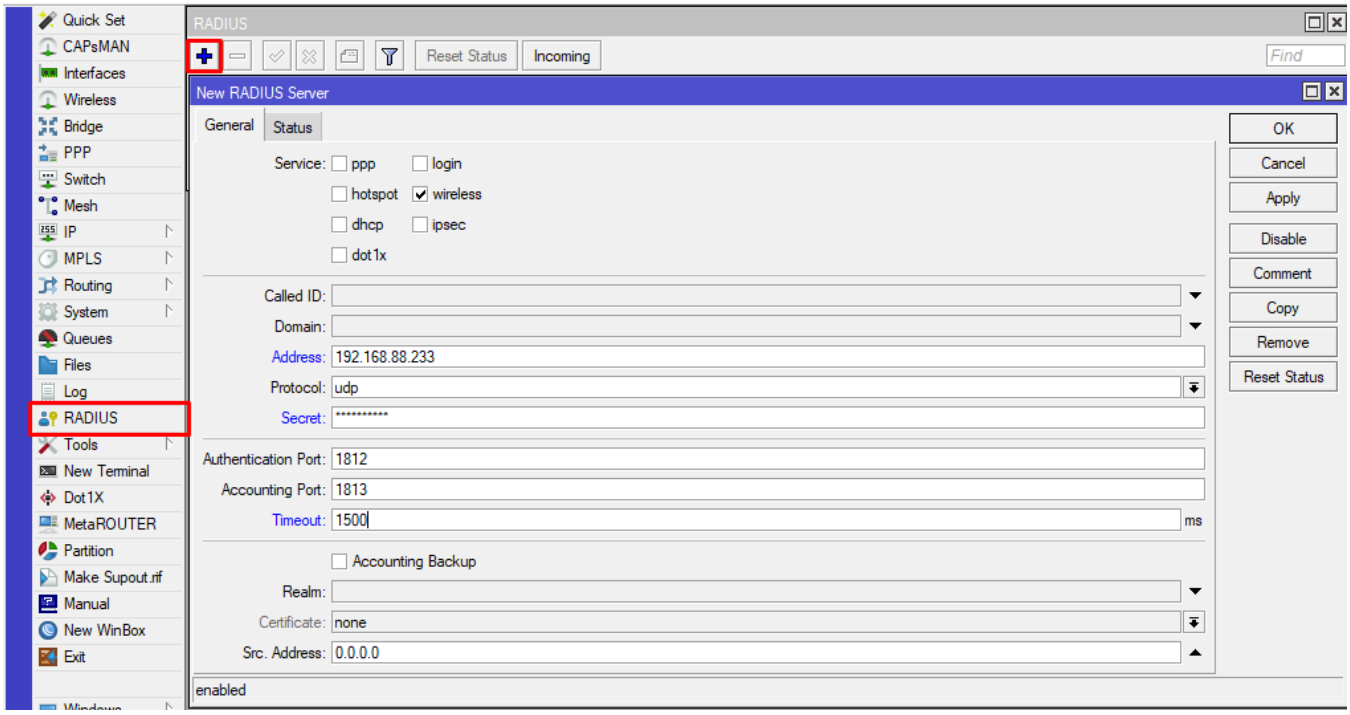
Command-line equivalent is: “/interface wireless set wlan1 interworking-profile=Orion_MikroTik”. If you don't see the interworking-profile field, press "Advanced mode".

Note: NAS-id that's used by Orion to differentiate networks is equal to system identity, to adjust the nas-id, you can do "/system identity set name=exampleName". Graphical interface support for interworking profiles are added from versions above 6.47.10, 6.48.3.

Configuration guide using RadSec proxy and Orion Wifi:

This guide describes how to set up your MikroTik devices so you can use them with RadSec proxy and Orion Wifi, though the main configuration steps remain the same and will work with different providers as well:
This guide assumes that you have configured a radsecproxy with Orion Wifi credentials. Make sure to use the latest long-term or stable RouterOS releases.

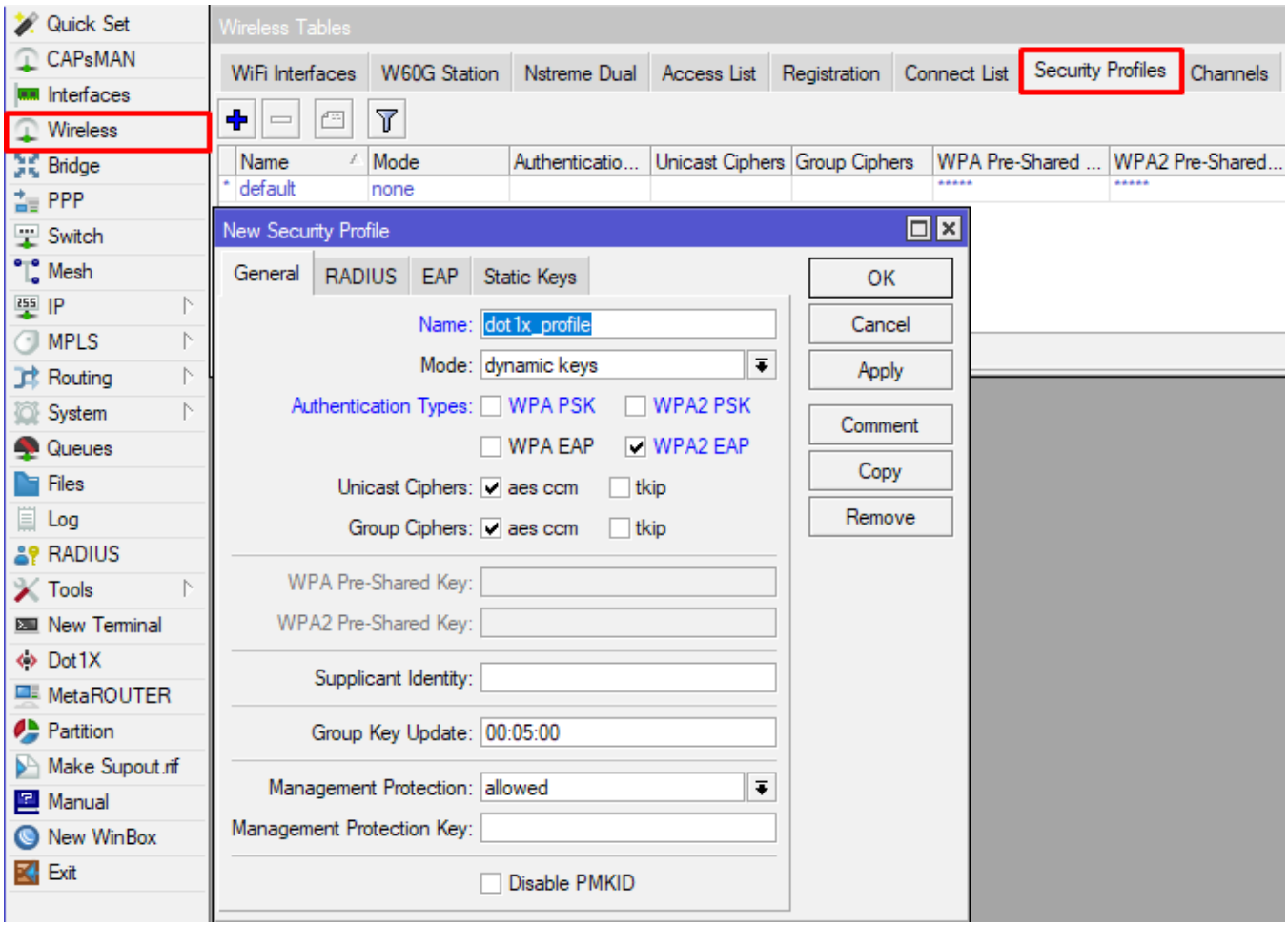
It is important to set up a secure RADIUS connection between the wireless LAN controller and Orion Wifi.
Orion Wifi uses RADIUS over TLS (RadSec) to ensure end-to-end encryption of AAA traffic. This guide is made for scenarios where the RouterOS access point redirects AAA traffic to a RadSec proxy (radsecproxy) before the traffic is sent over the internet.
1) Configure the Radius client that points to radsecproxy.

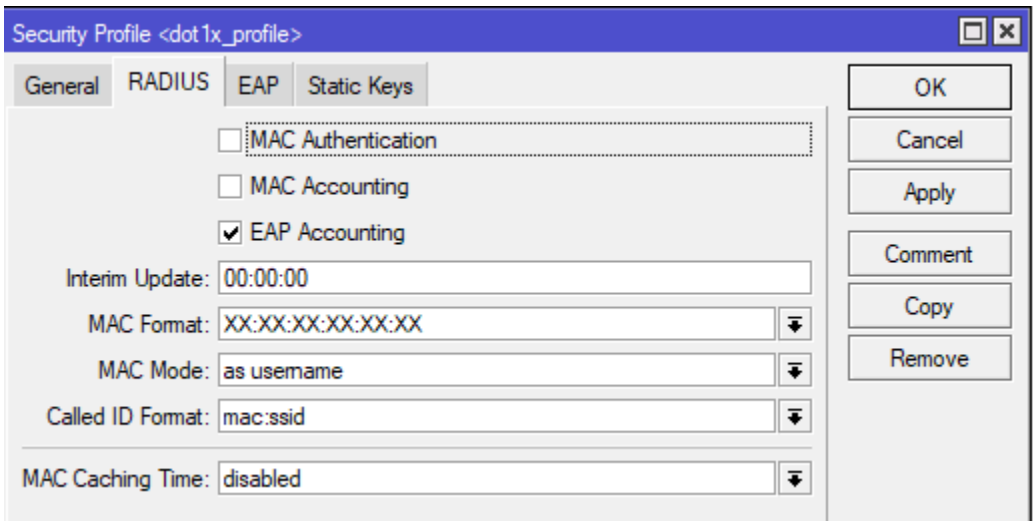


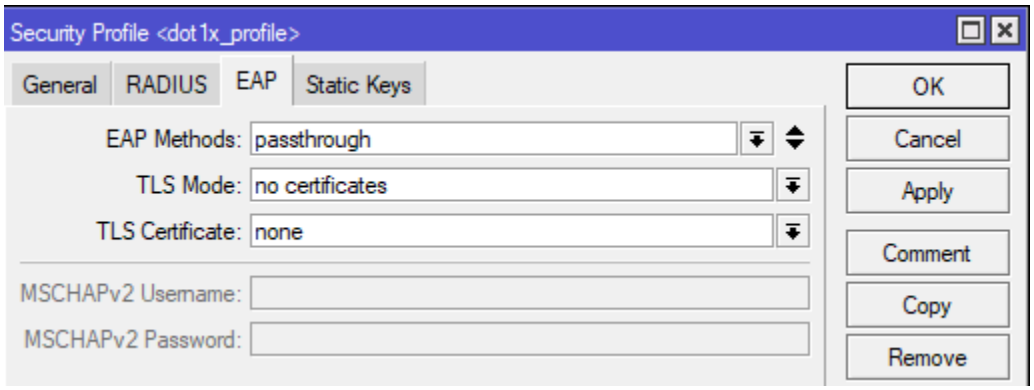
Command line equivalent is *"/radius add address=192.168.88.233 secret=yourSecret service=wireless timeout=1s500ms*"

The secret should match the one configured on the radsecproxy, in this example "192.168.88.233" is a virtual machine running the proxy.

2) Create a wireless security profile that would perform 802.1x authentication

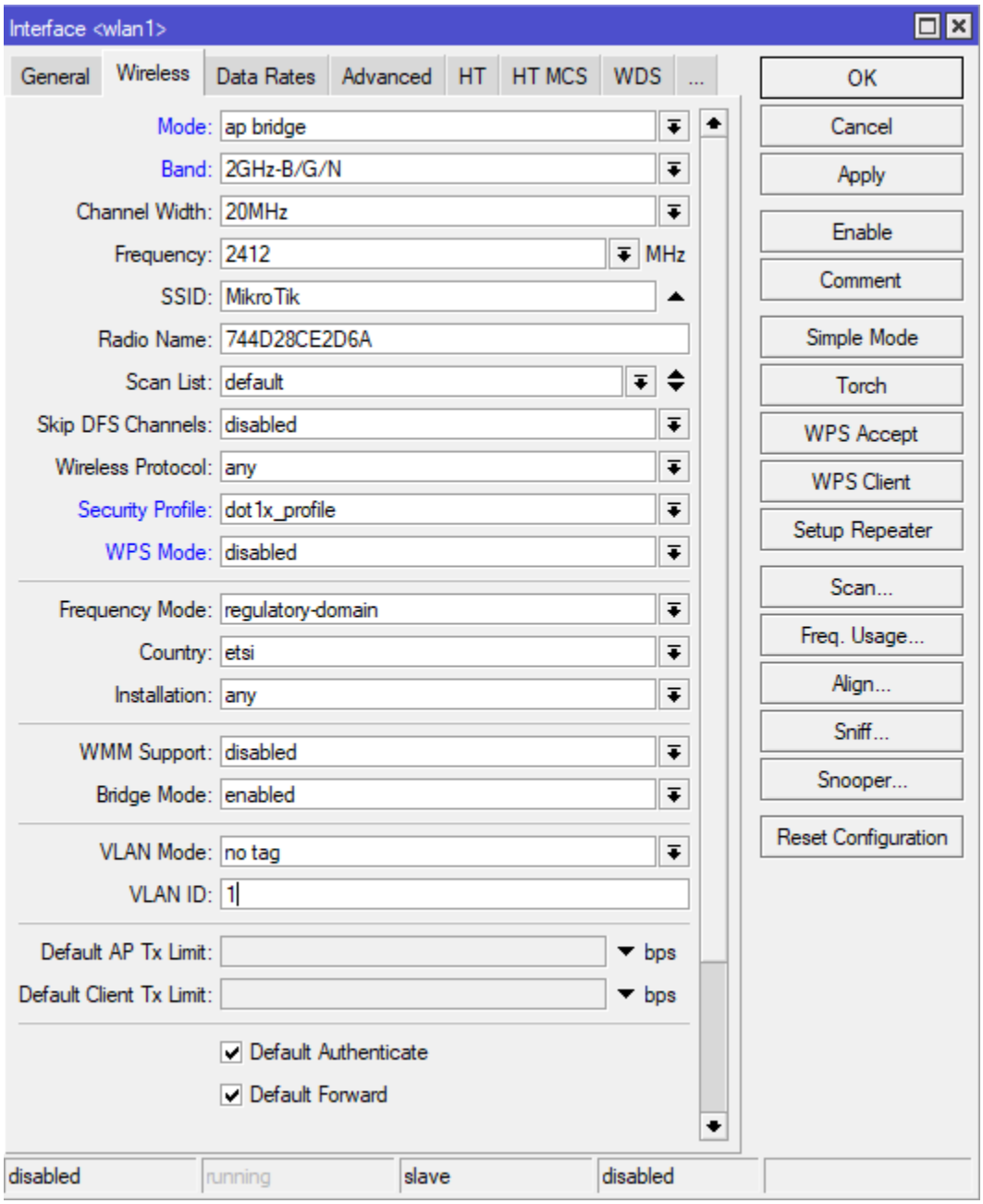

Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Tools
New Terminal
Dot1X
MetaROUTER
Partition
Make Supout.rif
Manual
New WinBox
Exit
Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Name
Mode
Authenticatio...
Unicast Ciphers
Group Ciphers
WPA Pre-Shared ...
WPA2 Pre-Shared...
default
none
New Security Profile
General
RADIUS
EAP
Static Keys
OK
Cancel
Apply
Comment
Copy
Remove
Name: dot1x_profile
Mode: dynamic keys
Authentication Types: WPA PSK WPA2 PSK
WPA EAP WPA2 EAP
Unicast Ciphers: aes ccm tkip
Group Ciphers: aes ccm tkip
WPA Pre-Shared Key:
WPA2 Pre-Shared Key:
Supplicant Identity:
Group Key Update: 00:05:00
Management Protection: allowed
Management Protection Key:
Disable PMKID


Security Profile <dot1x_profile>
General
RADIUS
EAP
Static Keys
OK
Cancel
Apply
Comment
Copy
Remove
MAC Authentication
MAC Accounting
EAP Accounting
Interim Update: 00:00:00
MAC Format: XX:XX:XX:XX:XX:XX
MAC Mode: as username
Called ID Format: mac:ssid
MAC Caching Time: disabled



Command line equivalent is *"/interface wireless security-profiles add authentication-types=wpa2-eap management-protection=allowed mode=dynamic-keys name=dot1x_profile supplicant-identity="" radius-eap-accounting=yes eap-methods=passthrough*".

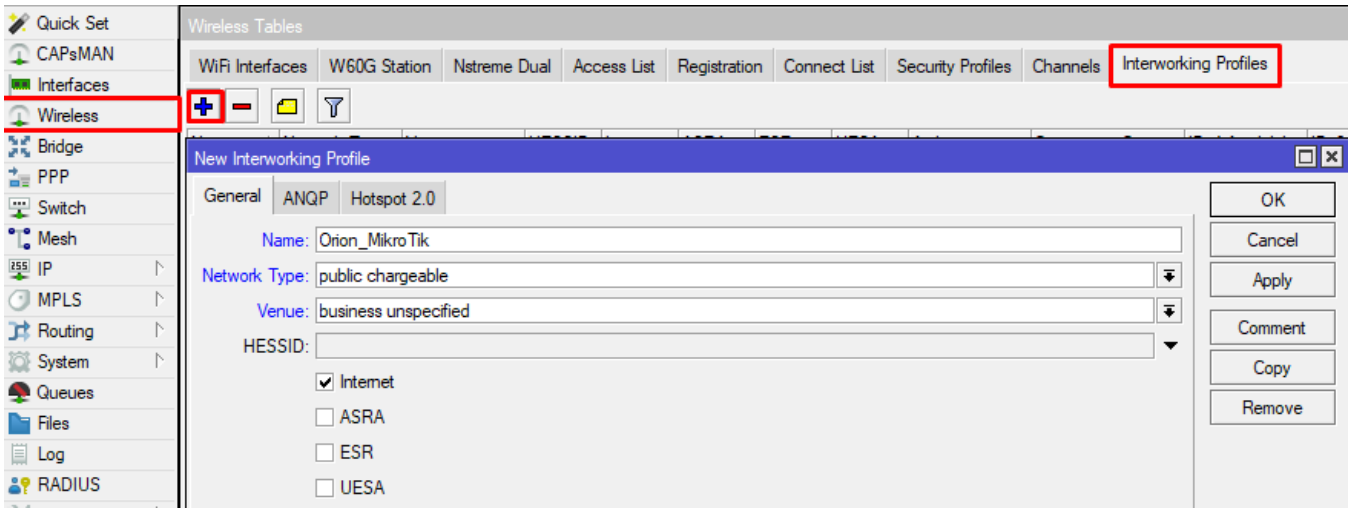
3) The next step is configuring the wireless interface and assigning the created security profile. Press "Advanced mode" to see all the options.

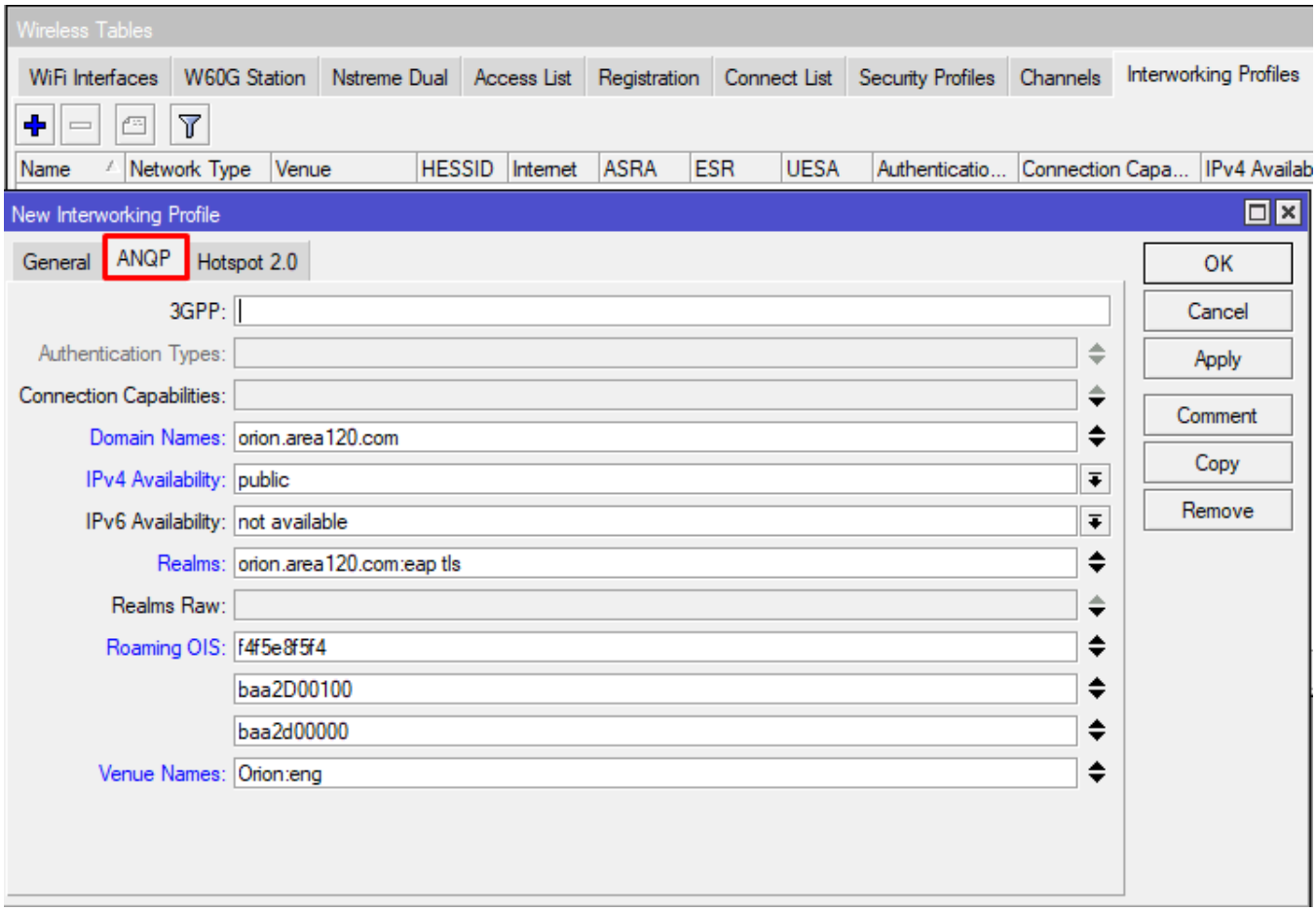


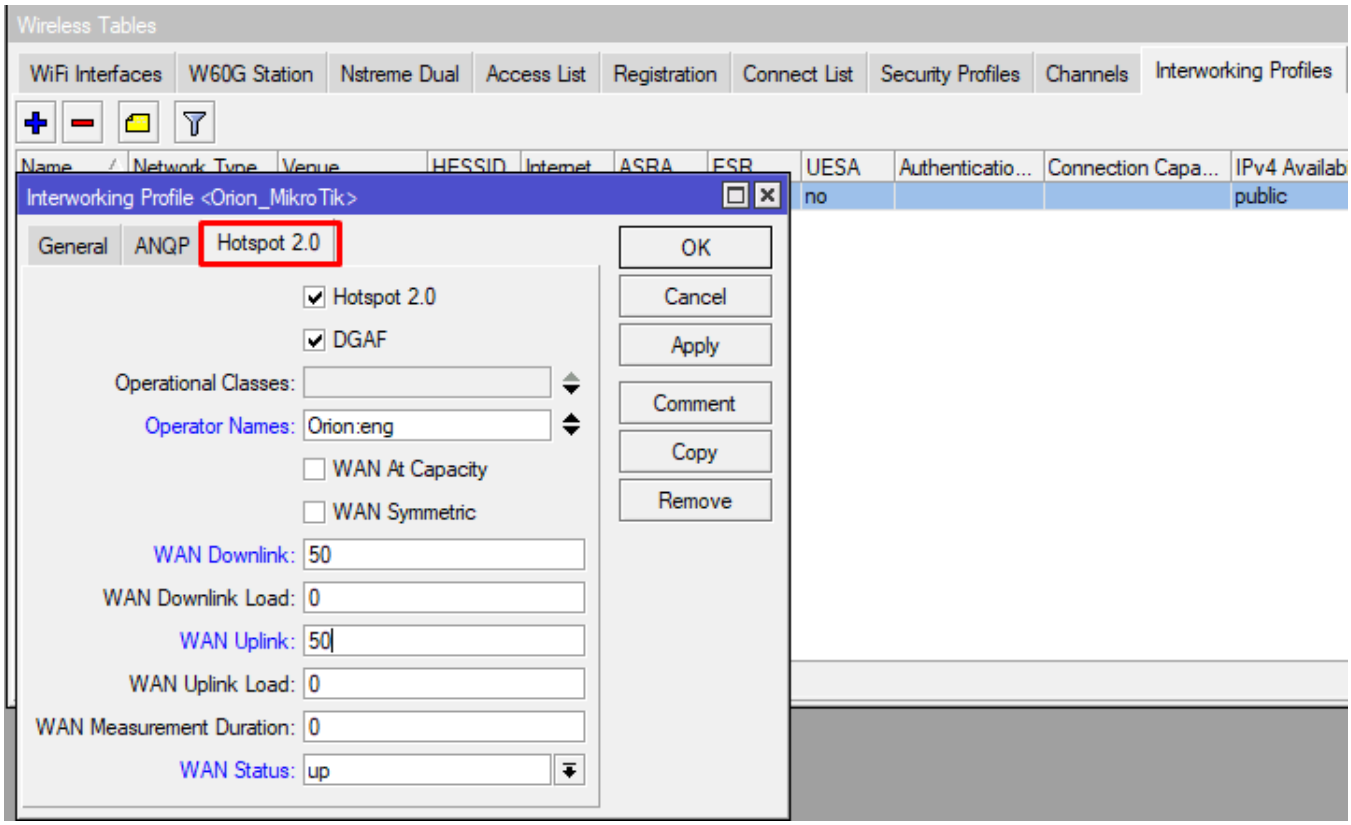
Command line equivalent is: "*/interface wireless set [find default-name=wlan1] mode=ap-bridge security-profile=dot1x_profile wps-mode=disabled*".

Make sure the correct country profile is configured. In this example, we are using “wlan1”, but the same command would work with other interfaces, or as “*/interface wireless set wlan1*”.

4) Configure interworking settings (hotspot 2.0).


Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
IP
MPLS
Routing
System
Queues
Files
Log
RADIUS
Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Interworking Profiles
New Interworking Profile
General
ANQP
Hotspot 2.0
Name: Orion_MikroTik
Network Type: public chargeable
Venue: business unspecified
HESSID:
Internet
ASRA
ESR
UESA
OK
Cancel
Apply
Comment
Copy
Remove

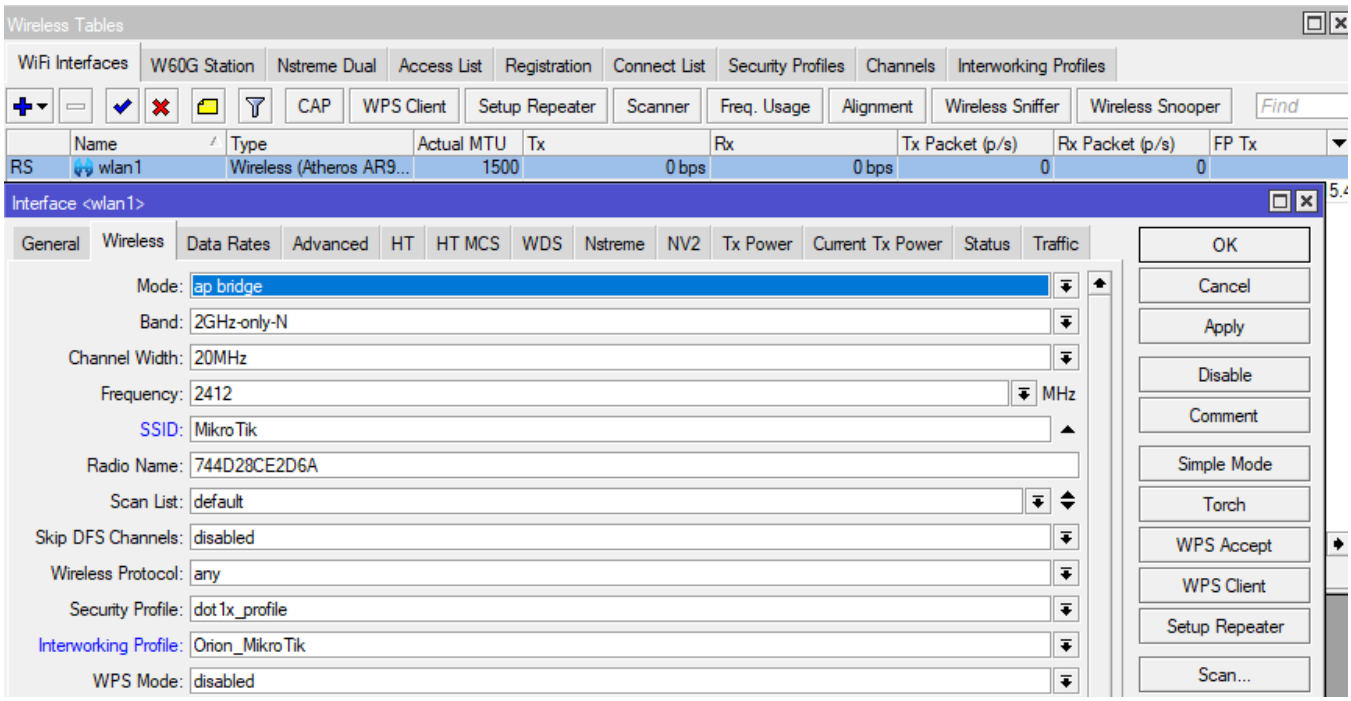

Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Interworking Profiles
Name
Network Type
Venue
HESSID
Internet
ASRA
ESR
UESA
Authenticatio...
Connection Capa...
IPv4 Availab
New Interworking Profile
General
ANQP
Hotspot 2.0
3GPP:
Authentication Types:
Connection Capabilities:
Domain Names: orion.area120.com
IPv4 Availability: public
IPv6 Availability: not available
Realms: orion.area120.com:eap tls
Realms Raw:
Roaming OIS: f4f5e8f5f4
baa2D00100
baa2d00000
Venue Names: Orion:eng
OK
Cancel
Apply
Comment
Copy
Remove



Command line equivalent: *"/interface wireless interworking-profile add domain-names=orion.area120.com ipv4-availability=public name=Orion_MikroTik network-type=public-chargeable operator-names=Orion:eng realms=orion.area120.com:eap-tls roaming-ois=f4f5e8f5f4,baa2D00100,baa2d00000 venue=business-unspecified venue-names=Orion:eng wan-downlink=50 wan-uplink=50 wan-status=up*".

> ⚠ Be sure to specify somevalue in "wan-downlink" and "wan-uplink", in this scenario value of "50" is used as a placeholder, some client devices use it to evaluate, if they should join the network. Set "venue" – venue type, "venue-names" and other attributes as applicable. "domain-names" should be of hotspot 2.0 Operator.

5) Assign the interworking profile to the interface.



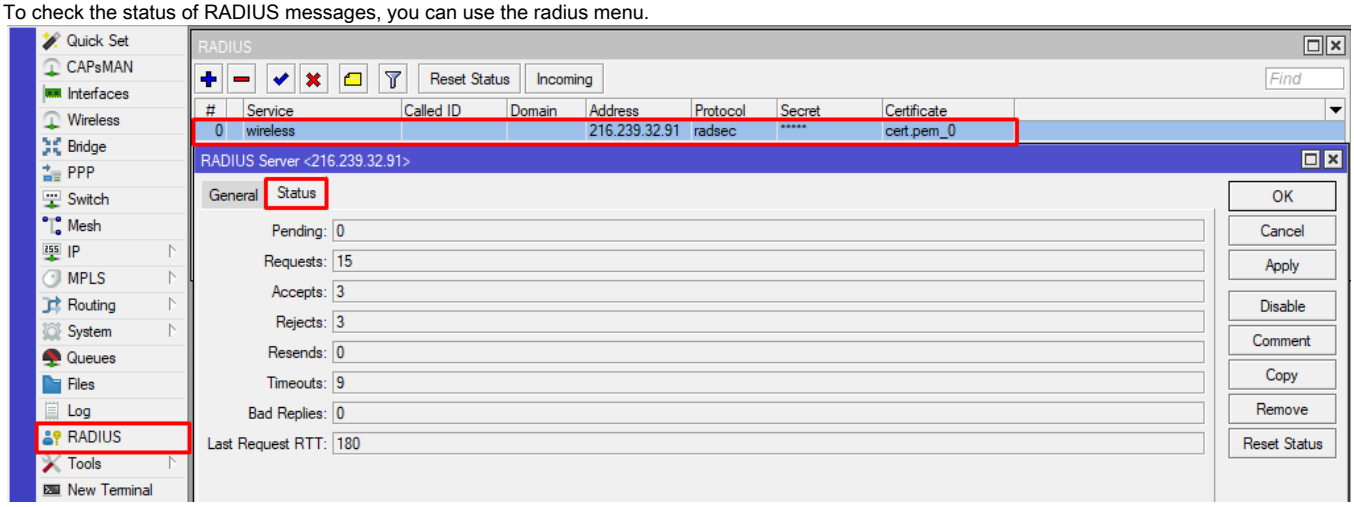
This step can also be done with the following command: “/interface wireless set wlan1 interworking-profile=Orion_MikroTik”.

If the radsecproxy is working, then clients with the appropriate Hotspot profile installed should be able to connect.

Note: NAS-id that's used by Orion to differentiate networks is equal to system identity, to adjust the nas-id, you can do "/system identity set name=exampleName". Graphical interface support for interworking profiles is added from versions above 6.47.10, 6.48.3.

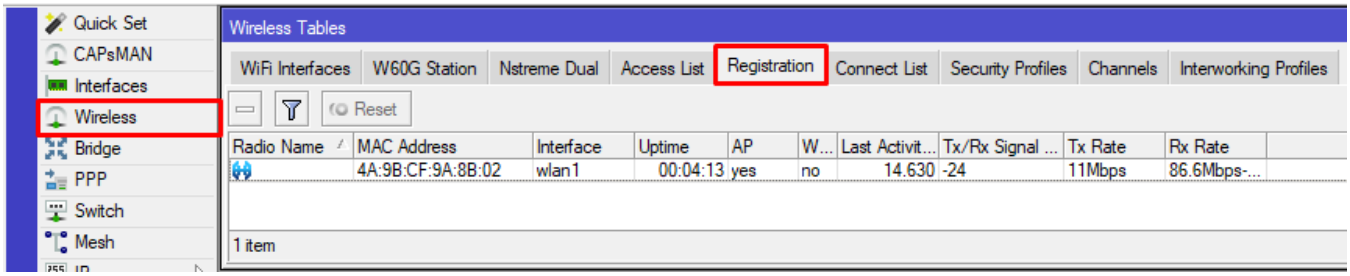
Troubleshooting

To check the status of RADIUS messages, you can use the radius menu.



Or alternatively via the command line run "/radius monitor X", X being the numerical ID, you can see the IDs with "/radius print".
For more information, additional logging can be configured under "/system logging add topics=radius,debug,packet". You can view results under "/log".

To view active wireless connections check the wireless registration table (*/interface wireless registration-table print*)


Quick Set
CAPsMAN
Interfaces
Wireless
Bridge
PPP
Switch
Mesh
Wireless Tables
WiFi Interfaces
W60G Station
Nstreme Dual
Access List
Registration
Connect List
Security Profiles
Channels
Interworking Profiles
Reset
Radio Name
MAC Address
Interface
Uptime
AP
W...
Last Activit...
Tx/Rx Signal ...
Tx Rate
Rx Rate
4A:9B:CF:9A:8B:02
wlan1
00:04:13
yes
no
14.630
-24
11Mbps
86.6Mbps-...
1 item

Wireless Case Studies

In This Section

Enterprise wireless security with User Manager v5

User Manager version 5 (available for RouterOS v7) supports user authentication via the Extensible Authentication Protocol (EAP).

This guide will explain the steps needed to configure User Manager v5 as the authentication server for MikroTik wireless access points with users being offered PEAP and EAP-TLS authentication methods.

The guide assumes a standalone device running User Manager at the network address 10.0.0.10 and 2 Access Points - one at 10.0.0.11 and the other at 10.0.0.12

Installing User Manager

User Manager v5 can be found in the 'Extra packages' archive for the latest release of RouterOS v7.

Download the archive for the appropriate CPU architecture, extract it, upload the User Manager package to the router and reboot it.

Generating TLS certificates

When using secure EAP methods, the client device (supplicant) verifies the identity of the authenication server before sending its own credentials to it. For this to happen, the authentication server needs a TLS certificate.

This certificate should:

1. Be valid and signed by a certificate authority which is trusted by the client device
2. Have a fully qualified domain name in the Common Name (CN) and Subject Alt Name fields
3. Have the Extended Key Usage attribute indicating that it is authorized for authentcating a TLS server
4. Have Validity period of no more than 825 days

The EAP-TLS method requires the client device to have a TLS certificate (instead of a password).

To be considered valid by User Manager, a client certificate must:

1. Be valid and signed by an authority, which is trusted by the device running User Manager
2. Have the user name in the Subject Alt Name (SAN) field. For backward compatibility, you can also add it in the CN field. For more information please see: https://datatracker.ietf.org/doc/html/rfc5216#section-5.2

Finally, the WPA3 enterprise specification includes an extra secure mode, which provides 192-bit cryptographic security.

This mode requires using EAP-TLS with certificates that:

1. Use either P-384 elliptic curve keys or RSA keys which are at least 3072 bits in length
2. Use SHA384 as the digest (hashing) algorithm

For the sake of brevity (and to showcase more of RouterOS' capabilities), this guide will show how to generate all the certificates on the device running User Manager, but in a large scale enterprise environment, the authentication server and client devices would each generate private keys and certificate signing requests locally, then upload CSRs to a certificate authority for signing.

Commands executed on device running User Manager

```
# Generating a Certificate Authority
/certificate
add name=radius-ca common-name="RADIUS CA" key-size=secp384r1 digest-algorithm=sha384 days-valid=1825 key-
usage=key-cert-sign,crl-sign
sign radius-ca ca-crl-host=radius.mikrotik.test
# Generating a server certificate for User Manager
add name=userman-cert common-name=radius.mikrotik.test subject-alt-name=DNS:radius.mikrotik.test key-
size=secp384r1 digest-algorithm=sha384 days-valid=800 key-usage=tls-server
sign userman-cert ca=radius-ca
# Generating a client certificate
add name=maija-client-cert common-name=maija@mikrotik.test key-usage=tls-client days-valid=800 key-
size=secp384r1 digest-algorithm=sha384
sign maija-client-cert ca=radius-ca
# Exporting the public key of the CA as well as the generated client private key and certificate for
distribution to client devices
export-certificate radius-ca file-name=radius-ca
# A passphrase is needed for the export to include the private key
export-certificate maija-client-cert type=pkcs12 export-passphrase="true zebra capacitor ziptie"
```

Configuring User Manager

Commands executed on device running User Manager

```
# Enabling User Manager and specifying, which certificate to use
/user-manager
set enabled=yes certificate=userman-cert
# Enabling CRL checking to avoid accepting revoked user certificates
/certificate settings
set crl-download=yes crl-use=yes
# Adding access points
/user-manager router
add name=ap1 address=10.0.0.11 shared-secret="Use a secure password generator for this"
add name=ap2 address=10.0.0.12 shared-secret="Use a secure password generator for this too"
# Limiting allowed authentication methods
/user-manager user group
set [find where name=default] outer-auths=eap-tls,eap-peap
add name=certificate-authenticated outer-auths=eap-tls
# Adding users
/user-manager user
add name=maija@mikrotik.test group=certificate-authenticated
add name=paija@mikrotik.test group=default password="right mule accumulator nail"
```

Configuring access points

AP running regular wireless package

Commands executed on ap1

```
# Configuring radius client
/radius
add address=10.0.0.10 secret="Use a secure password generator for this" service=wireless timeout=1s
/radius incoming
set accept=yes
# Adding a security profile and applying it to wireless interfaces
/interface/wireless/security-profile
add name=radius mode=dynamic-keys authentication-types=wpa2-eap
/interface/wireless
set [find] security-profile=radius
```

AP running wifiwave2 package

Commands executed on ap2

```
# Configuring radius client
/radius
add address=10.0.0.10 secret="Use a secure password generator for this too" service=wireless timeout=1s
/radius incoming
set accept=yes
# Configuring enabled authentication types. Can also be done via a security profile, but note that interface
properties, if specified, override profile properties
/interface/wifiwave2 set [find] security.authentication-types=wpa2-eap,wpa3-eap
```

A wifiwave2 AP can also be configured to use the extra secure wpa3-eap-192 mode, but note that it requires that all client devices support the GCMP-256 cipher and use EAP-TLS authentication.

Notes on client device configuration

Windows

When manually installing a CA in Windows, make sure to explicitly place it in the 'Trusted Root Certification Authorities' certificate store. It will not be placed there automatically.

Android

When connecting to a network with EAP authentication, Android devices ask the user to specify a 'domain'. This refers to the expected domain of the host name included in the RADIUS server's TLS certificate ('mikrotik.test' in our example).

By default, Android devices use the device's built-in root CA list for validating the RADIUS server's certificate. When using your own CA, it needs to be selected in the appropriate dropdown menu.

iOS

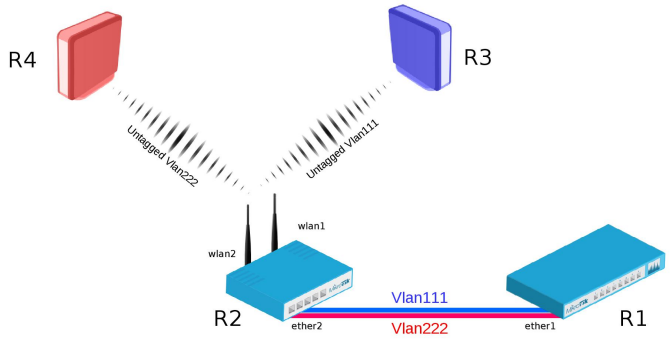
Apple iOS does not appear to actually trust a manually imported CA to authenticate RADIUS servers. The server certificate is marked as 'Not Trusted' unless the CA was imported using Apple's proprietary 'Configurator' utility or an approved third party MDM tool.

VLANs on Wireless

Summary

VLANs provide the possibility to isolate devices into different Layer2 segments while still using the same Layer1 medium. This is very useful in setups where you want to separate different types of devices of users. This feature is also very useful for Wireless setups since you can isolate different Virtual APs and restricting access to certain services or networks by using Firewall. Below is an example with a setup with two Access Points on the same device that isolates them into saparate VLANs. This kind of scenario is very common when you have a **Guest AP** and **Work AP**.

Example



Bridge VLAN Filtering since RouterOS v6.41 provides VLAN aware Layer2 forwarding and VLAN tag modifications within the bridge.

R1:

- Add necessary VLAN interfaces on ethernet interface to make it a VLAN trunk port. Add ip addresses on VLAN interfaces.

```
[admin@R1] >
/interface vlan
add interface=ether1 name=vlan111 vlan-id=111
add interface=ether1 name=vlan222 vlan-id=222

/ip address
add address=192.168.1.1/24 interface=vlan111
add address=192.168.2.1/24 interface=vlan222
```

R2:

- Add VirtualAP under wlan1 interface and create wireless security-profiles for wlan1 and wlan2

```
[admin@R2] >
/interface wireless
set [ find default-name=wlan1 ] disabled=no mode=ap-bridge security-profile=vlan111 ssid=vlan111 vlan-id=111
vlan-mode=use-tag
add disabled=no master-interface=wlan1 name=wlan2 security-profile=vlan222 ssid=vlan222 vlan-id=222 vlan-
mode=use-tag
```

It is important to set wlan1,wlan2 vlan-mode to "use-tag".

- Create bridge with *vlan-filtering=yes*
- Add necessary bridge ports
- Add *tagged* interfaces under *interface bridge vlan* section with correct *vlan-ids*

```
[admin@R2] >
/interface bridge
add fast-forward=no name=bridge1 vlan-filtering=yes

/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=wlan1
add bridge=bridge1 interface=wlan2
/interface bridge vlan
add bridge=bridge1 tagged=ether2,wlan1 vlan-ids=111
add bridge=bridge1 tagged=ether2,wlan2 vlan-ids=222
```

Some devices have a built-in switch chip that can switch packets between Ethernet ports with wire-speed performance. Bridge VLAN filtering disables hardware offloading (except on CRS3xx series switches), which will prevent packets from being switched, this does not affect Wireless interfaces as traffic through them cannot be offloaded to the switch chip either way.

VLAN filtering is not required in this setup, but is highly recommended due to security reasons. Without VLAN filtering it is possible to forward unknown VLAN IDs in certain scenarios. Disabling VLAN filtering does have performance benefits.

R3:

- Add IP address on wlan1 interface.
- Create wireless security-profile compatible with R2 wlan1.

```
[admin@R3] >
/ip address
add address=192.168.1.3/24 interface=wlan1

/interface wireless
set [ find default-name=wlan1 ] disabled=no security-profile=vlan111
```

R4:

- Add IP address on wlan1 interface.
- Create wireless security-profile compatible with R2 wlan2.

```
[admin@R4] >
/ip address
add address=192.168.2.4/24 interface=wlan1

/interface wireless
set [ find default-name=wlan1 ] disabled=no security-profile=vlan222
```

Wireless Station Modes

Overview

Wireless interface in any of *station* modes will search for acceptable access point (AP) and connect to it. The connection between station and AP will behave in slightly different way depending on type of *station* mode used, so correct mode must be chosen for given application and equipment. This article attempts to describe differences between available *station* modes.

Primary difference between *station* modes is in how L2 addresses are processed and forwarded across wireless link. This directly affects the ability of wireless link to be part of L2 bridged infrastructure.

If L2 bridging over wireless link is not necessary - as in case of routed or MPLS switched network, basic **mode=station** setup is suggested and will provide highest efficiency.

Availability of particular *station* mode depends on **wireless-protocol** that is used in wireless network. Please refer to applicability matrix for information on mode support in protocols. It is possible that connection between station and AP will be established even if particular mode is not supported for given protocol. Beware that such connection will not behave as expected with respect to L2 bridging.

802.11 limitations for L2 bridging

Historically 802.11 AP devices were supposed to be able to bridge frames between wired network segment and wireless, but station device was not supposed to do L2 bridging.

Consider the following network:

```
[X]---[AP]-(      )-[STA]---[Y]
```

where X-to-AP and STA-to-Y are ethernet links, but AP-to-STA are connected wirelessly. According to 802.11, AP can transparently bridge traffic between X and STA, but it is not possible to bridge traffic between AP and Y, or X and Y.

802.11 standard specifies that frames between station and AP device must be transmitted in so called *3 address* frame format, meaning that header of frame contains 3 MAC addresses. Frame transmitted from AP to station has the following addresses:

- destination address - address of station device, also radio receiver address
- radio transmitter address - address of AP
- source address - address of originator of particular frame

Frame transmitted from station to AP has the following addresses:

- radio receiver address - address of AP
- source address - address of station device, also radio transmitter address
- destination address

Considering that every frame must include radio transmitter and receiver address, it is clear that *3 address* frame format is not suitable for transparent L2 bridging over station, because station can not send frame with source address different from its address - e.g. frame from Y, and at the same time AP can not format frame in a way that would include address of Y.

802.11 includes additional frame format, so called *4 address* frame format, intended for "wireless distribution system" (WDS) - a system to interconnect APs wirelessly. In this format additional address is added, producing header that contains the following addresses:

- radio receiver address
- radio transmitter address
- destination address
- source address

This frame format includes all necessary information for transparent L2 bridging over wireless link. Unluckily 802.11 does not specify how WDS connections should be established and managed, therefore any usage of *4 address* frame format (and WDS) is implementation specific.

Different *station* modes attempt to solve shortcomings of standard station mode to provide support for L2 bridging.

Applicability Matrix

The following matrix specifies *station* modes available for each **wireless-protocol**. Note that there are 2 columns for 802.11 protocol: **802.11** specifies availability of mode in "pure" 802.11 network (when connecting to any vendor AP) and **ROS 802.11** specifies availability of mode when connecting to RouterOS AP that implements necessary proprietary extensions for mode to work.

Table applies to RouterOS v5rc11 and above:

	802.11	ROS 802.11	nstreme	nv2
station	V	V	V	V
station-wds		V	V	V
station-pseudobridge	V	V	V	
station-pseudobridge-clone	V	V	V	
station-bridge		V	V	V

Mode *station*

This is standard mode that does not support L2 bridging on station - attempts to put wireless interface in bridge will not produce expected results. On the other hand this mode can be considered the most efficient and therefore should be used if L2 bridging on station is not necessary - as in case of routed or MPLS switched network. This mode is supported for all wireless protocols.

Mode *station-wds*

This mode works only with RouterOS APs. As a result of negotiating connection, separate WDS interface is created on AP for given station. This interface can be thought of point-to-point connection between AP and given station - whatever is sent out WDS interface is delivered to station (and only to particular station) and whatever station sends to AP is received from WDS interface (and not subject to forwarding between AP clients), preserving L2 addresses.

This mode is supported for all wireless protocols except when 802.11 protocol is used in connection to non-RouterOS device. Mode uses *4 address* frame format when used with 802.11 protocol, for other protocols (such as nstreme or nv2), protocol internal means are used.

This mode is safe to use for L2 bridging and gives most administrative control on AP by means of separate WDS interface, for example use of bridge firewall, RSTP for loop detection and avoidance, etc.

With station-wds mode, it is not possible to connect to CAPsMAN controlled CAP.

Mode *station-pseudobridge*

From the wireless connection point of view, this mode is the same as standard station mode. It has limited support for L2 bridging by means of some services implemented in station:

- MAC address translation for IPv4 packets - station maintains IPv4-to-MAC mapping table and replaces source MAC address with its own address when sending frame to AP (in order to be able to use *3 address* frame format), and replaces destination MAC address with address from mapping table for frames received from AP. IPv4-to-MAC mappings are built also for VLAN encapsulated frames.
- single MAC address translation for the rest of protocols - station learns source MAC address from first forwarded non-IPv4 frame and uses it as default for reverse translation - this MAC address is used to replace destination MAC address for frames received from AP if IPv4-to-MAC mapping can not be performed (e.g. - non-IPv4 frame or missing mapping).

This mode is limited to complete L2 bridging of data to single device connected to station (by means of single MAC address translation) and some support for IPv4 frame bridging - bridging of non-IP protocols to more than one device will not work. Also MAC address translation limits access to station device from AP side to IPv4 based access - the rest of protocols will be translated by single MAC address translation and will not be received by station itself.

This mode is available for all protocols except nv2 and **should be avoided when possible**. The usage of this mode can only be justified if AP does not support better mode for L2 bridging (e.g. when non-RouterOS AP is used) or if only one end-user device must be connected to network by means of station device.

Mode *station-pseudobridge-clone*

This mode is the same as *station-pseudobridge* mode, except that it connects to AP using "cloned" MAC address - that is either address configured in **station-bridge-clone-mac** parameter (if configured) or source address of first forwarded frame. This essentially appears on AP as if end-user device connected to station connected to AP.

Mode *station-bridge*

This mode works only with RouterOS APs and provides support for transparent protocol-independent L2 bridging on the station device. RouterOS AP accepts clients in *station-bridge* mode when enabled using **bridge-mode** parameter. In this mode, the AP maintains a forwarding table with information on which MAC addresses are reachable over which station device.

This mode is MikroTik proprietary and cannot be used to connect to other brands of devices.

This mode is safe to use for L2 bridging and is the preferred mode unless there are specific reasons to use *station-wds* mode. With station-bridge mode, it is not possible to connect to CAPsMAN controlled CAP.

Wireless Troubleshooting

Wireless Debug Logs

Debugging wireless problems using Logs.

By default RouterOS wireless log shows that client connects and disconnects as simple entries:

```
22:32:18 wireless,info 00:80:48:41:AF:2A@wlan1: connected
```

It is enough for regular users to know that the wireless client with MAC address "00:80:48:41:AF:2A" is connected to wireless interface "wlan1". But actually there are more log entries available than are shown in standard logging. They are called 'debug' logs which give more detailed information. In the following Debug Log example you will see the same client connecting to the AP in more detail than found in typical logging:

```
22:33:20 wireless,debug wlan1: 00:80:48:41:AF:2A attempts to connect
22:33:20 wireless,debug wlan1: 00:80:48:41:AF:2A not in local ACL, by default accept
22:33:20 wireless,info 00:80:48:41:AF:2A@wlan1: connected
```

Debug Logs will give you more specific information on each step of the Client wireless connection and disconnection. The first line shows that the wireless client tried to connect to the AP. On the second line the AP checked to see if that client is allowed to connect to the AP and the resulting action. And only on the third line do you see that the client is connected. This is merely one example of the debug log messages. The description of all debug entries is written below.

To enable the wireless debug logs you should execute such commands:

```
[admin@MikroTik] > /system logging
[admin@MikroTik] system logging> add topics=wireless,debug action=memory
```

This will help you understand and fix wireless problems with ease and with less interaction with the support team.

STATION MODE

<MAC>@<DEV>: lost connection, <REASON>

Station has lost connection to AP because of <REASON>

<MAC>@<DEV>: failed to connect, <REASON>

Station attempted to connect to AP, but failed due to <REASON>

<MAC>@<DEV>: established connection on <FREQ>, SSID <SSID>

Station attempted and succesfully connected to AP with SSID <SSID> on frequency <FREQ>.

<MAC>@<DEV>: MIC failure!!!

TKIP message integrity check failure, somebody must be trying to break into or DOS network, If more than 1 MIC failure is encountered during 60s period, "TKIP countermeasures" state is entered.

<MAC>@<DEV>: enter TKIP countermeasures

Entered TKIP countermeasures state, this means that Station will disconnect from AP and keep silence for 60s.

AP MODE

<DEV>: radar detected on <FREQ>

Radar detected on frequency <FREQ>, AP will look for other channel

<DEV>: data from unknown device <MAC>, sent deauth [(XXX events suppressed, YYY deauths suppressed)]

Data frame from unknown device (read - not registered to this AP) with mac address <MAC> received, AP sent deauthentication frame to it (as per 802.11). XXX is number of events that are not logged so that the log does not become too large (logs are limited to 1 entry per 5s after first 5 entries), YYY is the number of deauthentication frames that should have been sent, but were not sent, so that resources are not wasted sending too many deauthentication frames (only 10 deauth frames per second are allowed).

The likely cause of such a message is that the Station previously connected to the AP, which does not yet know it has been dropped from AP registration table, sending data to AP. Deauthentication message tells the Station that it is no longer connected.

<DEV>: denying assoc to <MAC>, failed to setup compression

Failed to initialize compression on AP, most likely because there are too many clients attempting to connect and use compression.

<DEV>: <MAC> is new WDS master

WDS slave has established connection to WDS master, this means that WDS slave starts accepting clients and acting as AP.

<DEV>: <MAC> was WDS master

This message appears after connection with <MAC> is lost, means that WDS slave will disconnect all clients and start scanning to find new WDS master.

<MAC>@<DEV>: connected [, is AP][, wants WDS]

Station with address <MAC> connected. if "is AP" present - remote device is AP, if "is WDS" presents, remote device wants to establish WDS link.

<MAC>@<DEV>: disconnected, <REASON>

Connection with Station with address <MAC> terminated due to <REASON>

<DEV>: TKIP countermeasures over, resuming

TKIP countermeasures (60s silence period) over, AP resumes acting as AP.

<DEV>: starting TKIP countermeasures

Entering TKIP countermeasures state (60s silence period), all clients will be lost.

<REASON>

"joining failed" - can only happen on Prism cards in station mode, failed to connect to AP due to some reason

"join timeout" - happens on Station, failed to synchronize to AP (receive first beacon frame). Most likely weak signal, remote turned off, strong interference, some other RF related issue that makes communication impossible.

"no beacons" - no beacons received from remote end of WDS link. Most likely weak signal, remote turned off, strong interference, some other RF related issue that makes communication impossible.

"extensive data loss" - local interface decided to drop connection to remote device because of inability to send data to remote after multiple failures at lowest possible rate. Possible causes - too weak signal, remote device turned off, strong interference, some other RF related issue that makes communication impossible.

"decided to deauth, <802.11 reason>" - local interface decided do deauthenticate remote device using 802.11 reason <802.11 reason>.

"inactivity" - remote device was inactive for too long

"device disabled" - local interface got disabled

"got deauth, <802.11 reason>" - received deauthentication frame from remote device, 802.11 reason code is reported in <802.11 reason>

"got disassoc, <802.11 reason>" - received disassociation frame from remote device, 802.11 reason code is reported in <802.11 reason>

"auth frame from AP" - authentication frame from remote device that is known to be AP, most likely mode changes on remote device from AP to Station.

"bad ssid" - bad ssid for WDS link

"beacon from non AP" - received beacon frame from remote device that is known to be non-AP node, most likely mode changes on remote device from Station to AP.

"no WDS support" - does not report WDS support

"failed to confirm SSID" - failed to confirm SSID of other end of WDS link.

"hardware failure" - some hardware failure or unexpected behavior. Not likely to be seen.

"lost connection" - can only happen on Prism cards in station mode, connection to AP lost due to some reason.

"auth failed <802.11 status>" - happens on Station, AP denies authentication, 802.11 status code is reported in <802.11 status>.

"assoc failed <802.11 status>" - happens on Station, AP denies association, 802.11 status code is reported in <802.11 status>.

"auth timeout" - happens on Station, Station does not receive response to authentication frames, either bad link or AP is ignoring this Station for some reason.

"assoc timeout" - happens on Station, Station does not receive response to association frames, either bad link or AP is ignoring this Station for some reason.

"reassociating" - happens on AP: connection assumed to be lost, because Station that is considered already associated attempts to associate again. All connection related information must be deleted, because during association process connection parameters are negotiated (therefore "disconnected"). The reason why Station reassociates must be looked for on Station (most likely cause is that Station for some reason dropped connection without telling AP - e. g. data loss, configuration changes).

"compression setup failure" - connection impossible, because not enough resources to do compression (too many stations that want to use compression already connected)

"control frame timeout" - AP was unable to transmit to the client (similar to error message that you see in the 802.11 protocol - extensive data loss)

<802.11 reason> and <802.11 status>

These are numeric reason/status codes encoded into 802.11 management messages. Log messages include numeric code and textual description from appropriate standard in 802.11 standards group. Although these are intended to be as descriptive as possible, it must be taken into account that actual reason/status code that appears in management frames depends solely on equipment or software manufacturer - where one device sends 802.11 management frame including proper reason/status code for situation that caused the frame, other may send frame with "unspecified" reason/status code. Therefore reason/status code should only be considered informational.

As 802.11 standards evolve, RouterOS may miss textual descriptions for reason/status codes that some devices use. In such case numeric value should be used to lookup meaning in 802.11 standards.

In order to properly interpret reason/status code, good understanding of 802.11 group standards is necessary. Most of the textual descriptions are self-explaining. Explanation for some of most commonly seen reson/status codes follows.

class 2 frame received (6) - device received "class 2" frame (association/reassociation management frame) before completing 802.11 authentication process;

class 3 frame received (7) - device received "class 3" frame (data frame) before completing association process;

Wireless FAQ

Settings

Why I can't connect to MikroTik 802.11n AP with Apple Mac devices?

This problem is only seen on Mac devices based on Broadcom wireless chipsets. In order to connect with such wireless device to MikroTik 802.11n AP make sure that you don't use 'short' preamble-mode. Use 'long' or 'both' preamble-mode and Mac wireless devices will be able to connect.

By changing some wireless settings the wireless link works unstable

Sometimes when you change some wireless setting for tuning the links you got so far that the link isn't establishing any more or works unstable and you don't remember what settings you had in the beginning. In this case, you can use the *reset-configuration* command in the wireless menu - it will reset the all the wireless settings for the specific wireless interface and you will be able to configure the interface from the start. Note that executing this command also disables the interface, so please be careful not to execute this command if you are configuring router remotely using that wireless link that you want to reset the configuration.

What are wireless retransmits and where to check them?

Wireless retransmission occur when an interface sends out a frame and doesn't receive back an acknowledgment (ACK), causing it to try sending the frame again until an acknowledgment is received or the maximum allowed retransmission count for a packet is reached. Wireless retransmits increase the latency and lower the throughput of a wireless link. The number of retransmissions taking place can be determined by subtracting the value of the **frames** parameter from the value of the 'hw-frames **parameter for a given entry in the registration table. Some number of retransmissions is to be expected, but if the value of** frames **exceeds the value of** frames **multiple times, there is an issue with the wireless link that requires troubleshooting.**

Can I compare frames with hw-frames also on Nstreme links?

The **frames** counts only those which contain actual data. In the case of Nstreme, only the ACK can be transmitted in a single frame, if there is no other data to send. These ACK frames will not be added to the **frames** count, but they will appear at **hw-frames**. If there is traffic on both directions at maximum speed (eg. there will be no only-ack frames), then you can't compare **frames** to **hw-frames** as in case of regular wireless links.

What TX-power values can I use?

The tx-power default setting is the maximum tx-power that the card can use and is taken from the cards eeprom. If you want to use larger tx-power values, you are able to set them, but **do it at your own risk**, as it will probably damage your card eventually! Usually, one should use this parameter only to reduce the tx-power.

In general, tx-power controlling properties should be left at the default settings. Changing the default setting may help with some cards in some situations, but without testing, the most common result is the degradation of range and throughput. Some of the problems that may occur are:

- overheating of the power amplifier chip and the card which will cause lower efficiency and more data errors;
- overdriving the amplifier which will cause more data errors;
- excessive power usage for the card and this may overload the 3.3V power supply of the board that the card is located on resulting in voltage drop and reboot or excessive temperatures for the board.

What TX-power-mode is the best?

TX-power-mode tells the wireless card which tx-power values should be used. By default, this setting is set to *default*.

- **default** means that the card will use the tx-power values from the cards eeprom and will ignore the setting what is specified by the user in the *tx-power* field.
- **card-rates** means that for different data rates the tx-power is calculated according to the cards transmit power algorithm from the cards eeprom and as an argument it takes *tx-power* value specified by the user.
- **all-rates-fixed** means that that the card will use one tx-power value for all data rates which is specified by the user in *tx-power* field.

Note that it is not recommended to use 'all-rates-fixed' mode as the wireless card tx-power for the higher data rates is lower and by forcing to use the fixed tx-power rates also for the higher data rates might result in the same problems like in the previous question about tx-power setting. In the case of MikroTik Radio devices, the power will not be higher than the power written in the EEPROM. For most of the cases if you want to change the tx-power settings it is recommended to use the *tx-power-mode=card-rates* and it is recommended to lower and not to raise tx-power. In case of AR9300 and newer Atheros wireless chipsets "tx-power-mode=all-rate-fixed" is the only option as "card-rates" option isn't working on those chipsets.

What is CCQ and how are the values determined?

Client Connection Quality (CCQ) is a value in percent that shows how effective the bandwidth is used regarding the theoretically maximum available bandwidth. CCQ is weighted average of values Tmin/Treal, that get calculated for every transmitted frame, where Tmin is time it would take to transmit given frame at highest rate with no retries and Treal is time it took to transmit frame in real life (taking into account necessary retries it took to transmit frame and transmit rate).

What is hw-retries setting?

Number of times sending frame is retried without considering it a transmission failure. The data rate is decreased upon failure and frame is sent again. Three sequential failures on lowest supported rate suspend transmission to this destination for the duration of *on-fail-retry-time.* After that, the frame is sent again. The frame is being retransmitted until transmission success, or until the client is disconnected after *disconnect-timeout*. The frame can be discarded during this time if *frame-lifetime* is exceeded. In case of Nstreme "on-fail-retry-time", "disconnect-timeout" and "frame-lifetime" settings are not used. So after three sequential failures on the lowest supported rate, the wireless link is disconnected with "extensive data loss" message.

What is disconnect-timeout setting?

This interval is measured from the third sending failure on the lowest data rate. At this point 3 * (*hw-retries* + 1) frame transmits on the lowest data rate had failed. During *disconnect-timeout* packet transmission will be retried with *on-fail-retry-time* interval. If no frame can be transmitted successfully during *disconnect-timeout*, the connection is closed, and this event is logged as "extensive data loss". Successful frame transmission resets this timer.

What is adaptive-noise-immunity setting?

Adaptive Noise Immunity (ANI) adjusts various receiver parameters dynamically to minimize interference and noise effect on the signal quality. This setting is added in the wireless driver for Atheros AR5212 and newer chipset cards

How does wireless device measure signal strength, when access-list or connect-list are used ?

The reported signal level is exponentially weighted moving average with a smoothing factor of 50%.

What error correction methods are supported in the RouterOS wireless?

ARQ method is supported in nstreme protocols. Regular 802.11 standard does not include ARQ - retransmission of corrupt frames is based on acknowledgment protocol. RouterOS supports forward error correction coding (convolutional coding) with coding rates: 1/2, 2/3, or 3/4.

Configuring RouterOS device for 160MHz use

If the RouterOS device supports 4x4 transmission, additionally to setting 160MHz channel width, make sure to set "rate-set=default" on the wireless interface so all streams are available

If the client does not support Extended NSS and can only perform 2x2 transmission, set "vht-supported-mcs=mcs0-9,mcs0-9,none"

Setups

Will an amplifier improve the speed on my link?

It depends on your signal quality and noise. Remember that you can probably get a better link with low output power setting, and a good antenna. An amplifier increases the noise and will only cause problems with the link.

The amplifier gets a boost on both the transmitted **and** received signal. Thus, in "silent" areas, where you are alone or with very few "noise" or "competition", you might get excellent results. On the other side, in crowded areas, with lots of wireless activity, you will also increase signal received from every other competitor or noise source, which may dramatically lower the overall quality of the link. Also, take in account the EIRP to see if your link remains in legal limits.

You could also get a better signal on "11b only" radios, which see most of 802.11g as "noise", thus filtering better the usable signal.

How to fine-tune the wireless link with hw-retries?

You should understand that for 802.11 devices there is a really limited amount of information (or "feedback" from the environment) that devices can use to tune their behavior:

- signal strength, which could be used to figure out best transmit rate knowing receiver sensitivity. Sill this is not reliable taking into account that sensitivity for different receivers varies (e.g. changes over time), path conditions are not symmetric (and device can only measure signal strength it receives), etc.
- by receiving/not receiving acknowledgment for frame sent.

Taking into account that using signal strength is not reliable, 802.11 devices are essentially left with only one "feedback" to tune its operation - success /failure of transmission. When transmission fails (ACK not received in time), there is no way how sender can figure out why it failed - either because of noise, multipath, direct interference (and whether that disturbed actual data frame or the ACK itself) - frame just did not make it and in general it does not matter "why". All that matters is the packet error rate.

Therefore RouterOS implements an algorithm to try to use medium most efficiently in any environment by using only this limited information, giving users the ability to control how the algorithm works and describing the algorithm. And there are only a few usage guidelines, not a set of values you should use in a particular situation.

In general - the larger *hw-retries*, the better "feedback" device gets about medium ability to deliver frame at particular rate (e.g. if sending frame with rate 54mbps fails 16 times, it is telling you more than if it fails with 2 retries) and the better it can figure out optimal transmit rate, at the expense of latency it can introduce in network - because during all those failing retries, other devices in this channel cannot send. So **bigger** *hw-retries* can be suggested for ptp backbone links - where it is known that link must be always on. **Less** *hw-retries* make rate selection adapt faster at the expense of some accuracy (going below 2 is not reasonable in most cases), this can be suggested for ptmp links, where it is normal for links to connect/disconnect and keeping latency down is important.

on-fail-retry-time and *disconnect-timeout* controls how hard device will try to consider remote party "connected". Larger *disconnect-timeout* will make the device not "disconnect" other party, even if there are lots of loss at the smallest possible transmission rate. This again is most useful for "weak" links that are known that they "must" be established (e.g. backbone links). In ptmp networks large *disconnect-timeout* will again increase latency in the network during the time e.g. AP will attempt to send data to some client that has just been disabled (AP will try to do this for the whole *disconnect-timeout*).

frame-lifetime allows to tune for how long AP is attempting to use frame for transmitting before considering that it is not worth delivering it (for example, if sending frame fails at lowest possible rate, *on-fail-retry-time* timer is enabled, if during this timer *frame-lifetime* expires, particular frame is dropped and the next transmission attempt will happen with the next frame. Disabled *frame-lifetime* means that wireless will ensure in order delivery of "all" data frames, no matter how long it takes, "or" will drop the connection if everything fails). This allows optimizing for different types of traffic e.g. for real-time traffic - if the primary use of the wireless network is e.g. voip, then it can be reasonable to limit *frame-lifetime*, because voip tolerates small loss better than high latency.

Is it possible to use the wireless repeater only with one radio interface?

This setup it possible by using WDS on the wireless interface which is running in ap-bridge mode. And in newer RouterOS versions it is possible to configure wireless repeater mode.

Nv2 wireless link disconnects very often

When Nv2 wireless link experiences disconnections and in log section most of the messages are 'control frame timeout', you can try to lower the transmit output power of the wireless cards if the signal of the link is strong. We suggest using tx-power-mode=card-rates for lowering the tx-power of the wireless card. If the problem continues to try to use a different wireless frequency that might have less interference. If that also didn't help, please contact support@mikrotik.com with a support output file from the affected AP and the Station which are made after those disconnections.

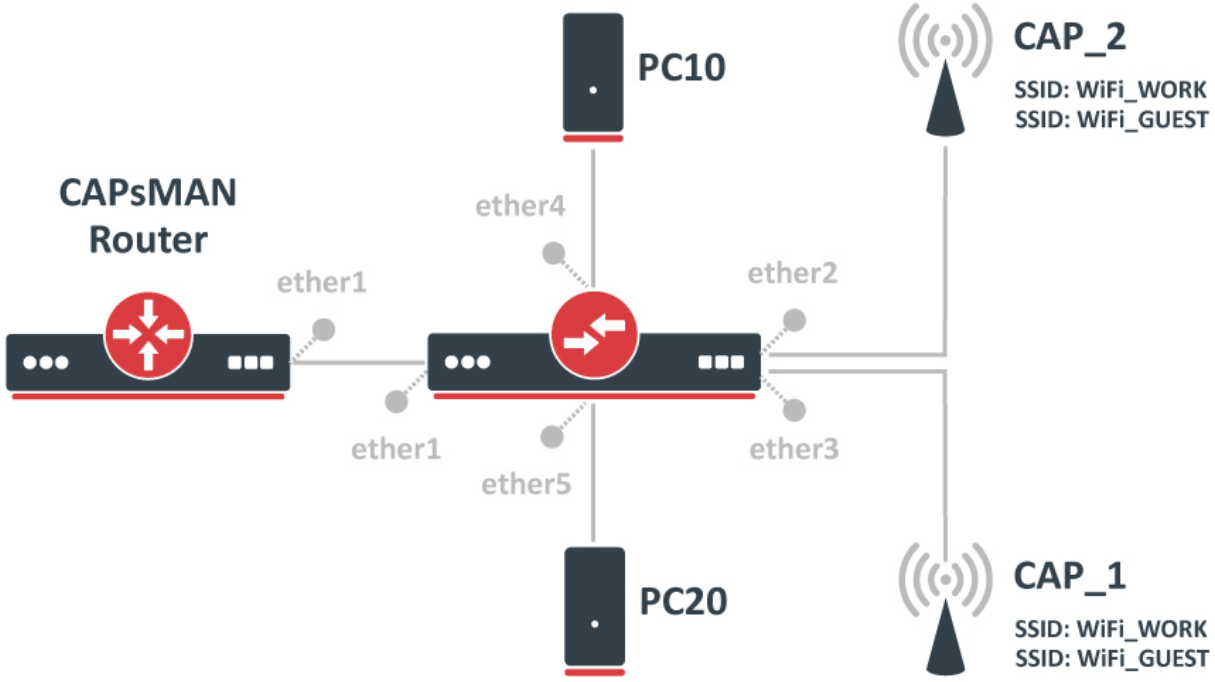
CAPsMAN with VLANs

Summary

It is possible to create a centralized Access Point management setup for a home or office environment that is scalable to many Access Points, such a setup is quite easy to configure and has been explained in the Simple CAPsMAN setup guide, but for more complex setups VLANs might be required. CAPsMAN has the functionality to assign a certain VLAN ID under certain conditions. This guide will provide an example on how to assign a VLAN ID to Wireless packets based on the AP, to which a wireless client connects to. CAPsMAN with VLANs can be achieved either by using Local Forwarding Mode or CAPsMAN Forwarding Mode, the Local Forwarding Mode will provide the possibility to use a switch between your APs and CAPsMAN router to switch packets (to achieve larger throughput), while CAPsMAN Forwarding Mode should be used when all traffic should always be forwarded to the CAPsMAN router (in most cases to filter packets).

In this example, we are going to assign all our Wireless clients to **VLAN10**, if they connect to **WiFi_WORK**, and going to assign Wireless clients to **VLAN20**, if they connect to **WiFi_GUEST**. We are going to use Virtual APs along with CAPsMAN to create multiple SSIDs for our Wireless clients to connect to while using a single physical device. An example of how to use a single SSID for a single physical device will also be shown by using CAPsMAN provisioning rules.

Using Local Forwarding Mode



In Local Forwarding Mode, the CAPsMAN router is distributing the configuration across all CAPs that are being provisioned by the CAPsMAN router. In Local Forwarding Mode traffic is not required to be sent to the CAPsMAN router, rather it can be sent to a different router without involving the CAPsMAN router when forwarding traffic. This mode allows you to tag traffic to a certain VLAN ID before it is sent to your network from your Wireless client, which adds the possibility to use a switch to limit certain VLAN IDs to certain ports. In Local Forwarding Mode traffic is not encapsulated with a special CAPsMAN header, which can only be removed by a CAPsMAN router.

CAPsMAN Router:

- Create appropriate CAP configurations for each VLAN

```
/caps-man configuration
add country=latvia datapath.local-forwarding=yes datapath.vlan-id=10 datapath.vlan-mode=use-tag
name=Config_WORK security.authentication-types=wpa-psk,wpa2-psk \
    security.passphrase=secret_work_password ssid=WiFi_WORK
add country=latvia datapath.local-forwarding=yes datapath.vlan-id=20 datapath.vlan-mode=use-tag
name=Config_GUEST security.authentication-types=\
    wpa-psk,wpa2-psk security.passphrase=secret_guest_password ssid=WiFi_GUEST
```

- We are going to create a single CAPsMAN provisioning rule to create the **WiFi_WORK** and the **WiFi_GUEST** SSIDs on a single device, each connected CAP is going to create these SSIDs automatically

```
/caps-man provisioning
add action=create-dynamic-enabled master-configuration=Config_WORK slave-configurations=Config_GUEST
```

You can create even more Virtual APs by adding multiple slave-configurations. That requires multiple CAPsMAN configurations that were created earlier.

- For security reasons, limit the CAPsMAN to a single interface

```
/caps-man manager interface
set [ find default=yes ] forbid=yes
add disabled=no interface=ether1
```

- Enable the CAPsMAN manager

```
/caps-man manager
set enabled=yes
```

- Setup DHCP Server for each VLAN

```
/interface vlan
add interface=ether1 name=VLAN10 vlan-id=10
add interface=ether1 name=VLAN20 vlan-id=20
/ip address
add address=192.168.10.1/24 interface=VLAN10
add address=192.168.20.1/24 interface=VLAN20
/ip pool
add name=dhcp_pool10 ranges=192.168.10.2-192.168.10.254
add name=dhcp_pool20 ranges=192.168.20.2-192.168.20.254
/ip dhcp-server
add address-pool=dhcp_pool10 disabled=no interface=VLAN10 name=dhcp10
add address-pool=dhcp_pool20 disabled=no interface=VLAN20 name=dhcp20
/ip dhcp-server network
add address=192.168.10.0/24 dns-server=8.8.8.8 gateway=192.168.10.1
add address=192.168.20.0/24 dns-server=8.8.8.8 gateway=192.168.20.1
```

Switch:

In this example, we are going to be using Bridge VLAN Filtering to filter unknown VLANs and to assign other devices to the same networks. Some devices are capable of offloading this to the built-in switch chip, check Basic VLAN switching guide to see how to configure it on different types of devices.

- Setup Bridge VLAN Filtering

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
add bridge=bridge1 interface=ether4 pvid=10
add bridge=bridge1 interface=ether5 pvid=20
/interface bridge vlan
add bridge=bridge1 tagged=ether1,ether2,ether3 untagged=ether4 vlan-ids=10
add bridge=bridge1 tagged=ether1,ether2,ether3 untagged=ether5 vlan-ids=20
```

In this example, untagged traffic is going to be used to communicate between CAPs and CAPsMAN Router. By default, if PVID is not changed, untagged traffic is going to be forwarded between ports that have the same PVID value set (including the default PVID).

CAPs:

- Create a bridge and assign a port to it, that is connected to the CAPsMAN Router

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
```

- Enable CAP mode on the AP, and make sure you specify to use the newly created bridge

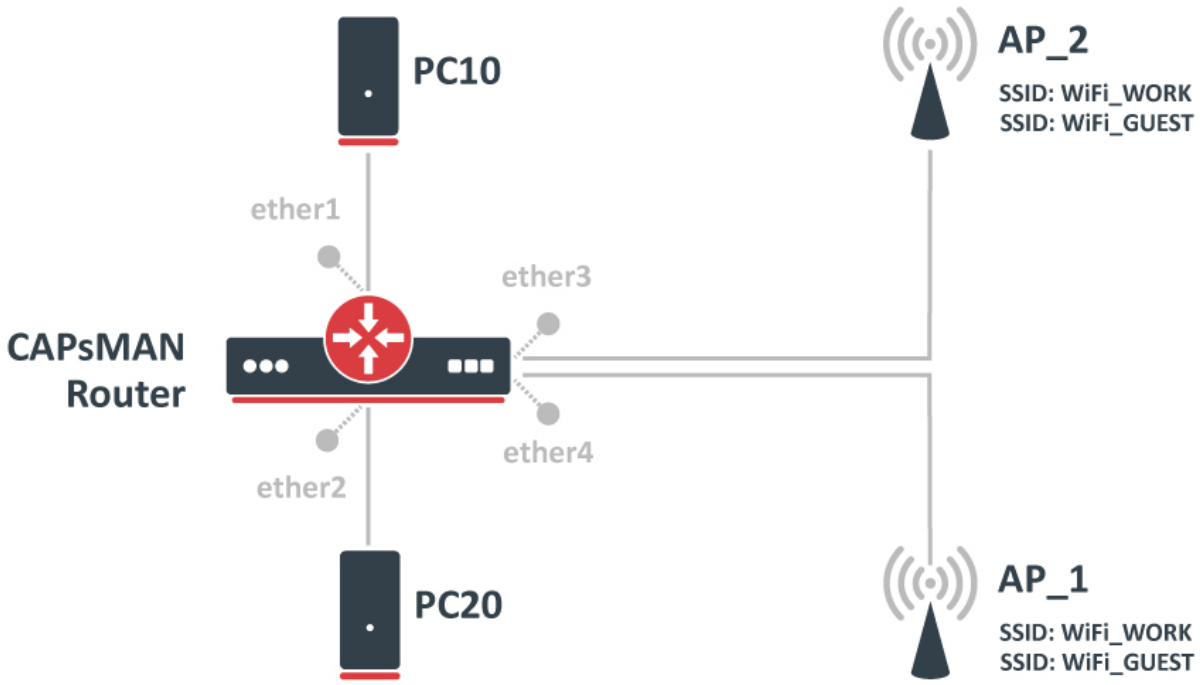
```
/interface wireless cap
set bridge=bridge1 discovery-interfaces=bridge1 enabled=yes interfaces=wlan1
```

- After CAPs are successfully connected to the CAPsMAN Router, the wlan1 (SSID **WiFi_WORK**) and a newly created virtual wlan5 (SSID **WiFi_GUEST**) interfaces get dynamically added as bridge ports. The VLAN is assigned for a wireless interface and as a result, all data coming from wireless gets tagged and only data with this tag will be sent out over wireless. A bridge vlan-filtering can be disabled if additional VLAN managing and controlling is not needed. The associated VLAN can be seen with a port VLAN ID (PVID) property.

```
[admin@CAP_1] /interface bridge port pr
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                  BRIDGE                  HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-
COST    HORIZON
 0   H ether1                     bridge1                 yes    1     0x80         10
10       none
 1  D  wlan1                      bridge1                       10     0x80         10
10       none
 2  D  wlan5                      bridge1                       20     0x80         10
10       none
```

That is it! Connect Wireless clients to your APs and check connectivity.

Using CAPsMAN Forwarding Mode



In CAPsMAN Forwarding Mode all traffic that is coming from a CAP is encapsulated with a special CAPsMAN header, which can only be removed by a CAPsMAN router, this means that a switch will not be able to distinguish the VLAN ID set by the CAP since the VLAN tag is also going to be encapsulated. This mode limits the possibility to divert traffic in Layer2 networks, but gives you the possibility to forward traffic from each CAP over Layer3 networks for a distant CAPsMAN router to process the traffic, this mode is useful when you want to control multiple CAPs in remote locations, but want to use a central gateway.

CAPsMAN router:

- Setup Bridge VLAN filtering to limit interfaces to appropriate VLANs

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1 pvid=10
add bridge=bridge1 interface=ether2 pvid=20
/interface bridge vlan
add bridge=bridge1 tagged=bridge1 untagged=ether1 vlan-ids=10
add bridge=bridge1 tagged=bridge1 untagged=ether2 vlan-ids=20
```

CAPsMAN will attach CAP interfaces to the bridge and automatically will add appropriate entries to the bridge VLAN table

Note: CAPsMAN will attach CAP interfaces to the bridge and automatically will add appropriate entries to the bridge VLAN table. This feature is available starting with RouterOS v6.43

- Create appropriate CAP configurations for each VLAN

```
/caps-man configuration
add country=latvia datapath.bridge=bridge1 datapath.vlan-id=10 datapath.vlan-mode=use-tag name=Config_WORK
security.authentication-types=wpa-psk,wpa2-psk \
    security.passphrase=secret_work_password ssid=WiFi_WORK
add country=latvia datapath.bridge=bridge1 datapath.vlan-id=20 datapath.vlan-mode=use-tag name=Config_GUEST
security.authentication-types=wpa-psk,wpa2-psk \
    security.passphrase=secret_guest_password ssid=WiFi_GUEST
```

- We are going to create a single CAPsMAN provisioning rule to create the **WiFi_WORK** and the **WiFi_GUEST** SSIDs on a single device, each connect CAP is going to create these SSIDs automatically

```
/caps-man provisioning
add action=create-dynamic-enabled master-configuration=Config_WORK slave-configurations=Config_GUEST
```

You can create even more Virtual APs by adding multiple slave-configurations. That requires multiple CAPsMAN configurations that were created earlier.

- For security reasons, limit the CAPsMAN to interfaces. to which CAPs are going to be connected

```
/caps-man manager interface
set [ find default=yes ] forbid=yes
add disabled=no interface=ether3
add disabled=no interface=ether4
```

- Enable the CAPsMAN manager

```
/caps-man manager
set enabled=yes
```

- Setup DHCP Server for each VLAN

```
/interface vlan
add interface=bridge1 name=VLAN10 vlan-id=10
add interface=bridge1 name=VLAN20 vlan-id=20
/ip address
add address=192.168.10.1/24 interface=VLAN10
add address=192.168.20.1/24 interface=VLAN20
/ip pool
add name=dhcp_pool10 ranges=192.168.10.2-192.168.10.254
add name=dhcp_pool20 ranges=192.168.20.2-192.168.20.254
/ip dhcp-server
add address-pool=dhcp_pool10 disabled=no interface=VLAN10 name=dhcp10
add address-pool=dhcp_pool20 disabled=no interface=VLAN20 name=dhcp20
/ip dhcp-server network
add address=192.168.10.0/24 dns-server=8.8.8.8 gateway=192.168.10.1
add address=192.168.20.0/24 dns-server=8.8.8.8 gateway=192.168.20.1
```

CAPs:

- Enable CAP mode on each AP, specify which interface is connected to the CAPsMAN router

```
/interface wireless cap
set discovery-interfaces=ether1 enabled=yes interfaces=wlan1
```

- After CAPs are successfully connected to the CAPsMAN Router, two CAP interfaces will be dynamically created on the CAPsMAN Router. Both of these interfaces will get dynamically added as bridge ports on the same CAPsMAN Router due to explicitly selecting the bridge interface with datapath.bridge=bridge1 and using the default CAPsMAN Forwarding Mode datapath.local-forwarding=no. Because of using a bridge with enabled vlan-filtering, both CAP interfaces will also show up in a bridge VLAN table.

```
[admin@CAPsMAN_Router] /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                         BRIDGE                        HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-
COST    HORIZON
 0     ether1                            bridge1                       yes   10     0x80
10                 10       none
 1     ether2                            bridge1                       yes   20     0x80
10                 10       none
 2  D  cap16                             bridge1                             10     0x80
10                 10       none
 3  D  cap17                             bridge1                             20     0x80
10                 10       none
[admin@CAPsMAN_Router] /interface bridge vlan print
Flags: X - disabled, D - dynamic
 #   BRIDGE                           VLAN-IDS  CURRENT-TAGGED                      CURRENT-
UNTAGGED
 0 D bridge1                          1
bridge1
 1   bridge1                          10        cap16
ether1
 2   bridge1                          20        cap17                               ether2
```

That is it! Connect Wireless clients to your APs and check connectivity.

Case studies

Without Virtual APs

Not everyone wants to create Virtual APs since that does decrease the total throughput. If you want to use multiple devices to create multiple SSIDs, then it is possible to assign a certain configuration on a CAP based on its identity. To achieve this you should use CAPsMAN provisioning rules along with RegEx expressions. In this example we are going to assign the **Config_WORK** configuration to CAPs that have identity set to "'**AP_WORK_***" and we are going to assign the **Config_GUEST** configuration to CAPs that have identity set to "**AP_GUEST_***". To do this, you simply need to change the CAPsMAN provisioning rules.

- Remove any existing provisioning rules

```
/caps-man provisioning remove [f]
```

- Create new provisioning rules that will assign appropriate configuration on a CAP based on its identity

```
/caps-man provisioning
add action=create-dynamic-enabled identity-regexp=^AP_GUEST_ master-configuration=Config_GUEST
add action=create-dynamic-enabled identity-regexp=^AP_WORK_ master-configuration=Config_WORK
```

Don't forget to set a proper identity on the CAPs since CAPsMAN is going to assign appropriate configuration on the APs based on it's identity.

WifiWave2 Troubleshooting(viewing restricted)

Overview

This article explains common WifiWave2 log messages, and possible issues and misconfiguration.

Terminology

AP - access point

Station - wireless client

Association -

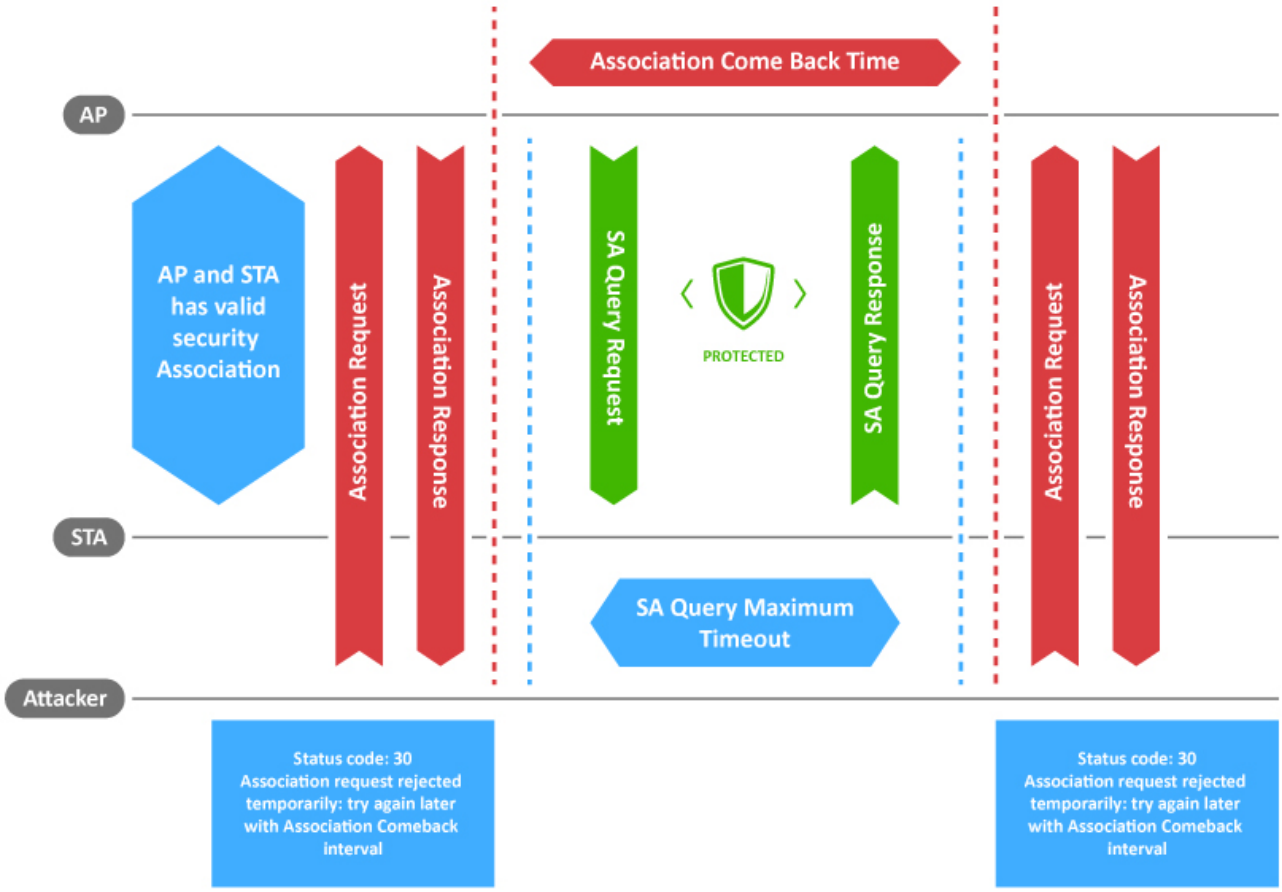
Authentication -

Wireless logs

Station mode

AP mode

SA query timeout



In short - this means that the client left the router range, AP sent an information request in order to check, if the client is still present and did not receive an answer. Thus - the client has left the range. Completely normal debug log message - this is not an error or warning. Just an informational message.

In greater detail:

SA Query Timeout is a normal part of wireless behavior. It is a security feature.

SA Query is triggered in the following scenario:

1) On AP there is a valid security association for the station

2) AP receives an Association request from an already associated station

3) AP responds with Association request rejected - "Association Comeback interval" Status Code:30. - this is done in order for AP to understand if the association request came from an attacker, or if it came from a station that got out of range, and was not able to disassociate beforehand.

4) AP sends SA Query Request to the station. Using original encryption that was used with the client beforehand. If the client sends SA Query Response, it will mean that the initial association request came from Attacker.

5) If the Client doesn't give SA Query response, it means that the real client got disconnected, or rather was out of range, and didn't disassociate from AP properly, and restarted association to AP - no attacker is present in this case. And at this point, you will see SA Query Timeout in the log.

Unless you see excessive amounts of this message in the log, it is not a problem, and even if you do, it does not necessarily mean that there is a bug in RouterOS, it can appear due to external factors as well.

Spectral scan

Introduction

The spectral scan can scan all frequencies supported by your wireless card, and plot them directly in the console. The exact frequency span depends on the card. Allowed ranges on r52n: [4790; 6085], [2182; 2549].

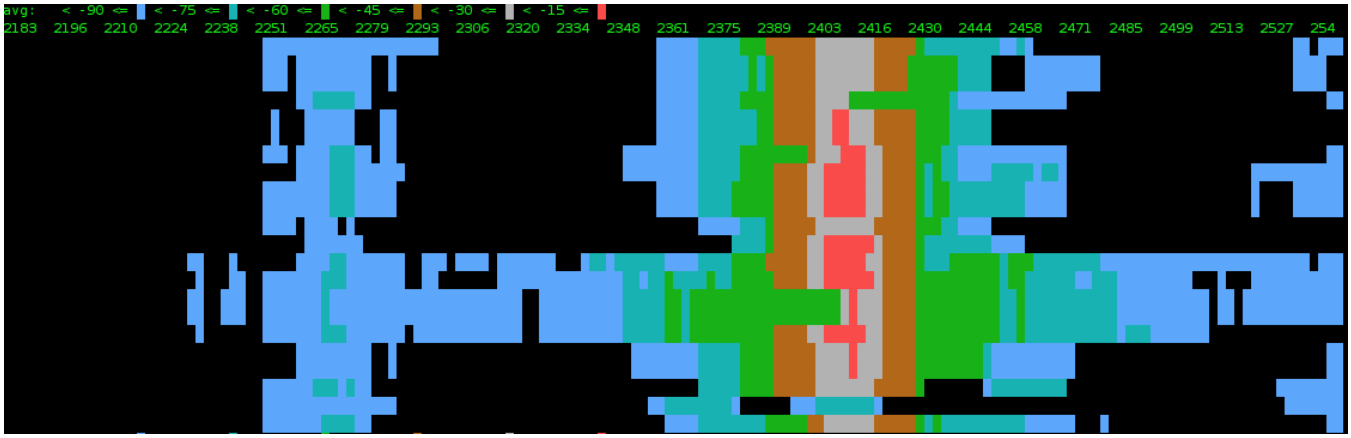
A wireless card can generate 4us long spectral snapshots for any 20mhz wide channel. This is considered a single spectral sample.

To improve data quality spectrum is scanned with 10mhz frequency increments, which means doubled sample coverage at each specific frequency (considering 20mhz wide samples).

> Currently, is NOT supported for Atheros 802.11ac chips (e.g. QCA98xx, IPQ-4018). See https://mikrotik.com/products determine the wireless chip on your device.

Console

Spectral History



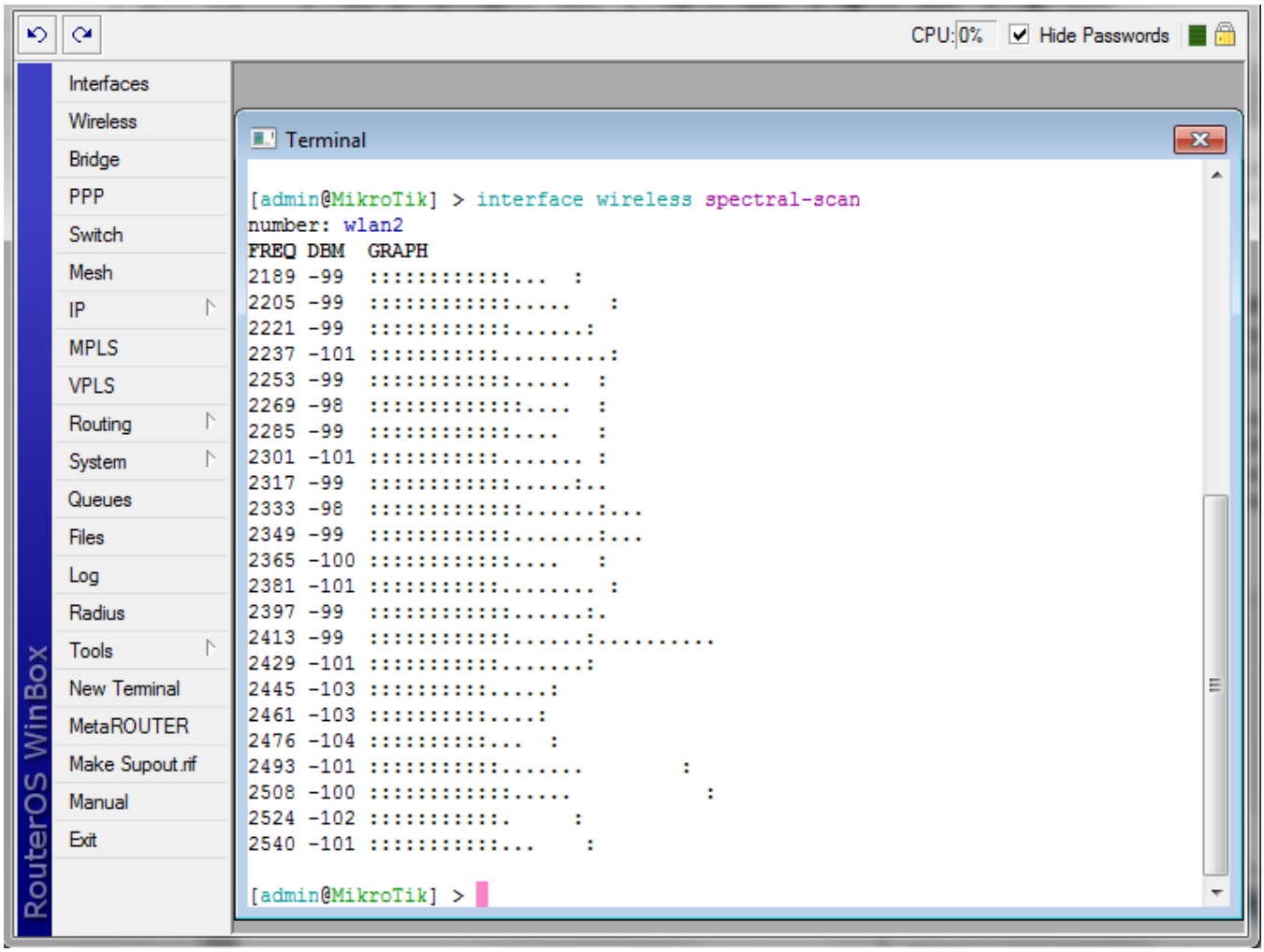

```
/interface wireless spectral-history <wireless interface name>
```

Plots spectrogram. Legend and frequency ruler is printed every 24 lines. Numbers in the ruler correspond to the value at their leftmost character position. Power values that fall in different ranges are printed as different colored characters with the same foreground and background color, so it is possible to copy and paste the terminal output of this command.

- *value* -- select value that is plotted on the output. 'interference' is special as it shows detected interference sources (affected by the 'classify-samples' parameter) instead of power readings, and cannot be made audible;
- *interval* -- interval at which spectrogram lines are printed;
- *duration* -- terminate command after a specified time. default is indefinite;
- *buckets* -- how many values to show in each line of a spectrogram. This value is limited by the number of columns in the terminal. It is useful to reduce this value if using 'audible';
- *average-samples* -- Number of 4us spectral snapshots to take at each frequency, and calculate average and maximum energy over them. (default 10);

- *classify-samples* -- Number of spectral snapshots taken at each frequency and processed by the interference classification algorithm. Generally, more samples give more chance to spot certain types of interference (default 50);
- *range* --
  - 2.4ghz - scan the whole 2.4ghz band;
  - 5ghz - scan the whole 5ghz band;
  - current-channel - scan current channel only (20 or 40 MHz wide);
  - range - scan specific range;
- *audible=yes* -- play each line as it is printed. There is a short silence between the lines. Each line is played from left to right, with higher frequencies corresponding to higher values in the spectrogram.

Spectral Scan




```
/interface wireless spectral-scan <wireless interface name>
```

Continuously monitor spectral data. This command uses the same data source as 'spectral-history', and thus shares many parameters.

Each line displays one spectrogram bucket -- frequency, the numeric value of power average, and a character graphic bar. A bar shows average power value with ':' characters and average peak hold with '.' characters. Maximum is displayed as a lone floating ':' character.

- *show-interference* -- add a column that shows detected interference sources;

Types of possibly classified interference:

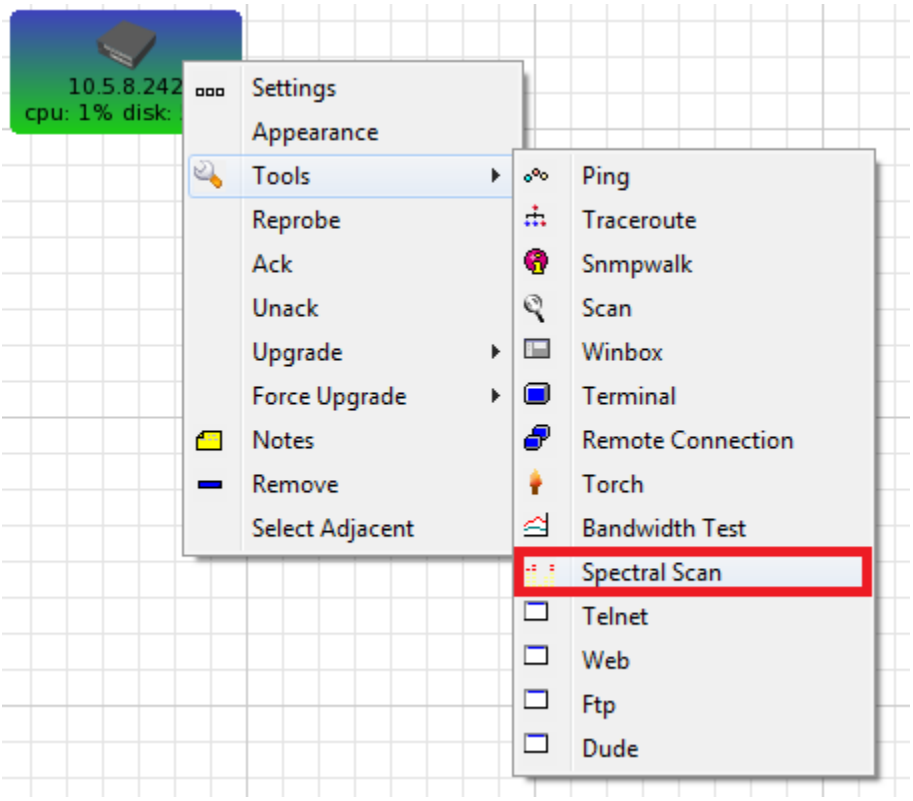
- Bluetooth-headset
- Bluetooth-stereo
- cordless-phone
- microwave-oven

- CWA
- video-bridge
- wifi

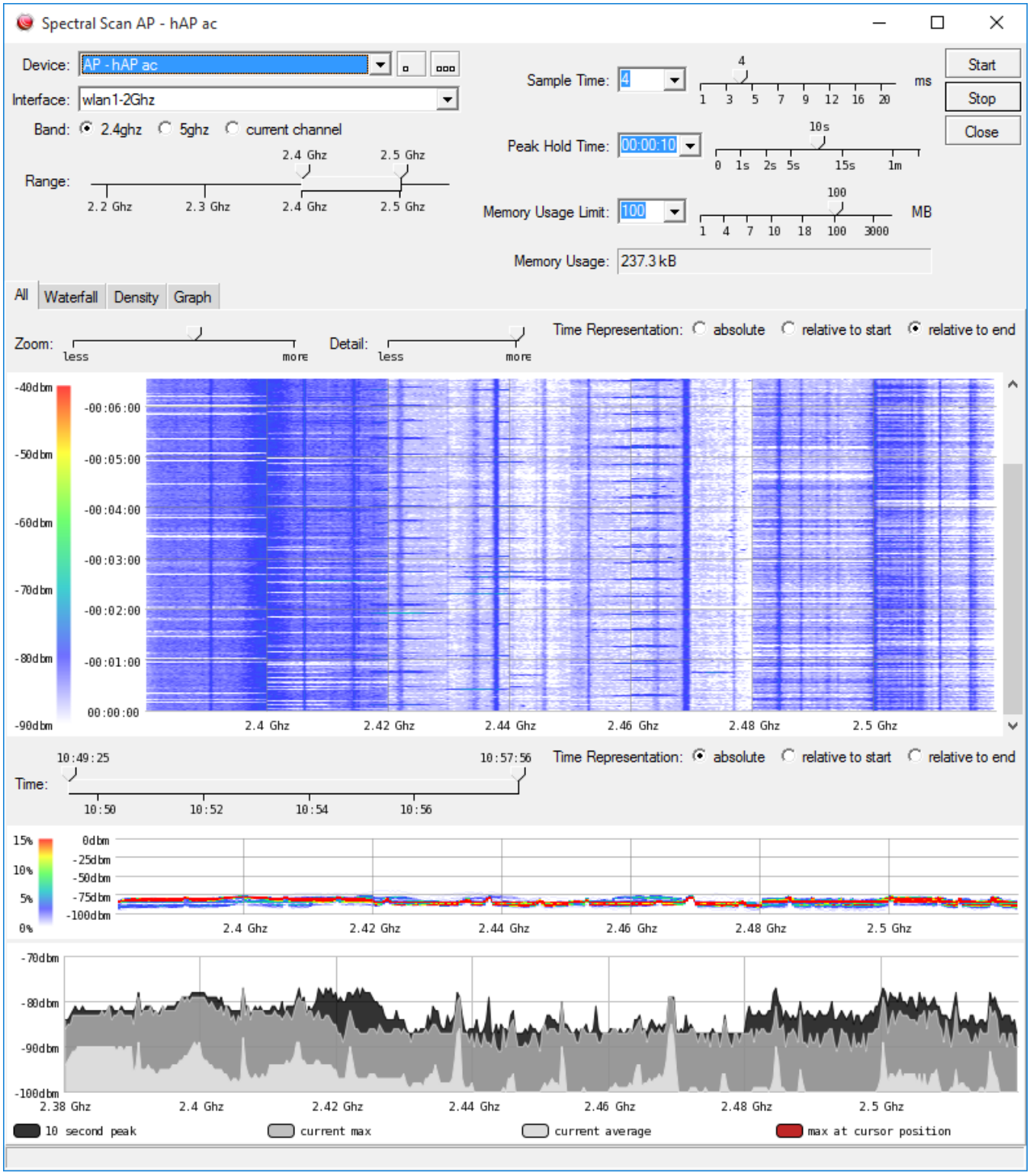
The Dude

The Dude is a free network monitoring and management program by MikroTik. You can download it here.

The Dude has a built-in capability to run graphical Spectral Scan from any of your RouterOS devices with a supported wireless card. Simply select this device in your Dude map, right click and choose Tools -> Spectral Scan.



This will bring up the Spectral Scan GUI with various options and different view modes:


Spectral Scan AP - hAP ac
Device: AP - hAP ac
Interface: wlan1-2Ghz
Band: 2.4ghz 5ghz current channel
Range: 2.2 Ghz 2.3 Ghz 2.4 Ghz 2.5 Ghz
Sample Time: 4 ms
Peak Hold Time: 00:00:10
Memory Usage Limit: 100 MB
Memory Usage: 237.3 kB
Start
Stop
Close
All
Waterfall
Density
Graph
Zoom: less more
Detail: less more
Time Representation: absolute relative to start relative to end
-40dbm
-50dbm
-60dbm
-70dbm
-80dbm
-90dbm
-00:06:00
-00:05:00
-00:04:00
-00:03:00
-00:02:00
-00:01:00
00:00:00
2.4 Ghz
2.42 Ghz
2.44 Ghz
2.46 Ghz
2.48 Ghz
2.5 Ghz
Time: 10:49:25 10:57:56
10:50
10:52
10:54
10:56
15%
10%
5%
0%
0dbm
-25dbm
-50dbm
-75dbm
-100dbm
-70dbm
-80dbm
-90dbm
-100dbm
2.38 Ghz
10 second peak
current max
current average
max at cursor position

Internet of Things

In This Section:

Bluetooth

Summary

Bluetooth is a short-range wireless technology that allows broadcasting the data over specific Bluetooth channels.

There are 40 unique bands (channels) and each band has a 2 MHz separation. 37, 38, and 39 channels are used for primary advertising, and 0-36 are used for data transmission.

During the advertising process, the BLE advertising packet is broadcasted. This packet contains the Preamble, Access Address, PDU and CRS fields.

The Preamble and Access Address fields help the receiver detect frames. CRS field is used to check errors. PDU consists of PDU Header and PDU Payload. PDU defines the packet itself.

PDU Header contains information about the PDU type. Based on the type, the payload fields may differ.

For example, when PDU type is ADV_NONCONN_IND → PDU Payload consists of "AdvA" (a field that contains information about the advertiser's address) and "AdvData" (a field that contains data information) fields:

1 octet = 1 byte = 8 bits

Preamble	1 octet
Access-Address	4 octets
PDU	• PDU Header = 2 octets • PDU Payload = AdvA (6 octets)+AdvData (0...31 octets)
CRS	3 octets

There are different PDU types:

- ADV_IND (where payload consists of AdvA [6octets] + AdvData [0-31 octets] and which is used for connectable, scannable undirected advertising);
- ADV_NOCONN_IND (where payload consists of AdvA [6octets] + AdvData [0-31 octets] and which is used for non-connectable, non-scannable undirected advertising);
- ADV_SCAN_IND (where payload consists of AdvA [6octets] + AdvData [0-31 octets] and which is used for scannable, undirected advertising);
- SCAN_REQ (where payload consists of ScanA [6octets] + AdvA [6octets], where ScanA field contains scanner's address and AdvA contains advertiser's address, and which is used for requesting SCAN_RSP response);
- SCAN_RSP (where payload consists of AdvA [6octets] + ScanRspData [0-31 octets], where ScanRspData can contain any data from the advertiser's host and which is used to respond to a SCAN_REQ request);
- ADV_DIRECT_IND (where payload consists of AdvA [6octets] + TargetA [6octets], where TargetA is the device address field to which the PDU is addressed, and which is used for connectable, directed advertising);
- etc

You can find more information about the packet structure over here (Bluetooth specifications).

The main application for the Bluetooth interface in RouterOS is to monitor Bluetooth advertising packets (scanner feature) that are broadcasted by other devices (like for example, Bluetooth tags) or broadcast advertising packets (advertiser feature).

Configuration

Sub-menu: `/iot bluetooth`

note: **iot** package is required.

note: Check your device's specifications page to make sure that the Bluetooth is supported by the unit.

IoT package is available with RouterOS version 6.48.3. You can get it from our download page - under "Extra packages".

Devices

In this menu you can check and set general Bluetooth chip parameters:

```
/iot bluetooth print
Columns: NAME, PUBLIC-ADDRESS, RANDOM-STATIC-ADDRESS, ANTENNA
  #  NAM  PUBLIC-ADDRESS     RANDOM-STATIC-ADD  ANTENNA
  0  bt1  00:00:00:00:00:00  F4:4E:E8:04:77:3A  internal
/iot bluetooth set
```

note: Public address is the IEEE registered, permanent address. This address can not be changed. In the "print" example above, the device does not have a public address assigned (all octets are set to 0).

Configurable settings are shown below:

Property	Description
antenna (*string*; Default: internal)	Choose whether to use an internal or an external Bluetooth antenna
name (*string*; Default:)	Descriptive name of Bluetooth chip/interface
random-static-address (*MAC address*; Default:)	A user-configurable address for the Bluetooth chip

You can monitor chip stats with the command:

```
/iot bluetooth print stats
Columns: NAME, RX-BYTES, TX-BYTES, RX-ERRORS, TX-ERRORS, RX-EVT, TX-CMD, RX-ACL, TX-ACL
  #  NAM  RX-BYTE  TX-  R  T  RX-EV  TX  R  T
  0  bt1  1857835  235  0  0  46677  45  0  0
```

Advertisers

In this menu, it is possible to set up the Bluetooth chip to broadcast advertising packets. You can check and set advertiser settings with the commands:

```
/iot bluetooth advertisers print
Flags: X - DISABLED
Columns: DEVICE, MIN-INTERVAL, MAX-INTERVAL, OWN-ADDRESS-TYPE, CHANNEL-MAP, AD-SIZE
#   DEVICE  MIN-INTERVAL  MAX-INTERVAL  OWN-ADDRESS-TYPE  CHANNEL-MAP  AD-SIZE
0 X bt1     1280ms        2560ms        random-static             37        0
                                                                  38
                                                                  39
/iot bluetooth advertisers set
```

Configurable settings are shown below:

Property	Description
ad-structures (*string*; Default:)	Choose a pre-configured structure for the advertisement packets. For more information see the "AD structures" section.
channel-map (*37* / *38* / *39*; Default: 37, 38, 39)	Channels used for advertising.

disabled (*yes* / *no*; Default: **yes**)	An option to disable or enable the Bluetooth chip to broadcast advertising packets.
max-interval (*integer:20..10240;* Default: **2560 ms**)	The maximal interval for broadcasting advertising packets.
min-interval (*integer:20..10240;* Default: **1280 ms**)	The minimal interval for broadcasting advertising packets.
own-address-type (*public* / *random-static* / *rpa-fallback-to-public* / *rpa-fallback-to-random*; Default: **random-static**)	The MAC address that is going to be used in the advertising packet's payload: • public → To use the IEEE registered, permanent address. • random-static → To use user-configurable address (will be changed on the next power-cycle). • rpa-fallback-to-public → To use Resolvable Random Private Address (RPA) that can only be resolved if the receiver has our Identity Resolving Key (IRK). If RPA can not be generated, the public address will be used instead. • rpa-fallback-to-random → Same as "rpa-fallback-to-public" but if RPA can not be generated, the random-static address will be used instead.

***note**:* Advertising packets will be broadcasted each *min-interval* **<= X <=** *max-interval* milliseconds.

AD structures

This section allows you to define the payload for the advertising packets that are going to be broadcasted by the Bluetooth chip.

Currently, only 4 types are supported: 0x08 "Shortened Local Name"; 0x09 "Complete Local Name"; 0xFF "Manufacturer Specific Data"; "Service Data"

You can check and set "AD structures" settings with the commands:

```
/iot bluetooth advertisers ad-structures print
Columns: NAME, TYPE, DATA
#  NAME  TYPE              DATA
0  test  short-local-name  TEST
/iot bluetooth advertisers ad-structures set
```

Configurable properties are shown below:

Property	Description
data (*string*; Default:)	Define advertising packet's AdvData part of the payload
name (*string*; Default:)	Descriptive name of AD structure
type (complete-local-name | manufacturer-data | short-local-name | service-data; Default:)	An option to set AD structure's type: • 0x08 "Shortened Local Name" • 0x09 "Complete Local Name" • 0xFF "Manufacturer Specific Data" • 0x20 "Service Data 32-bit"

If, for example, the "Shortened Local Name" type is chosen and the "data" field is configured with "TEST" → AdvData part of the payload is going to look like this:

05 08 54 45 53 54 (hexadecimal format)

, where the first octet (05) shows the number of bytes to follow (5 bytes) and the second octet (08) shows the type (Shortened Local Name). 3d, 4th, 5th and 6th (and etc) octets are the "data" [54 (hex)=**T** (ASCII), 45 (hex)=**E** (ASCII), 53 (hex)=**S** (ASCII), 54 (hex)=**T** (ASCII)].

The same applies to the "Complete Local Name" type. Only the second octet in the AdvData payload is going to differ and will be set to 09.

For the "Manufacturer Specific Data" type, you will need to configure the "data" field in the hexadecimal format. The second octet for this type is going to be set to FF.

Connections

Availible starting with v**7.12beta9**.

Currenetly, only "`central`" role is supported. "`Pheriperal device`" role, "`pairing`" and "`encryption`" options are not supported.

Availible sections are:

Section	Description
async-data	used to view subscribed data.
characteristics	used to view all supported characteristics of the device.
connect	used to connect to the device that is in the `connactable` state.
disconnect	used to disconnect from the device.
read	used to read characteristics values.
write	used to write characteristics values.
subscribe	used to subscribe to a charasteristic value.
unsubscribe	used to unsubscribe from a charasteristic value.

In order to connect to a Bluetooth device that is in the `connactable` state, use the command (where `pdev` is the device address):

```
/iot bluetooth connections connect pdev=DC:2C:6E:0F:C0:3D
```

To connect to TG-BT5-IN/OUT tags, make sure to put it into the `connactable` state by applying a magnet to the magnetic switch.

To view an already established connection:

```
/iot bluetooth connections print
```

To view device characteristics:

```
/iot bluetooth connections characteristics print
Columns: PDEV, NAME, UUID
 #  PDEV               NAME                            UUID
 0  DC:2C:6E:0F:C0:3D  Service Changed                 2a05
 1  DC:2C:6E:0F:C0:3D  Database Hash                   2b2a
 2  DC:2C:6E:0F:C0:3D  Client Supported Features       2b29
 3  DC:2C:6E:0F:C0:3D  Device Name                     2a00
 4  DC:2C:6E:0F:C0:3D  Appearance                      2a01
...
...
...
```

To read a specific characteristic, specify the `pdev` address and the `uuid`:

```
/iot bluetooth connections read pdev=DC:2C:6E:0F:C0:3D uuid=2a00
```

Scanners

In this menu, you can set up the scanner settings for the Bluetooth chip. When disabled, the device is no longer able to receive advertising reports. When enabled, you can monitor advertising reports in the "Advertising reports" tab (which will be explained later in the guide). You can check and set scanner settings with the commands:

```
/iot bluetooth scanners print
Flags: X - DISABLED
Columns: DEVICE, TYPE, INTERVAL, WINDOW, OWN-ADDRESS-TYPE, FILTER-POLICY, FILTER-DUPL
ICATES
#   DEVICE  TYPE     INTERVAL  WINDOW  OWN-ADDRESS-TYPE  FILTER-POLICY  FIL
0 X bt1     passive  10ms      10ms    random-static     default        off
/iot bluetooth scanners set
```

Configurable properties are shown below:

Property	Description
disabled (*yes* / *no*; Default: **no**)	An option to disable or enable the Bluetooth chip to receive advertising reports.
filter-duplicates (keep-newest | keep-oldest | keep-unique | off; Default: **off**)	An option to discard duplicate advertisements from the same advertiser: • keep-newest → Keeps the newest report (discards the oldest). Only the newest PDU from a single AdvA will be kept. • keep-oldest → Keeps the oldest report (discards the newest). Only the oldest PDU from a single AdvA will be kept. This type of PDU filtering happens at the controller level and as such it's the most efficient (energy /bandwidth-wise) method of duplicate filtering. • keep-unique → Only displays advertisements that have a unique payload. Meaning, if 1+ identical payloads (AdvData) are found, only the first payload is going to be displayed, while the "clones" are discarded/ignored. • off → Duplicates are not discarded. All PDUs with the same AdvA will be kept. ⓘ A duplicate advertising report is an advertising report sent from the same device address. The actual data ("AdvData" part of the payload) may change/differ and it is not considered significant when determining du plicate advertising reports. Meaning that, for example, if the Bluetooth interface receives 10 payloads (payload after payload with a 1-second interval) from the same tag: • if you are using the "keep-oldest" setting → Bluetooth interface will only display the first payload received (9 follow-up payloads will be filtered out) from that tag. • if you are using the "keep-newest" setting → Bluetooth interface will only display the last payload received (each follow-up payload will rewrite the previous one) from that tag.
filter-policy (default | whitelist / *no*; Default: **default**)	An option to set up a filtering policy (controller-level advertisement filtering): • default → When this policy is enabled, the scanner will only accept ADV_IND, ADV_NOCONN_IND, ADV_SCAN_IND, SCAN_RSP, and ADV_DIRECT_IND (where TargetA is the scanner's own Bluetooth address) PDU types. • whitelist → When this policy is enabled, the scanner will only accept ADV_IND, ADV_NOCONN_IND, ADV_SCAN_IND, SCAN_RSP PDU types that are broadcasted by the advertiser, whose address is configured in the "Whitelist" section, and ADV_DIRECT_IND type PDU (where TargetA is the scanner's own Bluetooth address).
interval (*integer:3..10240;* Default: **10 ms**)	Time after which scanner will start scanning the next advertisement channel.
own-address-type (*public* / *random-static* / *rpa-fallback-to-public* / *rpa-fallback-to-random*; Default: **random-static**)	Address type used in scan requests (if active scanning type is used): • public → To use the IEEE registered, permanent address. • random-static → To use user-configurable address (will be changed on the next power-cycle). • rpa-fallback-to-public → To use Resolvable Random Private Address (RPA) that can only be resolved with our Identity Resolving Key (IRK). If RPA can not be generated, the public address will be used instead. • rpa-fallback-to-random → Same as "rpa-fallback-to-public" but if RPA can not be generated, the random-static address will be used instead.

type (*active* | *passive;* Default: **passive**)	Defines the scanner's type: • active → Scanner can send scan requests if it receives a scannable advertisement. The scanner can send a SCAN_REQ in order to acquire a SCAN_RSP response. • passive → Scanner will only listen for advertisements, no data (e.g. scan requests) will be sent.
window (*integer:3..10240;* Default: **10 ms**)	The time that the scanner will spend scanning a single advertisement channel.

For example, if the scanner interval is set to 20ms, it means that only after 20ms, the device will begin scanning the next channel in line. If the scanner window is set to 10ms, it means that the device will scan each channel only during that 10ms window. Meaning, it will scan channel 37 for 10ms (window time) and begin scanning the next channel after 10 more ms (20ms[interval]-10ms[window]). It will take 10ms to scan channel 38, and after 10 more ms, the device will begin scanning channel 39.

Advertising reports

In this section, it is possible to monitor Bluetooth advertising reports (from the nearby broadcasters). The list is limited to 1024 entries (if the list gets full with 1024 entries, each new payload received will overwrite the "oldest" one). You can monitor advertising reports with the command:

```
/iot bluetooth scanners advertisements print
Columns: DEVICE, PDU-TYPE, TIME, ADDRESS-TYPE, ADDRESS, RSSI
 #  DEV  PDU-TYPE        TIME                ADDRES  ADDRESS            RSSI
 0  bt1  adv-noconn-ind  jul/28/2021 09:30:56  public  2C:C8:1B:93:16:49  -24dBm
 1  bt1  adv-noconn-ind  jul/28/2021 09:30:56  random  0B:16:17:9E:7B:EF  -60dBm
```

It is possible to set up a filter for the reports with the command:

```
/iot bluetooth scanners advertisements print where
```

For example, to print reports that are broadcasted by a specific Bluetooth address, use the command:

```
/iot bluetooth scanners advertisements print where address=XX:XX:XX:XX:XX:XX
 # DEVICE    PDU-TYPE       TIME                ADD... ADDRESS                  RSSI     LENGTH DATA
79 bt1       adv-noconn-ind jul/28/2021 09:46:38 public XX:XX:XX:XX:XX:XX        -70dBm         30 02010...
80 bt1       adv-noconn-ind jul/28/2021 09:46:43 public XX:XX:XX:XX:XX:XX        -67dBm         30 02010...
81 bt1       adv-noconn-ind jul/28/2021 09:46:44 public XX:XX:XX:XX:XX:XX        -70dBm         28 1bff0...
82 bt1       adv-noconn-ind jul/28/2021 09:46:48 public XX:XX:XX:XX:XX:XX        -75dBm         30 02010...
```

To show only advertising reports that have RSSI stronger than -30 dBm, use the command:

```
/iot bluetooth scanners advertisements print where rssi > -30
 # DEVICE         PDU-TYPE       TIME                ADDRESS-TYPE ADDRESS                   RSSI     LENGTH
DATA
307 bt1            adv-noconn-ind jul/29/2021 10:11:31 public       2C:C8:1B:93:16:49        -24dBm         22
15ff4f09.>
308 bt1            adv-noconn-ind jul/29/2021 10:11:31 public       2C:C8:1B:93:16:49        -26dBm         22
15ff4f09.>
```

Possible filters (you can filter the list of advertising reports with the help of the following parameters):

Filter	Description
address	Bluetooth advertisers address
address-type	Advertisers address type (for example, public or random)
data	Advertisement data in hex format (AdvData payload)

device	Bluetooth chip/interface name
epoch	Milliseconds since Unix Epoch
filter-comment	Comment of the matching whitelist filter
length	Advertisement data length
pdu-type	Advertisement PDU type
rssi	Signal strength
time	Time of the advertisement packet reception

Whitelist

In this tab, it is possible to configure a whitelist that is going to be used in the filter policy in the "Scanners" section. In other words, an option to specify which Bluetooth addresses are going to be scanned (displayed in the "Advertising reports").

You can view the whitelisted entries with the command:

```
/iot bluetooth whitelist print
Columns: DEVICE, ADDRESS-TYPE, ADDRESS
# DEVICE  ADDRESS-TYPE  ADDRESS
0 bt1     any           *:*:*:*:*:*
```

You can add a new whitelist entry with the command:

```
/iot bluetooth whitelist add
```

Configurable properties:

Property	Description
address (*MAC address*; Default:)	Advertiser's address
address-type (*any* / *public* / *random*; Default:)	Advertiser's address type
comment (*string*; Default:)	Short description of the whitelisted entry
copy-from	An option to copy an entry - for more information check the console documentation
device (*bt1*; Default:)	Select the Bluetooth interface/chip name
disabled (*yes* / *no*; Default:)	An option to disable or to enable the entry

> Only 8 whitelisted entries can be added prior to **7.14beta8** version.
> Starting with **7.14beta8** version, whitelist is no longer limited to 8 entries and address field supports asterisk wildcards.

If, for example, you want to whitelist all MAC addresses that begin with "DC:2C:..." octets, add an entry using wildcard asterisk characters:

```
/iot bluetooth whitelist add address=DC:2C:*:*:*:*
```

Wildcard asterisk can not be used in-between specific octets, like "`AA:*:*:BB:*:*`" (it is an invaldi entry).

Valid entries would be:

- `AA:BB:CC:DD:*:*`
- `AA:BB:CC:DD:EE:*`
- `AA:*:*:*:*:*`

Peripheral Devices

This section displays decoded Eddystone TLM, Eddystone UID, iBeacon and MikroTik Bluetooth payloads. If the "Peripheral Devices" captures other beacon types, they will not be decoded.

You can view a decoded list with a `print detail` command:

```
/iot bluetooth peripheral-devices print detail
 0 address-type=public address=60:C0:BF:87:E2:1C name="60:C0:BF:87:E2:1C" persist=no
   mtik-key="" rssi=-64
   last-data="0201041BFFCD0960C0BF87E21C025B1F198B21AC62CDAE0045FAFEFE057D7B"
   last-seen=2023-08-22 11:20:09 beacon-types=""

 1 address-type=public address=DC:2C:6E:0F:C0:3D name="DC:2C:6E:0F:C0:3D" persist=no
   mtik-key="" rssi=-47
   last-data="0303AAFE1716AAFE00E5B2B98DE4C81C47C2B14E7500000000000000"
   last-seen=2023-08-22 11:20:05 beacon-types=mikrotik,ibeacon,eddystone-uid
   mtik-version=1 mtik-encrypted=no mtik-acc-x=-0.007G mtik-acc-y=-0.015G
   mtik-acc-z=-0.007G mtik-temperature=23.808C mtik-battery=100%
   mtik-uptime=14342160s mtik-flags=""
   ibeacon-uuid="55555555-5555-5555-5555-222222222222" ibeacon-major=1280
   ibeacon-minor=512 ibeacon-rssi-at-1m=-68dBm eddy-rssi-at-1m=-27dBm
   eddy-namespace="b2b98de4c81c47c2b14e" eddy-instance="750000000000"

 2 address-type=public address=DC:2C:6E:F6:54:7D name="DC:2C:6E:F6:54:7D" persist=no
   mtik-key="" rssi=-74
   last-data="0201060303AAFE1116AAFE20000B701549023532D802384F46"
   last-seen=2023-08-22 11:20:13 beacon-types=eddystone-tlm eddy-version=0
   eddy-battery-voltage=2.928V eddy-temperature=21.285C eddy-packet-count=37040856
   eddy-uptime=3724474.2s

 3 address-type=public address=DC:2C:6E:0F:C0:3E name="DC:2C:6E:0F:C0:3E" persist=no
   mtik-key="" rssi=-72 last-data="15FF4F0901000214FFFF0200FDFF4F1774E00F000064"
   last-seen=2023-08-22 11:20:06 beacon-types=mikrotik mtik-version=1
   mtik-encrypted=no mtik-acc-x=-0.003G mtik-acc-y=0.007G mtik-acc-z=-0.011G
   mtik-temperature=23.308C mtik-battery=100% mtik-uptime=1040500s mtik-flags=""

 4 address-type=public address=60:C0:BF:20:9A:50 name="60:C0:BF:20:9A:50" persist=no
   mtik-key="" rssi=-66
   last-data="0201041BFF4160C0BF209A50FFA4CA8906E48C0377DCFDD2DF7AF02FFC6AC5"
   last-seen=2023-08-22 11:20:11 beacon-types=""
```

You can filter the list, for example, based on the "address" of the device (knowing MAC-address of the tag):

```
/iot bluetooth peripheral-devices print detail where address="DC:2C:6E:0F:C0:3E"
 0 address-type=public address=DC:2C:6E:0F:C0:3E name="my_tag" persist=yes
   mtik-key="" rssi=-60 last-data="15FF4F090100669DFCFF0600FCFF6117F1E50F000064"
   last-seen=2023-08-22 11:43:31 beacon-types=mikrotik mtik-version=1
   mtik-encrypted=no mtik-acc-x=-0.015G mtik-acc-y=0.023G mtik-acc-z=-0.015G
   mtik-temperature=23.378C mtik-battery=100% mtik-uptime=1041905s mtik-flags=""

/iot bluetooth peripheral-devices print value-list where address="DC:2C:6E:0F:C0:3E"
      address-type: public
           address: DC:2C:6E:0F:C0:3E
              name: my_tag
           persist: yes
          mtik-key:
              rssi: -71
         last-data: 15FF4F0901002AC60400000004004F17D4E90F000064
         last-seen: 2023-08-22 12:00:06
      beacon-types: mikrotik
      mtik-version: 1
    mtik-encrypted: no
        mtik-acc-x: 0.015G
        mtik-acc-y: 0G
        mtik-acc-z: 0.015G
  mtik-temperature: 23.308C
      mtik-battery: 100%
       mtik-uptime: 1042900s
        mtik-flags:
```

Or, for example, filter the list based on the beacon type:

```
/iot bluetooth peripheral-devices print detail where beacon-types=mikrotik
 0 address-type=public address=DC:2C:6E:0F:C0:3E name="my_tag" persist=yes
   mtik-key="" rssi=-69 last-data="15FF4F0901000747020002000100611778E60F000064"
   last-seen=2023-08-22 11:45:46 beacon-types=mikrotik mtik-version=1
   mtik-encrypted=no mtik-acc-x=0.007G mtik-acc-y=0.007G mtik-acc-z=0.003G
   mtik-temperature=23.378C mtik-battery=100% mtik-uptime=1042040s mtik-flags=""

 7 address-type=public address=2C:C8:1B:4B:BB:0A name="2C:C8:1B:4B:BB:0A" persist=no
   mtik-key="" rssi=-44 last-data="15FF4F090100770900000FCFFFDFFD519BF9EFF00005B"
   last-seen=2023-08-22 11:45:53 beacon-types=mikrotik mtik-version=1
   mtik-encrypted=no mtik-acc-x=0G mtik-acc-y=-0.015G mtik-acc-z=-0.011G
   mtik-temperature=25.832C mtik-battery=91% mtik-uptime=16752319s mtik-flags=""
```

Additionally, you have the option to set "`persist=yes`", which will make sure that the device/tag stays on the list forever (because devices that stop broadcasting payloads will be timed-out after one minute and removed from the list until new payloads start appearing in the air):

```
/iot bluetooth peripheral-devices set persist=yes address="DC:2C:6E:0F:C0:3E"
```

You can also set a name for the device, so you can easier find it on the list, with the command:

```
/iot bluetooth peripheral-devices set name="my_tag" address="DC:2C:6E:0F:C0:3E"
```

Decode-ad

In this menu, you can decode static MikroTik, Eddystone TLM, Eddystone UID and iBeacon payloads.

To decode a payload, just input it into the "data" field:

```
/iot bluetooth decode-ad data=0201060303AAFE1116AAFE20000B6E158402353AF20238576B
            type: eddystone-tlm
         version: 0
  battery-voltage: 2.926V
      temperature: 21.515C
     packet-count: 37042930
           uptime: 3724682.7s

/iot bluetooth decode-ad data=15FF4F090100032E0100FFFF00004F17C1E80F000064
         type: mikrotik
      version: 1
    encrypted: no
        acc-x: 0.003G
        acc-y: -0.003G
        acc-z: 0G
  temperature: 23.308C
       uptime: 1042625s
        flags:
      battery: 100%
```

Bluetooth tag-tracking using MQTT and ThingsBoard

Introduction

Bluetooth interface implementation in RouterOS allows the device to capture Bluetooth advertising packets that are broadcasted over 37, 38, and 39 advertising channels. More information can be found in the guide here.

Bluetooth tags like, for example, TG-BT5-IN and TG-BT5-OUT, do exactly that. They broadcast advertising payloads over the mentioned channels. To understand what kind of information is stored in the payload, make sure to check the link. The tags can be configured (using the MikroTik Beacon Manager app) to broadcast the payloads automatically, with an interval or/and when a movement, tilt, or free-fall trigger is detected. In simple terms, the tag will "tell" (broadcast to) all the surrounding Bluetooth scanners (like the KNOT) information about itself periodically.

When the payload is broadcasted by the tag, and the tag is within the KNOT's Bluetooth operating range, the KNOT will capture and display the payload under its "scanner" Bluetooth interface section. It would look like this:

```
/iot bluetooth scanners advertisements print
Columns: DEVICE, PDU-TYPE, TIME, ADDRESS-TYPE, ADDRESS, RSSI, LENGTH, DATA
#  DEVICE  PDU-TYPE        TIME                  ADDRESS-TYPE  ADDRESS            RSSI    LENGTH
DATA
0  bt1     adv-noconn-ind  mar/07/2023 12:11:57  public        DC:2C:6E:0F:C0:3D  -51dBm      22
15ff4f09010079100000ffff0000cf188a6b2b000064
1  bt1     adv-noconn-ind  mar/07/2023 12:11:58  public        2C:C8:1B:4B:BB:0A  -49dBm      22
15ff4f090100168dfeffffffeffa51ae1362200005e
```

The example above shows us, that the KNOT sees two Bluetooth tags within its operating range with MAC addresses "DC:2C:6E:0F:C0:3D" and "2C:C8:1B:4B:BB:0A", their respective payloads ("DATA" field) and the signal strength ("RSSI" field).

When testing locally to see how many payloads can the KNOT handle, we've achieved the following results → with 300 tags (in factory settings), scattered around the KNOT and using a Bluetooth filter "keep-newest" (which overwrites previously received payloads for each unique MAC address with the newest one, so that the Bluetooth list would display 1 payload per 1 unique tag's MAC address at all times), all 300 MAC addresses appeared in the KNOT's range after 30-40 seconds. This is where you need to keep in mind that all 300 tags broadcasting over the same channels at the same time will cause interference (delay in the reception). When we "cleared" the Bluetooth payload list, each second the list got 20 new entries and after about ~15 seconds, the list had 250-290 payloads. Then after ~15 seconds more, the list displayed all 300 unique tag payloads. **The actual number of tags your KNOT is able to handle is heavily dependent on the environment, so it is better to test it on sight.**

With the help of RouterOS scripting and scheduling, we can make the KNOT automatically-periodically scan the payload list and, in case, a specific payload or a specific tag's MAC address is found on the list, we can make the KNOT structure an MQTT message (out of the printed information shown in the example above) and send it to the configured server via MQTT, e-mail or HTTP post. Script examples will be shown later on in the guide.

As the title suggests, the goal is to implement a **Bluetooth tag-tracking solution** and the idea is quite simple. **When you have 2 KNOTs** (KNOT-A and KNOT-B), running the same script on a scheduler, **and the tag moves between their Bluetooth operating ranges**, the data on **the server will indicate whether** it was **KNOT-A or KNOT-B** that **have sent the** tag's **payload**. That will help you figure out the proximity of the tag. Whether the tag is broadcasting payloads in the KNOT-A zone, or in the KNOT-B zone.

You will need a server where the data is going to be stored and visualized. In this guide, we will showcase a server called ThingsBoard and how to communicate with it using the MQTT protocol.

ThingsBoard has a cloud solution and different local installation options (on different OS). Since we've added a container feature, it became possible to also run the platform within RouterOS. In order to do that, you will need a RouterOS device that has at least 2 GB RAM to spare or 1 GB RAM to spare with minimal load, has the option to increase storage (for example, an additional USB port), and is either ARM64 or AMD64 architecture. Consider using C HR machine, as it could be a good fit.

Setup requirements:

- a running ThingsBoard server;
- 2+ KNOTs with access to the server's network via ethernet, Wi-Fi, or cellular connection (the amount of the units required depends on the size of the area that needs to be covered);
- 1+ Bluetooth TG-BT5-IN and/or TG-BT5-OUT tags (depending on how many assets you need to track - 1 tag per asset).

Scenario explanation

Let's take a look at a basic example first. We have two KNOTs (KNOT-A and KNOT-B). We've tested Bluetooth ranges in our environment and could verify that both KNOTs are able to capture the tag at a 70-meter distance. If we install KNOT-A and KNOT-B 140 meters apart, their Bluetooth ranges will not overlap or overlap just slightly. When the tag moves into the KNOT-A range → the payload from the monitored tag will appear under the Bluetooth scanner list → the script will be initiated on a set schedule → an MQTT message with a report is going to be sent to the server → and, finally, the server will display that the tag is in the KNOT-A zone. When the tag moves into the KNOT-B range, the same happens, and the server will display that the tag is inside the KNOT-B zone.

The actual Bluetooth operating distance can vary from site to site because a lot of different factors can impact it, like 2.4 GHz interference or the surrounding materials used. For example, in line of sight, with no interference, the distance at which the KNOT is able to capture the tag's broadcasted payload, can be up to 180 meters (KNOT — ~180 meters — TG-BT5-OUT). But you also have to keep in mind that with further distance, more packets will be lost on the way. In the office environment, the range can go down to 30-100 meters.

Logically, if the Bluetooth operating ranges overlap and the tag is within the overlapped area (at the same time within KNOT-A and KNOT-B Bluetooth ranges), both KNOTs will send the data and the server is going to show that the tag is reported by both devices at the same time.

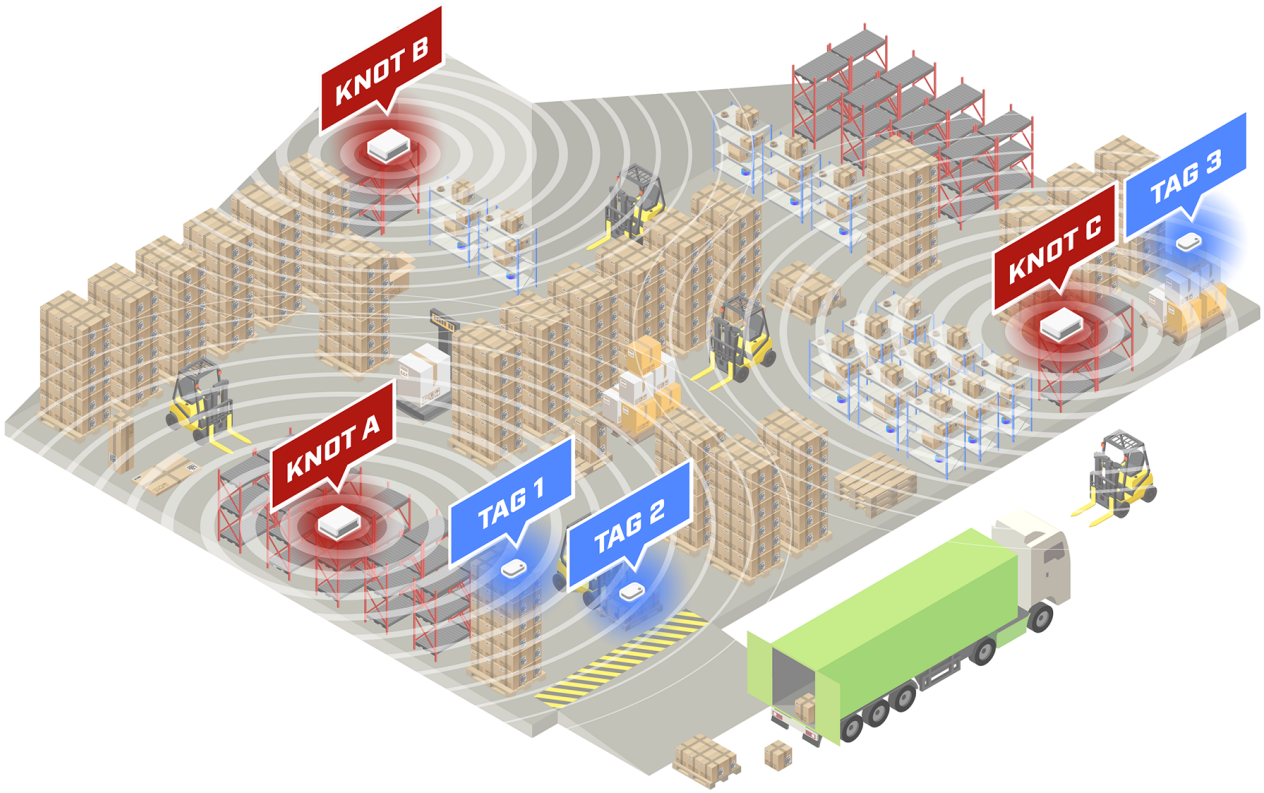
Surely, there will be zones where two or more KNOT's Bluetooth ranges overlap and you can use it to your advantage. Basically, you will have information that the tag, right now, is on the edges of the Bluetooth ranges in-between specific KNOT zones. In other words, when the asset moves into the overlapped area, you will have information on the server that the asset is somewhere between the two KNOT operating ranges, which is useful information to have **as it gives more precision**.

Additionally, **the tag's output power can be reduced using Tx power parameter**. Meaning, that even if the tag's payloads are broadcasted way too far and they are caught by other KNOTs that are not supposed to see those payloads (at longer distances) → you can reduce the output power of the tag, which will reduce the distance at which the KNOT is able to capture it. That way, you can "tweak" the "range" of reception and also avoid "interfering" with the signal in other zones.

The scripts we've prepared also allows you to set up a filter (that will be shown later on), which will make the KNOT ignore the payloads that the scanner captures unless the signal strength (RSSI) is stronger than the specified value. In the Bluetooth scanner example print shown above under the "Introduction " section, we can see that the KNOT sees one of the tags with an **RSSI** signal strength of **-51 dBm** (tag with a MAC address "**DC:2C:6E:0F:C0:3D**") and the other one with an **RSSI** signal strength of **-49 dBm** (tag with a MAC address "**2C:C8:1B:4B:BB:0A**"). So, **if we** apply a filter to the script to **ignore** all payloads that are received with signal strength (**RSSI**) **weaker than -50 dBm**, our **KNOT would report that only tag "2C:C8:1B:4B:BB:0A" is within the Bluetooth range**, because its RSSI is -49 dBm, and the second tag (with RSSI -51 dBm) will be ignored. What this means, essentially, it is a second way of "tweaking" the "range" of reception. The actual signal strength will vary between different locations (as mentioned before, because of interference and surrounding materials), so it needs to be tested on sight.

Example #1

One of the use cases is shown in the topology below:



The scale of objects and Bluetooth operating ranges are just shown as an example, to help visually understand and imagine the topology!

We have a warehouse area and **we have 3 assets** (pallets) that we are interested in tracking. **We** also **have 3 zones**: zone **A**, where newly arrived pallets are stored; zone **B**, where our assets are relocated to be inspected; and zone **C**, where assets are moved after inspection. To achieve Bluetooth asset-tracking, just install x1 KNOT per zone and x1 tag per asset.

If TAG 1 and TAG 2 (pallets) arrive in the "arrival" zone A, KNOT A will report to the server that both assets are within its Bluetooth range. If TAG 3 gets moved to zone C, the server will indicate it is within the KNOT C range. If TAG 1 and TAG 2 move toward the B zone, and stay on the edges between A and B zones, the server will show that it is in the overlapped area (at the same time within KNOT-A and KNOT-B ranges). If the tags move to the middle of the warehouse, the server will indicate that they are in all 3 zones at once, in the overlapping area.

Configuration

In this example, we will showcase a basic topology, when two KNOTs are used and we just want to know whether the tag is located in one part of the building or the other one (whether it is inside zone A or zone B).

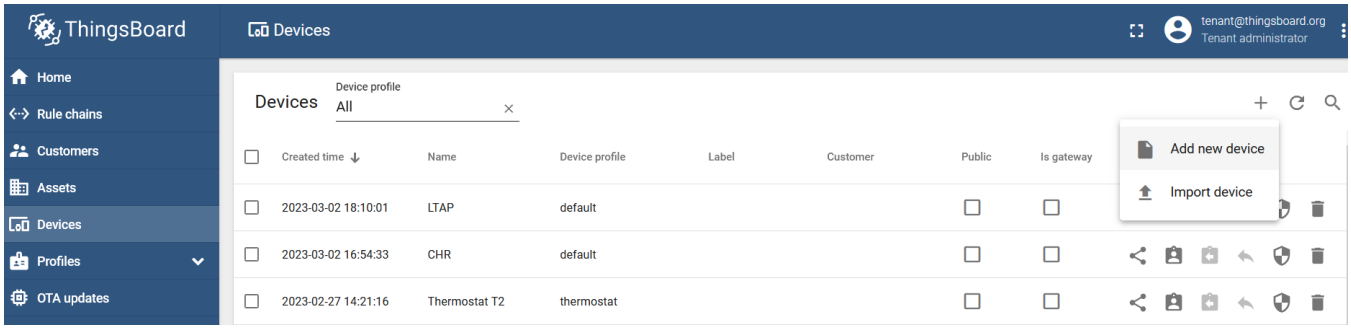
ThingsBoard preparation

Check the guide over here to understand how the ThingsBoard and the RouterOS can be set up to utilize MQTT communication.

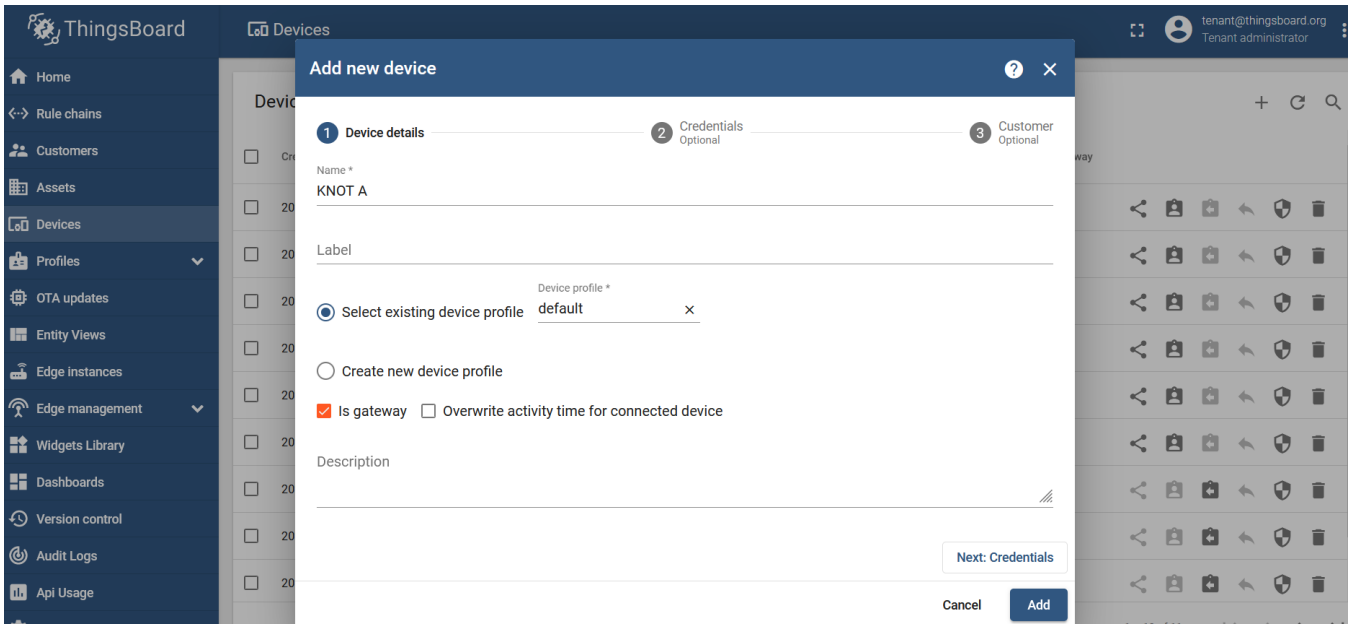
> This example will showcase access-token scenario for simplicity reasons, but you can use other available options as well. For the production environment, it is suggested to use SSL-MQTT, as non-SSL-MQTT can be easily packet captured and inspected.
>
> To understand how to implement SSL-MQTT communication on the instance that is run with the container. Check the guide linked here (Enablin g HTTPS and SSL MQTT section).

Create 2 KNOTs under the ThingsBoard GUI and make them "gateways".

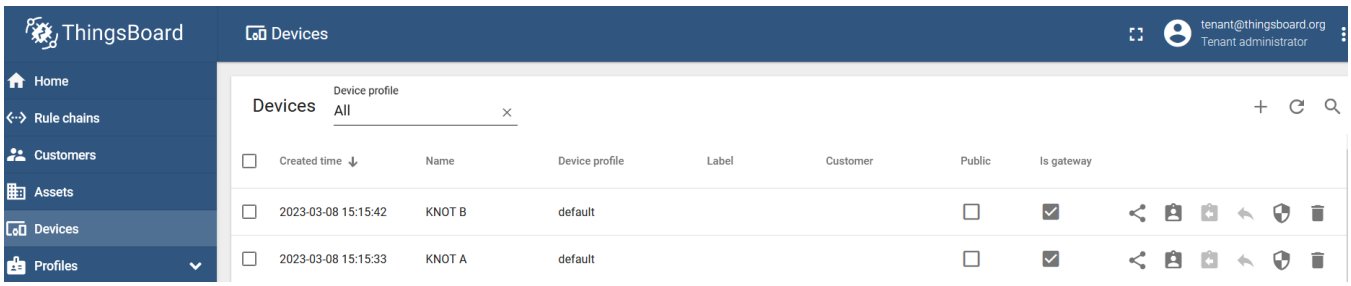
Go to the "Devices" section, click on "+" button and "Add new device":



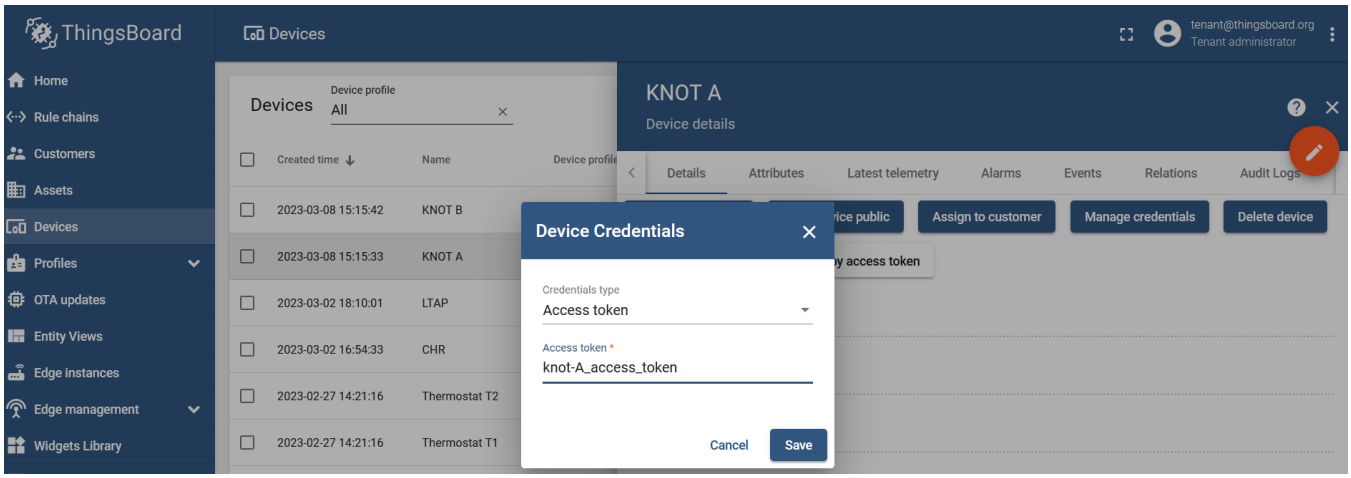
Name the device and checkbox the "Is gateway" option:



Do that for each KNOT that you have:



Set up a unique access token (unique credentials) for each KNOT under the device you've just created, under the "Manage credentials" tab:



RouterOS configuration

Preparation

Before we proceed, we need to confirm that our Bluetooth tag actually appears in the KNOT's Bluetooth range and that the KNOT detects them. To do that, you can issue the command "`/iot bluetooth scanners advertisements print`":

```
/iot bluetooth scanners advertisements print
 # DEVICE     PDU-TYPE        TIME                ADDRESS-TYPE ADDRESS                  RSSI     LENGTH
DATA
 0 bt1        adv-noconn-ind  mar/08/2023 12:35:15 public       2C:C8:1B:4B:BB:0A       -50dBm         22
15ff4f090100b0110100ffff00000019d68d2300005d
 1 bt1        adv-noconn-ind  mar/08/2023 12:35:16 public       DC:2C:6E:0F:C0:3D       -39dBm         22
15ff4f0901008f3cfcfffbfffaff301783c22c000064
 2 bt1        adv-noconn-ind  mar/08/2023 12:35:35 public       2C:C8:1B:4B:BB:0A       -50dBm         22
15ff4f09010084d500000400ffff0319ea8d2300005d
 3 bt1        adv-noconn-ind  mar/08/2023 12:35:45 public       2C:C8:1B:4B:BB:0A       -50dBm         22
15ff4f090100e607faffffff03000319f48d2300005d
```

Or you can check it using Webfig or Winbox under the IoT>Bluetooth>Advertising reports tab.

The list can be chaotic. Random payloads can appear on the list as the scanner captures everything around it. To help reduce the list, you can filter it using the tag's MAC address "`/iot bluetooth scanners advertisements print where address=DC:2C:6E:0F:C0:3D`":

```
/iot bluetooth scanners advertisements print where address=DC:2C:6E:0F:C0:3D
 # DEVICE     PDU-TYPE        TIME                ADDRESS-TYPE ADDRESS                  RSSI     LENGTH
DATA
 0 bt1        adv-noconn-ind  mar/08/2023 12:41:06 public       DC:2C:6E:0F:C0:3D       -49dBm         22
15ff4f0901005ab20100fdfffdff4017e1c32c000064
 1 bt1        adv-noconn-ind  mar/08/2023 12:41:26 public       DC:2C:6E:0F:C0:3D       -40dBm         22
15ff4f0901003497040000000fcff4017f5c32c000064
 2 bt1        adv-noconn-ind  mar/08/2023 12:41:36 public       DC:2C:6E:0F:C0:3D       -49dBm         22
15ff4f09010073fb0000000000003017ffc32c000064
 3 bt1        adv-noconn-ind  mar/08/2023 12:41:46 public       DC:2C:6E:0F:C0:3D       -43dBm         22
15ff4f090100b88cfffffffffffff401709c42c000064
```

To figure out how to decypher the payload, please check the guide here.

MQTT broker

On each KNOT setup MQTT broker.

For KNOT A:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=1883 username=knot-A_access_token
```

Where:

- `name` is the name that you wish to give to the broker and this name will be used later in the script;
- `address` is the IP address of the broker/ThingsBoard server;
- `port` is the TCP port that the broker is listening for → for non-SSL it is typically TCP 1883;
- `username` is dictated by the MQTT broker and, in our case, it is an "access token" that was generated in the ThingsBoard management portal.

For KNOT B → Do the same step. Just change `username` to the respective access token that was generated for the KNOT B device (gateway).

Script

Import the script shown below to each KNOT. To do that, just copy the below shown "code" and paste it into a new terminal window and press "ENTER":

```
/system script add dont-require-permissions=no name=tracking owner=admin policy=\
    ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="# Requ\
    ired packages: iot\r\
    \n\r\
    \n################################ Configuration ##############################\
    ##\r\
    \n# Name of an existing MQTT broker that should be used for publishing\r\
    \n:local broker \"tb\"\r\
    \n\r\
    \n# MQTT topic where the message should be published\r\
    \n:local topic \"v1/gateway/telemetry\"\r\
    \n\r\
    \n# POSIX regex for filtering advertisement Bluetooth addresses. E.g. \"^BC:33:\
    AC\"\r\
    \n# would only include addresses which start with those 3 octets.\r\
    \n# To disable this filter, set it to \"\"\r\
    \n:local addressRegex \"\"\r\
    \n\r\
    \n# POSIX regex for filtering Bluetooth advertisements based on their data. Sam\
    e\r\
    \n# usage as with 'addressRegex'.\r\
    \n:local advertisingDataRegex \"\"\r\
    \n\r\
    \n# Signal strength filter. E.g. -40 would only include Bluetooth advertisement\
    s\r\
    \n# whose signal strength is stronger than -40dBm.\r\
    \n# To disable this filter, set it to \"\"\r\
    \n:local rssiThreshold \"\"\r\
    \n\r\
    \n#Name the KNOT. Identity of the unit that will be senting the message. This n\
    ame will be reported to the MQTT broker.\r\
    \n:local gwName \"KNOT_A\"\r\
    \n\r\
    \n################################## Bluetooth ################################\
    ##\r\
    \n:put (\"[*] Gathering Bluetooth info...\")\r\
    \n\r\
    \n:global makeRecord do={\r\
    \n    :local jsonStr \"{\\\"ts\\\":\$ts,\\\"values\\\":{\\\"reporter\\\":\\\"\$\
    gwName\\\",\\\"rssi\\\":\$rssi}}\"\r\
    \n    :return \$jsonStr\r\
    \n}   \r\
    \n\r\
    \n# array of record strings collected for each advertising MAC address\r\
    \n:global macRecords [:toarray \"\"]\r\
    \n\r\
    \n# process advertisements and update macRecords\r\
    \n:local advertisements [/iot bluetooth scanners advertisements print detail as\
    -value where \\\r\
    \naddress ~ \$addressRegex and \\\r\
    \ndata ~ \$advertisingDataRegex and \\\r\
    \nrssi > \$rssiThreshold]\r\
    \n\r\
    \n/iot/bluetooth/scanners/advertisements clear\r\
    \n\r\
    \n:foreach adv in=\$advertisements do={\r\
```

```
\n:local address (\$adv->\"address\")\r\
\n:local rssi (\$adv->\"rssi\")\r\
\n:local epoch (\$adv->\"epoch\")\r\
\n               \r\
\n:local recordStr [\$makeRecord ts=\$epoch gwName=\$gwName rssi=\$rssi]\r\
\n\r\
\n:if ([:len (\$macRecords->\$address)] > 0) do={\r\
\n:local str (\$macRecords->\$address)\r\
\n:local newStr \"\$str,\$recordStr\"\r\
\n:set (\$macRecords->\$address) \$newStr} else={:set (\$macRecords->\$address)\
\_\$recordStr}}\r\
\n\r\
\n# TODO: add some logic to decide when we want to send data\r\
\n:local sendData true\r\
\n\r\
\n:if (\$sendData) do={\r\
\n:local jsonStr \"{\"\r\
\n\r\
\n:foreach addr,advRec in=\$macRecords do={\r\
\n:set jsonStr \"\$jsonStr\\\"\$addr\\\":[\$advRec],\"}\r\
\n\r\
\n:local payloadlength\r\
\n:set payloadlength [:len (\$jsonStr)]\r\
\n:local remcom\r\
\n:set remcom [:pick \$jsonStr 0 (\$payloadlength-1)]\r\
\n:set jsonStr \"\$remcom}\"\r\
\n:local message\r\
\n:set message \"\$jsonStr\"\r\
\n:log info \"\$message\";\r\
\n:put (\"[*] Message structured: \$message\")\r\
\n:put (\"[*] Total message size: \$[:len \$message] bytes\")\r\
\n:put (\"[*] Sending message to MQTT broker...\")\r\
\n/iot mqtt publish broker=\"\$broker\" topic=\"\$topic\" message=\$message}"
```

The script should appear under the "System>Scripts>Script List" tab with the name "tracking" or with the help of the command "`/system script print`".

There are certain script lines that you need to pay attention to.

Broker name configuration line, where you need to input the MQTT broker name that you have set:

> *:local broker "tb"*

You need to input a correct topic, used by the MQTT broker. Check ThingsBoard documentation for more details and, by default, the topic should be:

> *:local topic "v1/gateway/telemetry"*

MAC address filtering option inside the script itself. You can type in all 6 octets of the MAC address (to apply the filter to 1 specific tag), or you can filter the list using a few/couple of octets, like "^BC:33:AC" (to apply the filter, so that only MAC addresses that begin with "BC:33:AC:..." would be processed):

> *:local addressRegex "DC:2C:6E:0F:C0:3D"*

Payload content/data line. Allows you to filter the list, per specific payload content, like "manufacturer data". For example, "15ff4f09" would discard all payloads that are not MikroTik format payloads:

> *:local advertisingDataRegex "15ff4f09"*

RSSI signal strength filtering option. This filtering option was mentioned in the "Scenario explanation" section. This filter allows you to ignore any payload that has RSSI weaker than the configured value. For example. "-40" would only include Bluetooth advertisements that have signal strength stronger than -40dBm:

> *:local rssiThreshold "-40"*

KNOT identifier line. You will need to change it for each unique KNOT. For example, name your first KNOT → KNOT_A and your second KNOT → KNOT_B:

> *:local gwName "KNOT_A"*

The rest of the script does not need to be changed/altered

How does the script work when you run it? The script "inspects" the "Advertising reports" tab (payload list tab) using the filters applied and structures a JSON message. An example of the JSON message would be:

```
{
  "2C:C8:1B:4B:BB:0A": [
    {
      "ts": 1678967250600,
      "values": {
        "reporter": "KNOT_A",
        "rssi": -47
      }
    }
  ],
  "DC:2C:6E:0F:C0:3D": [
    {
      "ts": 1678967247850,
      "values": {
        "reporter": "KNOT_A",
        "rssi": -59
      }
    },
    {
      "ts": 1678967257849,
      "values": {
        "reporter": "KNOT_A",
        "rssi": -67
      }
    }
  ]
}
```

The data is structured per the ThingsBoard guide, where "`2C:C8:1B:4B:BB:0A`" and "`DC:2C:6E:0F:C0:3D`" are MAC addresses of tags that appeared in the KNOT's range, "`ts`" is unix timestamp of each payload broadcasted by the tag in milliseconds, "`reporter`" indicates which specific KNOT has sent the message and "`rssi`" is the signal strength of each payload broadcasted by the tag in dBm.

After the payload list is "searched" and the JSON message is structured, the Bluetooth interface payload list gets "cleaned" (or "flashed"), and the previously structured JSON message is sent to the ThingsBoard MQTT broker.

To run the script, use the command:

```
/system script run tracking
```

Scheduler

Apply a scheduler to the script, so that RouterOS periodically initiates the script by itself:

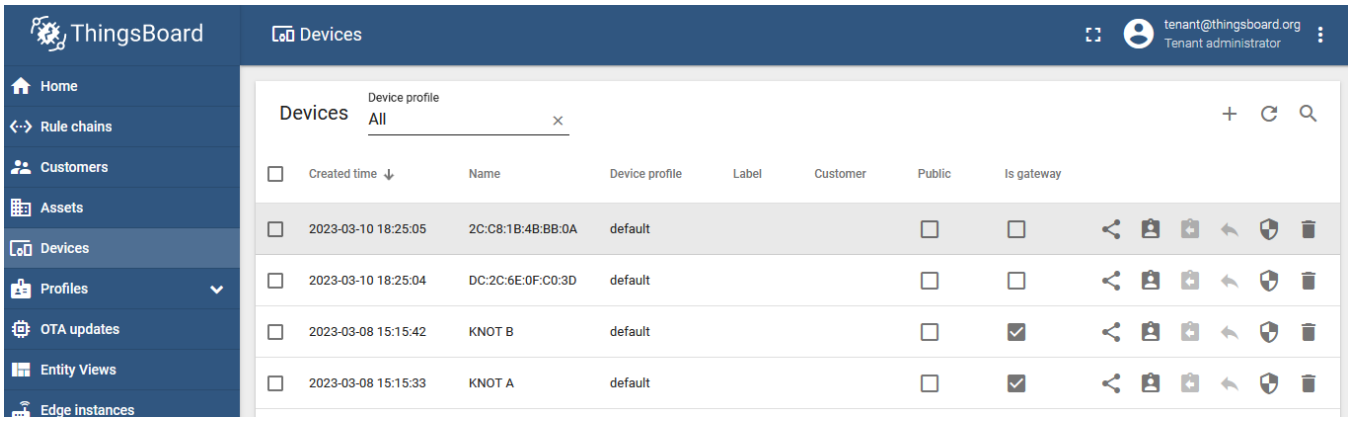
```
/system/scheduler/add name=bluetoothscheduler interval=30s on-event="/system/script/run tracking"
```

You can set up shorter and longer intervals. If you want to send data more often, so that the data is "fresher" → set up shorter time intervals (10-15 seconds). If you want to send fewer messages, less often → you can set up longer time intervals (30min+).

The JSON message structured using the script has a "`ts`" value (timestamp) assigned for each payload received. Meaning that **when the script is run**, for example, **every minute**, 1 tag is used and **the tag broadcasts** 1 payload every 10 seconds (that is **6 payloads per minute**) → ThingsBoard data (GUI) will be updated every minute, and every minute, 6 new entries will appear (each entry will indicate that it was received 10 seconds after the previous one). And if you send the message every 15 minutes when using 1 tag that is broadcasting a payload every 10 seconds (that is 6*15=90 payloads per 15 minutes) → ThingsBoard data (GUI) will be updated every 15 minutes but 90 entries will appear.

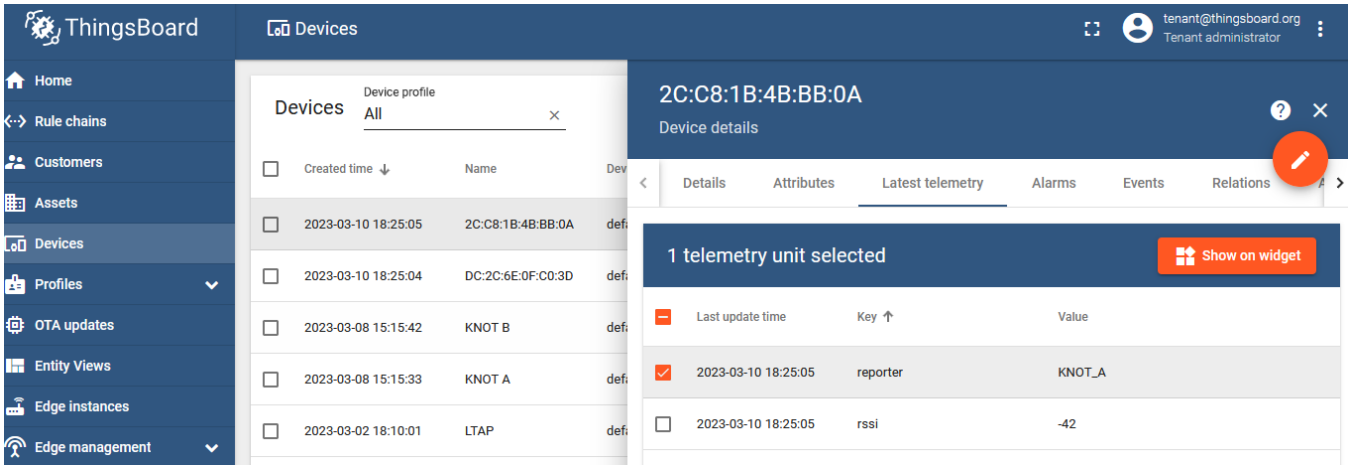
ThingsBoard data visualization and result verification

After you run the script with `/system script run tracking` or via a scheduler and refresh the GUI portal → all MAC addresses (tags) that are found in the JSON message, will be made into new devices under the ThingsBoard GUI:

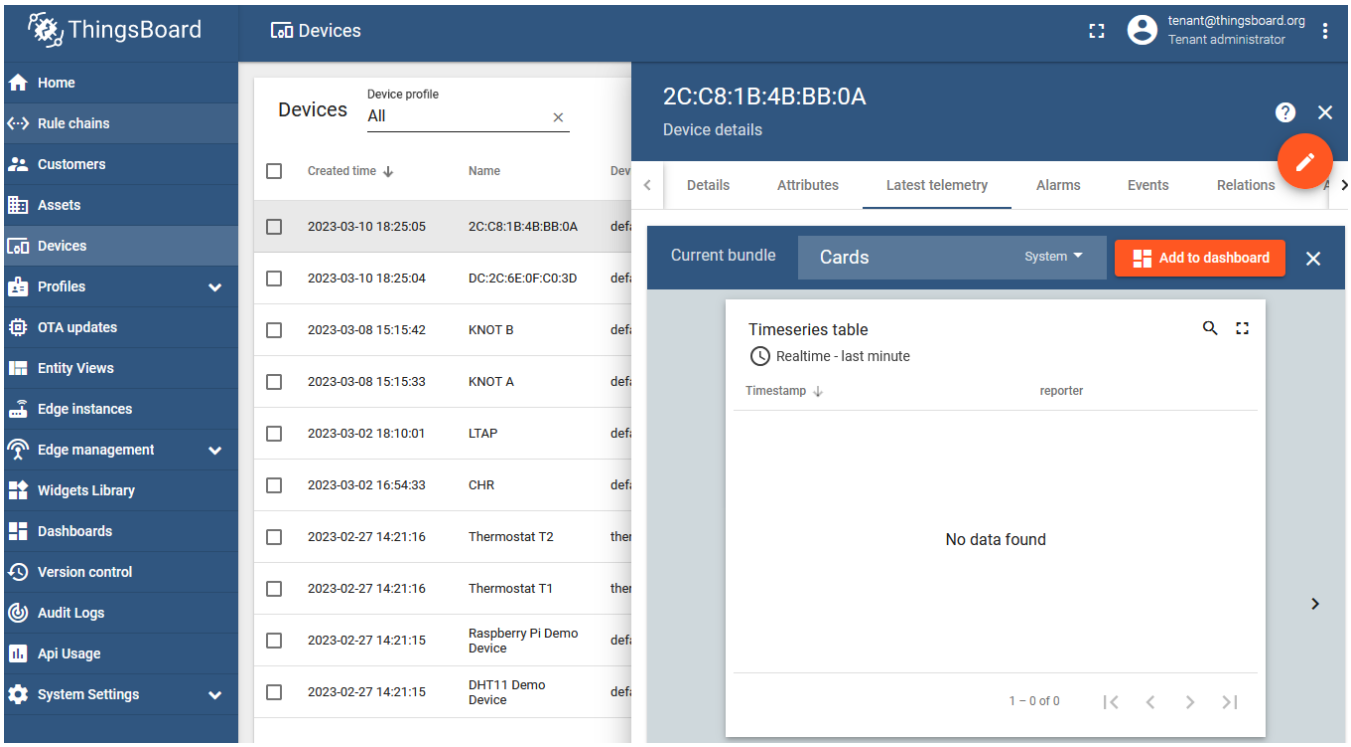


To help you visualize the data, you can use the built-in widgets or create your own one.

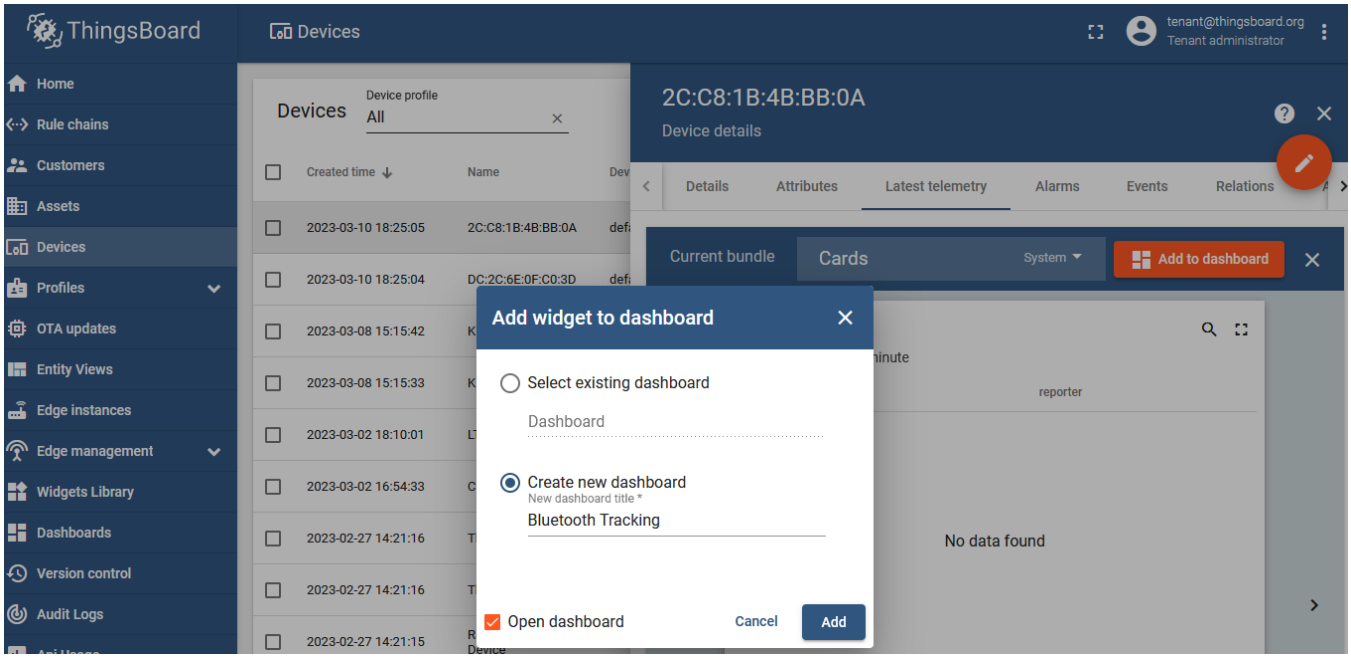
Select the tag's MAC address from the list of devices, go to the "Latest telemetry" section, checkbox the "reporter" parameter, and click on the "Show on widget" button:



Select a widget that you wish to use, for example under the "Cards" bundle, "Timeseries table" and click on "Add to dashboard":

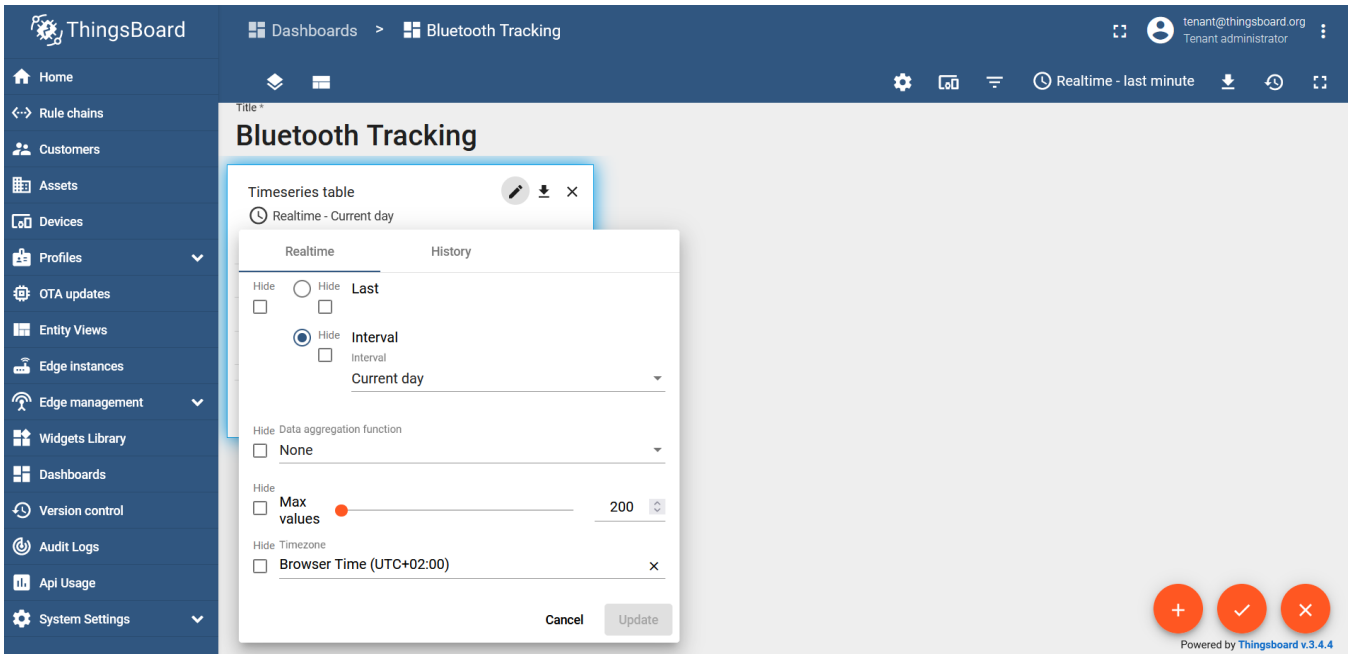


Create a new dashboard and name it, however, you like. Click on "Add":

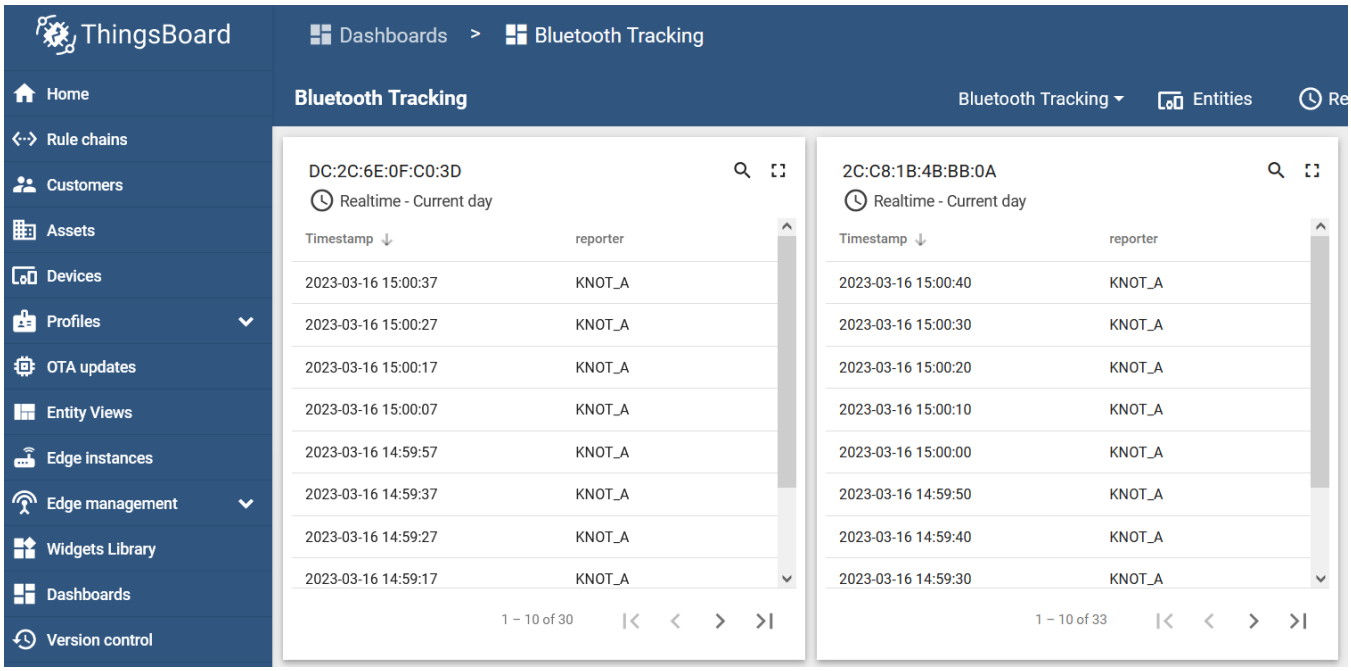


Do the same steps for your other tags that appeared under the "Devices" tab. Create a new widget for each unique tag under the same dashboard.

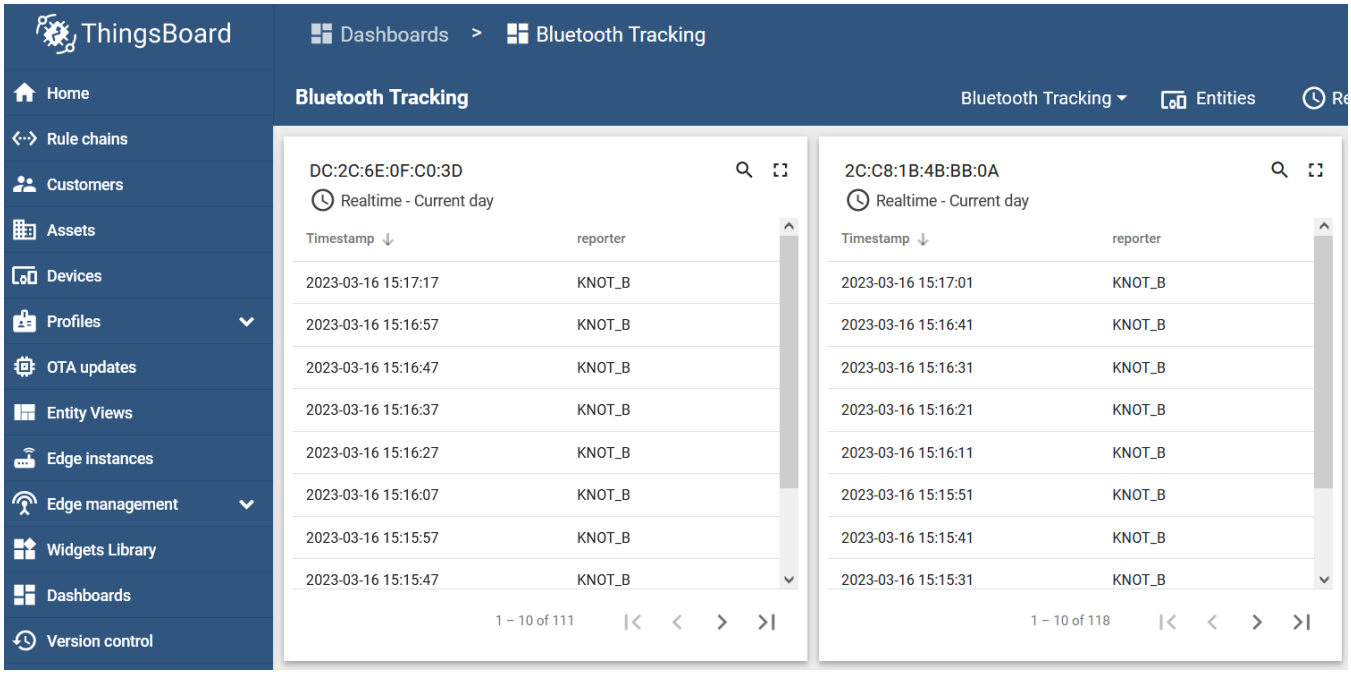
Change the widget's "Timewindow" from "Realtime-last minute" (which is used by default) to "Realtime-current day":



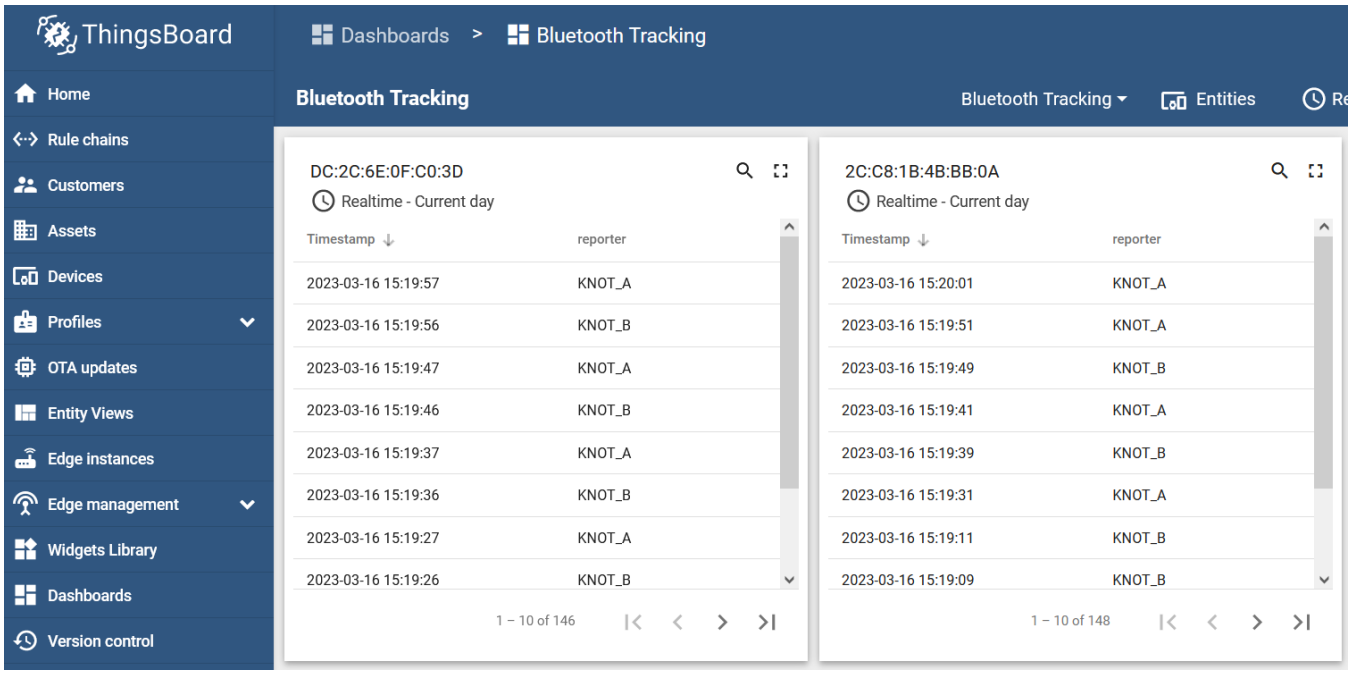
As a result, if both tags are inside the **KNOT A range**, the dashboard would show:



If they move to the **KNOT B range**, it would show:

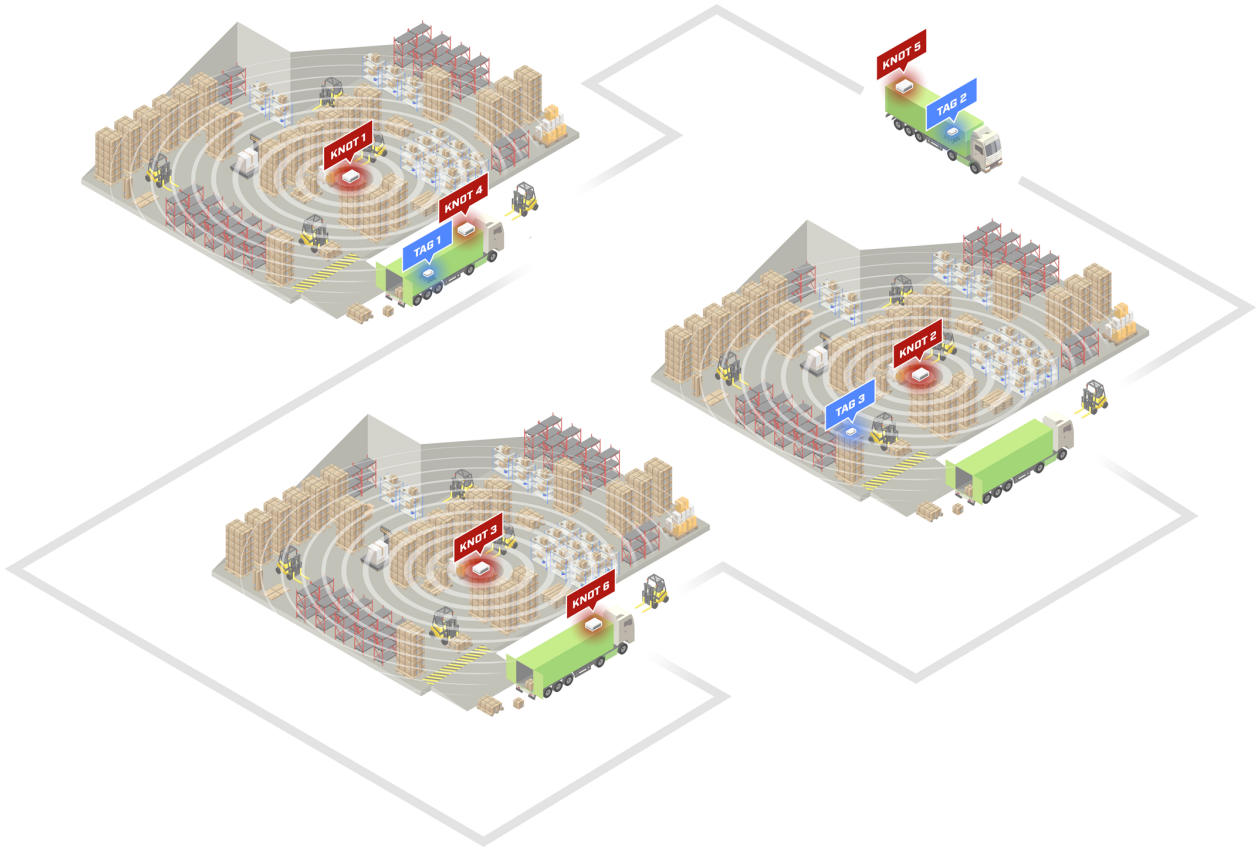


If the tags move to the **overlapped area**, inside both ranges, both reporters (KNOT_A and KNOT_B) should show up within a few seconds after each other, depending on the interval used in the scheduler:



Example #2

In the second example, we will showcase another topology:



We have a few warehouses, a few company delivery vehicles, and 3 assets that we are interested in tracking. Our assets are pallets that hold cargo and our goals are to know:

- in which specific warehouse (equipped with the KNOT) the asset (equipped with the tag) is currently in and how much time it spent inside the specific warehouse;
- whether the asset (equipped with the tag) is on the road, traveling between warehouses, and how much time it spent inside the vehicle (equipped with the KNOT);
- **(optionally) if TG-BT5-OUT tags are used**, what was the temperature during all this time? You can also/instead monitor other parameters that you can get out of the advertised payload, like for example acceleration;
- **(optionally)** find out the KNOT's GPS location.

To achieve Bluetooth asset-tracking, just install x1 KNOT per warehouse, x1 KNOT per vehicle, and x1 tag per asset.

We can see that TAG 1 is inside the vehicle, and this vehicle just parked near the warehouse. Both KNOT 1 and KNOT 4 will report to the server that the asset is inside their ranges. This will tell you that the asset is parked but not being transported yet.

We can see that TAG 2 is traveling between the warehouses and is only inside the KNOT 5 Bluetooth range. In this case, KNOT 5 will be the only reporter and the result that is displayed on the server would mean that the asset is getting transported.

We can see that TAG 3 is inside the warehouse. The server will indicate just that.

The data on the server will show the timestamps of each report sent by the KNOT, which will tell you for how long did the asset stay inside the specific device's Bluetooth range.

Configuration

In this example, we will showcase a basic topology with 2 warehouses, 1 vehicle/truck traveling between them, and 1 asset/pallet/tag.

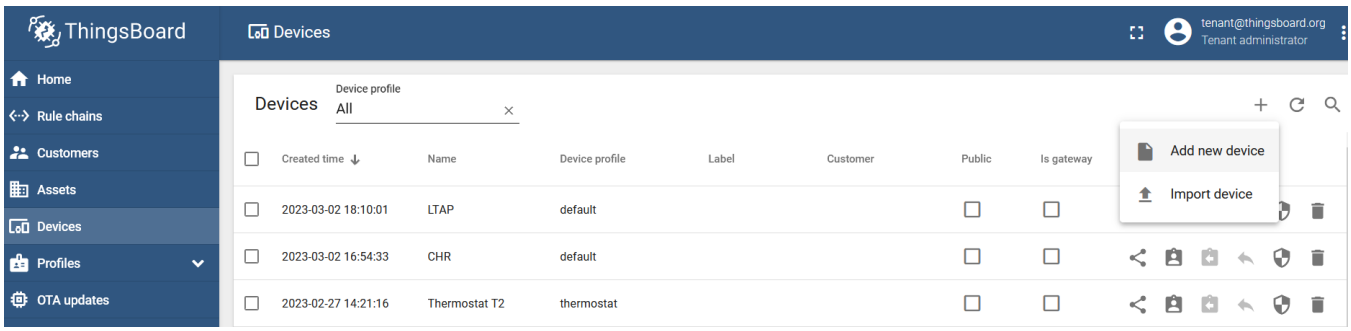
ThingsBoard preparation

Check the guide over here to understand how the ThingsBoard and the RouterOS can be set up to utilize MQTT communication.

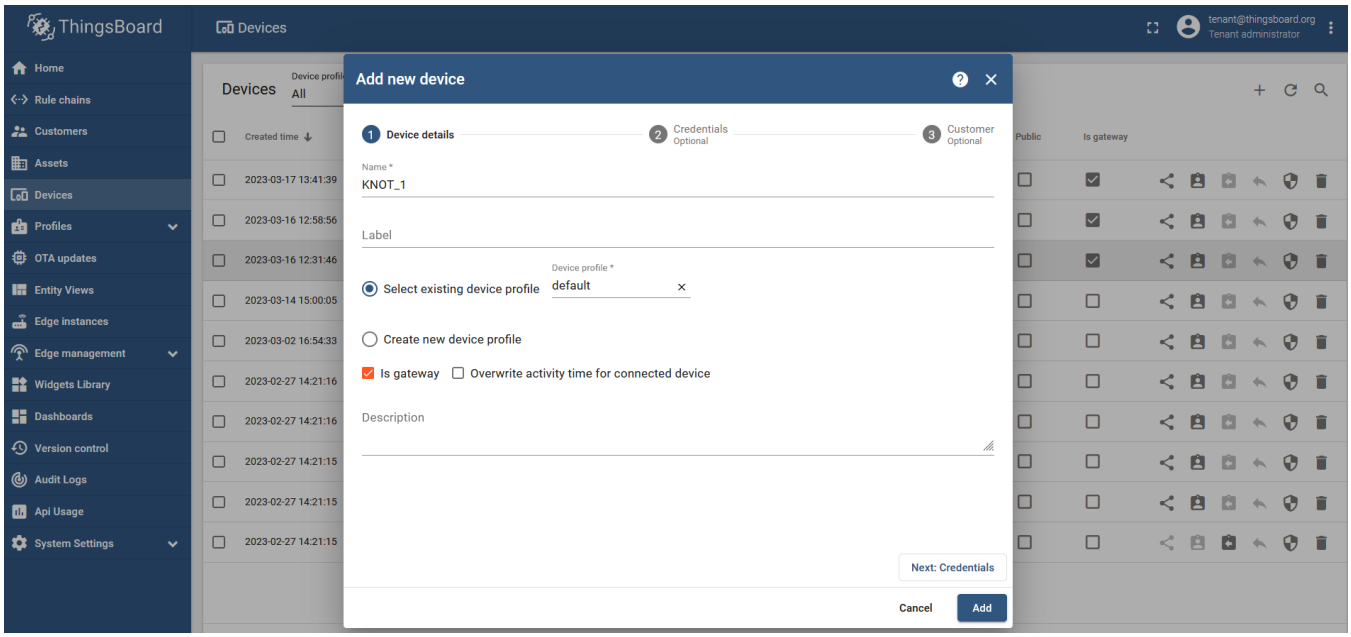
> This example will showcase access-token scenario for simplicity reasons, but you can use other available options as well. For the production environment, it is suggested to use SSL-MQTT, as non-SSL-MQTT can be easily packet captured and inspected.
>
> To understand how to implement SSL-MQTT communication on the instance that is run with the container. Check the guide linked here (Enabling HTTPS and SSL MQTT section).

Create 3 KNOTs under the ThingsBoard GUI and make them "gateways".

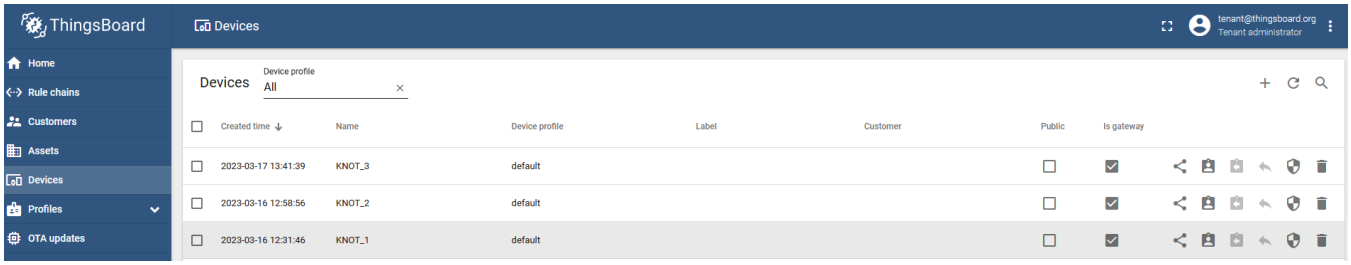
Go to the "Devices" section, click on "+" button and "Add new device":



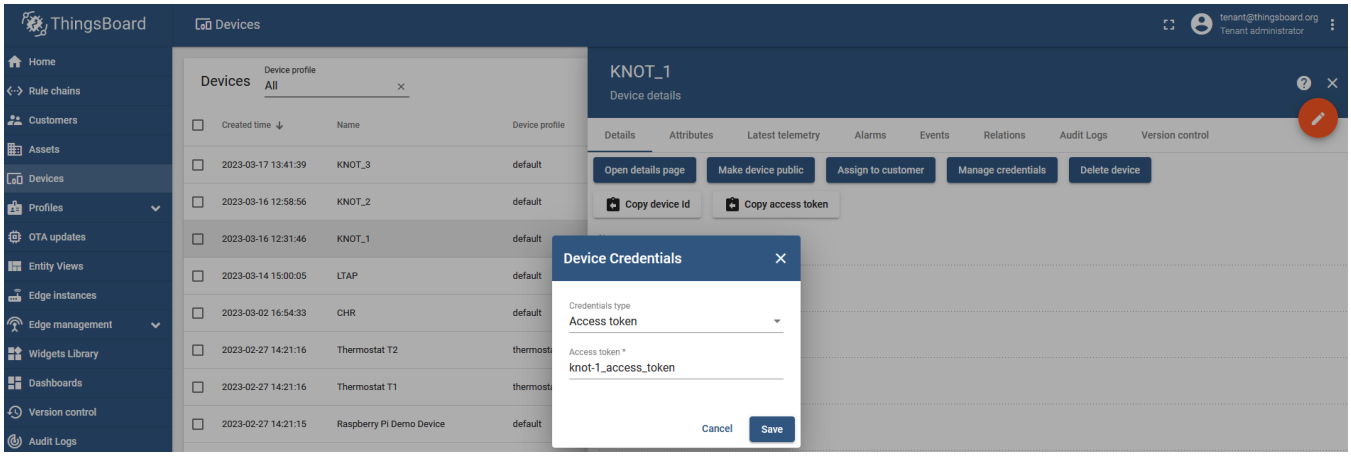
Name the device and checkbox the "Is gateway" option:



Do that for each KNOT that you have:



Set up a unique access token (unique credentials) for each KNOT under the device you've just created, under the "Manage credentials" tab:



RouterOS configuration

Preparation

Before we proceed, we need to confirm that our Bluetooth tag actually appears in the KNOT's Bluetooth range and that the KNOT detects them. To do that, you can issue the command "`/iot bluetooth scanners advertisements print`":

```
/iot bluetooth scanners advertisements print
 # DEVICE     PDU-TYPE        TIME                 ADDRESS-TYPE ADDRESS                   RSSI     LENGTH
DATA
 0 bt1        adv-noconn-ind  mar/08/2023 12:35:15 public       2C:C8:1B:4B:BB:0A        -50dBm         22
15ff4f090100b0110100ffff00000019d68d2300005d
 1 bt1        adv-noconn-ind  mar/08/2023 12:35:16 public       DC:2C:6E:0F:C0:3D        -39dBm         22
15ff4f0901008f3cfcfffbfffaff301783c22c000064
 2 bt1        adv-noconn-ind  mar/08/2023 12:35:35 public       2C:C8:1B:4B:BB:0A        -50dBm         22
15ff4f09010084d500000400ffff0319ea8d2300005d
 3 bt1        adv-noconn-ind  mar/08/2023 12:35:45 public       2C:C8:1B:4B:BB:0A        -50dBm         22
15ff4f090100e607faffffff03000319f48d2300005d
```

Or you can check it using Webfig or Winbox under the IoT>Bluetooth>Advertising reports tab.

The list can be chaotic. Random payloads can appear on the list as the scanner captures everything around it. To help reduce the list, you can filter it using the tag's MAC address "`/iot bluetooth scanners advertisements print where address=DC:2C:6E:0F:C0:3D`":

```
/iot bluetooth scanners advertisements print where address=DC:2C:6E:0F:C0:3D
 # DEVICE    PDU-TYPE        TIME                 ADDRESS-TYPE ADDRESS                   RSSI     LENGTH
DATA
 0 bt1       adv-noconn-ind  mar/08/2023 12:41:06 public       DC:2C:6E:0F:C0:3D        -49dBm         22
15ff4f0901005ab20100fdfffdff4017e1c32c000064
 1 bt1       adv-noconn-ind  mar/08/2023 12:41:26 public       DC:2C:6E:0F:C0:3D        -40dBm         22
15ff4f0901003497040000000fcff4017f5c32c000064
 2 bt1       adv-noconn-ind  mar/08/2023 12:41:36 public       DC:2C:6E:0F:C0:3D        -49dBm         22
15ff4f09010073fb0000000000003017ffc32c000064
 3 bt1       adv-noconn-ind  mar/08/2023 12:41:46 public       DC:2C:6E:0F:C0:3D        -43dBm         22
15ff4f090100b88cfffffffffffff401709c42c000064
```

To figure out how to decypher the payload, please check the guide here.

MQTT broker

On each KNOT setup MQTT broker.

For KNOT_1:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=1883 username=knot-1_access_token
```

Where:

- `name` is the name that you wish to give to the broker and this name will be used later in the script;
- `address` is the IP address of the broker/ThingsBoard server;
- `port` is the TCP port that the broker is listening for → for non-SSL it is typically TCP 1883;
- `username` is dictated by the MQTT broker and, in our case, it is an "access token" that was generated in the ThingsBoard management portal.

For KNOT_2 and KNOT_3 → Do the same steps. Just change `username` to the respective access token that was generated.

Script

Import the script shown below to each KNOT. To do that, just copy the below shown "code" and paste it into a new terminal window and press "ENTER":

```
/system script add dont-require-permissions=no name=tracking owner=admin policy=\
    ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="# Required package\
    s: iot\r\
    \n\r\
    \n################################# Configuration #################################\r\
    \n# Name of an existing MQTT broker that should be used for publishing\r\
    \n:local broker \"tb\"\r\
    \n\r\
    \n# MQTT topic where the message should be published\r\
    \n:local topic \"v1/gateway/telemetry\"\r\
    \n\r\
    \n# POSIX regex for filtering advertisement Bluetooth addresses. E.g. \"^BC:33:AC\"\r\
```

```
\n# would only include addresses which start with those 3 octets.\r\
\n# To disable this filter, set it to \"\"\r\
\n:local addressRegex \"\"\r\
\n\r\
\n# POSIX regex for filtering Bluetooth advertisements based on their data. Same\r\
\n# usage as with 'addressRegex'.\r\
\n:local advertisingDataRegex \"\"\r\
\n\r\
\n# Signal strength filter. E.g. -40 would only include Bluetooth advertisements\r\
\n# whose signal strength is stronger than -40dBm.\r\
\n# To disable this filter, set it to \"\"\r\
\n:local rssiThreshold \"\"\r\
\n\r\
\n#Name the KNOT. Identity of the unit that will be senting the message. This name will be \
reported to the MQTT broker.\r\
\n:local gwName \"1\"\r\
\n\r\
\n################################## Bluetooth ##################################\r\
\n:put (\"[*] Gathering Bluetooth info...\")\r\
\n\r\
\n:global makeRecord do={\r\
\n    :local jsonStr \"{\\\"ts\\\":\$ts,\\\"values\\\":{\\\"KNOT_\$gwName\\\":\\\"\$gwName\
\\\",\\\"rssi\\\":\$rssi}}\"\r\
\n    :return \$jsonStr\r\
\n}   \r\
\n\r\
\n# array of record strings collected for each advertising MAC address\r\
\n:global macRecords [:toarray \"\"]\r\
\n\r\
\n# process advertisements and update macRecords\r\
\n:local advertisements [/iot bluetooth scanners advertisements print detail as-value where\
\_\\\r\
\naddress ~ \$addressRegex and \\\r\
\ndata ~ \$advertisingDataRegex and \\\r\
\nrssi > \$rssiThreshold]\r\
\n\r\
\n/iot/bluetooth/scanners/advertisements clear\r\
\n\r\
\n:foreach adv in=\$advertisements do={\r\
\n:local address (\$adv->\"address\")\r\
\n:local rssi (\$adv->\"rssi\")\r\
\n:local epoch (\$adv->\"epoch\")\r\
\n                \r\
\n:local recordStr [\$makeRecord ts=\$epoch gwName=\$gwName rssi=\$rssi]\r\
\n\r\
\n:if ([:len (\$macRecords->\$address)] > 0) do={\r\
\n:local str (\$macRecords->\$address)\r\
\n:local newStr \"\$str,\$recordStr\"\r\
\n:set (\$macRecords->\$address) \$newStr} else={:set (\$macRecords->\$address) \$recordStr\
}}\r\
\n\r\
\n# TODO: add some logic to decide when we want to send data\r\
\n:local sendData true\r\
\n\r\
\n:if (\$sendData) do={\r\
\n:local jsonStr \"{\"\r\
\n\r\
\n:foreach addr,advRec in=\$macRecords do={\r\
\n:set jsonStr \"\$jsonStr\\\"\$addr\\\":[\$advRec],\"}\r\
\n\r\
\n:local payloadlength\r\
\n:set payloadlength [:len (\$jsonStr)]\r\
\n:local remcom\r\
\n:set remcom [:pick \$jsonStr 0 (\$payloadlength-1)]\r\
\n:set jsonStr \"\$remcom}\"\r\
\n:local message\r\
\n:set message \"\$jsonStr\"\r\
\n:log info \"\$message\";\r\
\n:put (\"[*] Message structured: \$message\")\r\
\n:put (\"[*] Total message size: \$[:len \$message] bytes\")\r\
\n:put (\"[*] Sending message to MQTT broker...\")\r\
```

```
\n/iot mqtt publish broker=\"\$broker\" topic=\"\$topic\" message=\$message}"
```

The script should appear under the "System>Scripts>Script List" tab with the name "tracking" or with the help of the command "`/system script print`".

There are certain script lines that you need to pay attention to.

Broker name configuration line, where you need to input the MQTT broker name that you have set:

> *:local broker "tb"*

You need to input a correct topic, used by the MQTT broker. Check ThingsBoard documentation for more details and, by default, the topic should be:

> *:local topic "v1/gateway/telemetry"*

MAC address filtering option inside the script itself. You can type in all 6 octets of the MAC address (to apply the filter to 1 specific tag), or you can filter the list using a few/couple of octets, like "^BC:33:AC" (to apply the filter, so that only MAC addresses that begin with "BC:33:AC:..." would be processed):

> *:local addressRegex "DC:2C:6E:0F:C0:3D"*

Payload content/data line. Allows you to filter the list, per specific payload content, like "manufacturer data". For example, "15ff4f09" would discard all payloads that are not MikroTik format payloads:

> *:local advertisingDataRegex "15ff4f09"*

RSSI signal strength filtering option. This filtering option was mentioned in the "Scenario explanation" section. This filter allows you to ignore any payload that has RSSI weaker than the configured value. For example. "-40" would only include Bluetooth advertisements that have signal strength stronger than -40dBm:

> *:local rssiThreshold "-40"*

KNOT identifier line. You will need to change it for each unique KNOT. For example, ID your first KNOT as "1", your second KNOT as "2" and your third KNOT as "3":

> *:local gwName "1"*

The rest of the script does not need to be changed/altered

How does the script work when you run it? The script "inspects" the "Advertising reports" tab (payload list tab) using the filters applied and structures a JSON message. An example of the JSON message would be:

```
{
  "2C:C8:1B:4B:BB:0A": [
    {
      "ts": 1680526087729,
      "values": {
        "KNOT_1": "1",
        "rssi": -47
      }
    }
  ],
  "DC:2C:6E:0F:C0:3D": [
    {
      "ts": 1680526065000,
      "values": {
        "KNOT_1": "1",
        "rssi": -49
      }
    },
    {
      "ts": 1680526075001,
      "values": {
        "KNOT_1": "1",
        "rssi": -50
      }
    }
  ]
}
```

The data is structured per the ThingsBoard guide, where "`2C:C8:1B:4B:BB:0A`" and "`DC:2C:6E:0F:C0:3D`" are MAC addresses of tags that appeared in the KNOT's range, "`ts`" is unix timestamp of each payload broadcasted by the tag in milliseconds, "`KNOT_X`" indicates which specific KNOT has sent the message and "`rssi`" is the signal strength of each payload broadcasted by the tag in dBm.

After the payload list is "searched" and the JSON message is structured, the Bluetooth interface payload list gets "cleaned" (or "flashed"), and the previously structured JSON message is sent to the ThingsBoard MQTT broker.

To run the script, use the command:

```
/system script run tracking
```

Script that includes temperature data (optional)

In case you wish to add temperature reports to the structured message, use the script below:

```
/system script add dont-require-permissions=no name=tracking+temp owner=admin policy=\
    ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="# Required package\
    s: iot\r\
    \n\r\
    \n################################ Configuration ################################\r\
    \n# Name of an existing MQTT broker that should be used for publishing\r\
    \n:local broker \"tb\"\r\
    \n\r\
    \n# MQTT topic where the message should be published\r\
    \n:local topic \"v1/gateway/telemetry\"\r\
    \n\r\
    \n# POSIX regex for filtering advertisement Bluetooth addresses. E.g. \"^BC:33:AC\"\r\
    \n# would only include addresses which start with those 3 octets.\r\
    \n# To disable this filter, set it to \"\"\r\
    \n:local addressRegex \"\"\r\
    \n\r\
    \n# POSIX regex for filtering Bluetooth advertisements based on their data. Same\r\
    \n# usage as with 'addressRegex'.\r\
    \n:local advertisingDataRegex \"\"\r\
    \n\r\
    \n# Signal strength filter. E.g. -40 would only include Bluetooth advertisements\r\
    \n# whose signal strength is stronger than -40dBm.\r\
    \n# To disable this filter, set it to \"\"\r\
    \n:local rssiThreshold \"\"\r\
    \n\r\
    \n#Name the KNOT. Identity of the unit that will be senting the message. This name will be \
    reported to the MQTT broker.\r\
    \n:local gwName \"1\"\r\
    \n\r\
```

```
\n################################## Bluetooth ##################################\r\
\n:global invertU16 do={\r\
\n    :local inverted 0\r\
\n    :for idx from=0 to=15 step=1 do={\r\
\n        :local mask (1 << \$idx)\r\
\n        :if (\$1 & \$mask = 0) do={\r\
\n            :set \$inverted (\$inverted | \$mask)\r\
\n        }\r\
\n    }\r\
\n    return \$inverted\r\
\n}\r\
\n\r\
\n:global le16ToHost do={\r\
\n    :local lsb [:pick \$1 0 2]\r\
\n    :local msb [:pick \$1 2 4]\r\
\n\r\
\n    :return [:tonum \"0x\$msb\$lsb\"]\r\
\n}\r\
\n:local from88 do={\r\
\n    :global invertU16\r\
\n    :global le16ToHost\r\
\n    :local num [\$le16ToHost \$1]\r\
\n\r\
\n    # Handle negative numbers\r\
\n    :if (\$num & 0x8000) do={\r\
\n        :set num (-1 * ([\$invertU16 \$num] + 1))\r\
\n    }\r\
\n\r\
\n    # Convert from 8.8. Scale by 1000 since floating point is not supported\r\
\n    :return ((\$num * 125) / 32)\r\
\n}\r\
\n:put (\"[*] Gathering Bluetooth info...\")\r\
\n\r\
\n:global makeRecord do={\r\
\n    :local jsonStr \"{\\\"ts\\\":\$ts,\\\"values\\\":{\\\"KNOT_\$gwName\\\":\\\"\$gwName\
\\\",\\\"temp\\\":\$temp}}\"\r\
\n    :return \$jsonStr\r\
\n}   \r\
\n\r\
\n# array of record strings collected for each advertising MAC address\r\
\n:global macRecords [:toarray \"\"]\r\
\n\r\
\n# process advertisements and update macRecords\r\
\n:local advertisements [/iot bluetooth scanners advertisements print detail as-value where\
\_\\\r\
\naddress ~ \$addressRegex and \\\r\
\ndata ~ \$advertisingDataRegex and \\\r\
\nrssi > \$rssiThreshold]\r\
\n\r\
\n/iot/bluetooth/scanners/advertisements clear\r\
\n\r\
\n:foreach adv in=\$advertisements do={\r\
\n:local address (\$adv->\"address\")\r\
\n:local ad (\$adv->\"data\")\r\
\n:local rssi (\$adv->\"rssi\")\r\
\n:local epoch (\$adv->\"epoch\")\r\
\n:local temp [\$from88 [:pick \$ad 28 32]]\r\
\n                \r\
\n:local recordStr [\$makeRecord ts=\$epoch gwName=\$gwName temp=\$temp]\r\
\n\r\
\n:if ([:len (\$macRecords->\$address)] > 0) do={\r\
\n:local str (\$macRecords->\$address)\r\
\n:local newStr \"\$str,\$recordStr\"\r\
\n:set (\$macRecords->\$address) \$newStr} else={:set (\$macRecords->\$address) \$recordStr\
}}\r\
\n\r\
\n# TODO: add some logic to decide when we want to send data\r\
\n:local sendData true\r\
\n\r\
\n:if (\$sendData) do={\r\
\n:local jsonStr \"{\"\r\
```

```
    \n\r\
    \n:foreach addr,advRec in=\$macRecords do={\r\
    \n:set jsonStr \"\$jsonStr\\\"\$addr\\\":[\$advRec],\"}\r\
    \n\r\
    \n:local payloadlength\r\
    \n:set payloadlength [:len (\$jsonStr)]\r\
    \n:local remcom\r\
    \n:set remcom [:pick \$jsonStr 0 (\$payloadlength-1)]\r\
    \n:set jsonStr \"\$remcom}\"\r\
    \n:local message\r\
    \n:set message \"\$jsonStr\"\r\
    \n:log info \"\$message\";\r\
    \n:put (\"[*] Message structured: \$message\")\r\
    \n:put (\"[*] Total message size: \$[:len \$message] bytes\")\r\
    \n:put (\"[*] Sending message to MQTT broker...\")\r\
    \n/iot mqtt publish broker=\"\$broker\" topic=\"\$topic\" message=\$message}"
```

In this case, the JSON message would look like this:

```
{
  "2C:C8:1B:4B:BB:0A": [
    {
      "ts": 1680527467840,
      "values": {
        "KNOT_1": "1",
        "temp": 26460
      }
    }
  ],
  "DC:2C:6E:0F:C0:3D": [
    {
      "ts": 1680527464996,
      "values": {
        "KNOT_1": "1",
        "temp": 24750
      }
    },
    {
      "ts": 1680527474996,
      "values": {
        "KNOT_1": "1",
        "temp": 24750
      }
    }
  ]
}
```

Because of the fact that floating point is not supported → every calculation behind a decimal point will be "rounded up" to a whole number. This is why the script will calculate the temperature and acceleration values **scaled by 1000** (multiplied by **1000**).
So, if you see the temperature as **temp=24750**, the real temperature is **24.750 C**. To add a decimal point, you will need to do an additional result "translation" on the server side or with additional scripting on the RouterOS side, which will increase the CPU usage of the device.

Script that includes GPS data (optional)

In case you also wish to include GPS data (longitude and latitude values from the KNOTs), use the script shown below:

```
/system script add dont-require-permissions=no name=tracking+gps+temp owner=admin policy=\
    ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="# Required package\
    s: iot\r\
    \n\r\
    \n################################ Configuration ################################\r\
    \n# Name of an existing MQTT broker that should be used for publishing\r\
    \n:local broker \"tb\"\r\
    \n\r\
    \n# MQTT topic where the message should be published\r\
    \n:local topic \"v1/gateway/telemetry\"\r\
    \n:local gwtopic \"v1/devices/me/telemetry\"\r\
    \n\r\
    \n# POSIX regex for filtering advertisement Bluetooth addresses. E.g. \"^BC:33:AC\"\r\
    \n# would only include addresses which start with those 3 octets.\r\
    \n# To disable this filter, set it to \"\"\r\
    \n:local addressRegex \"\"\r\
```

```
\n\r\
\n# POSIX regex for filtering Bluetooth advertisements based on their data. Same\r\
\n# usage as with 'addressRegex'.\r\
\n:local advertisingDataRegex \"15ff\"\r\
\n\r\
\n# Signal strength filter. E.g. -40 would only include Bluetooth advertisements\r\
\n# whose signal strength is stronger than -40dBm.\r\
\n# To disable this filter, set it to \"\"\r\
\n:local rssiThreshold \"\"\r\
\n\r\
\n#Name the KNOT. Identity of the unit that will be senting the message. This name will be \
reported to the MQTT broker.\r\
\n:local gwName \"2\"\r\
\n\r\
\n############GPS#############\r\
\n:global lat\r\
\n:global lon\r\
\n\r\
\n/interface ppp-client set ppp-out1 disabled=yes\r\
\n:log info (\"disabling WWAN to get GPS coordinates\")\r\
\n\r\
\n/interface ppp-client at-chat ppp-out1 input=\"AT+QGPSCFG=\\\"priority\\\",0\"\r\
\n:log info (\"enabling priority for GPS\")\r\
\n\r\
\n###the time in the delay below is the time that the device will wait for to get the coord\
inate fix\r\
\n:delay 32000ms\r\
\n:log info (\"reading GPS coordinates\")\r\
\n/system gps monitor once do={\r\
\n:set \$lat \$(\"latitude\")\r\
\n:set \$lon \$(\"longitude\")\r\
\n}\r\
\n:if (\$lat != \"none\") do={\\\r\
\n:log info (\"enabling priority back to WWAN\")\r\
\n/interface ppp-client at-chat ppp-out1 input=\"AT+QGPSCFG=\\\"priority\\\",1\"\r\
\n:log info (\"enabling WWAN\")\r\
\n/interface ppp-client set ppp-out1 disabled=no\r\
\n:delay 1000ms\r\
\n###if dial on demand is enabled\r\
\n/ping 1.1.1.1 count=1\r\
\n\r\
\n#the delay below waits for 5 seconds for the ppp connection to get established - this tim\
e can differ based on the signal strength\r\
\n:delay 5000ms\r\
\n:log info (\"posting coordinates via mqtt\")\r\
\n:local gpsmessage \\\r\
\n    \"{\\\"latitude\\\":\$lat,\\\r\
\n    \\\"longitude\\\":\$lon}\"\r\
\n/iot mqtt publish broker=\$broker topic=\$gwtopic message=\$gpsmessage} else={\\\r\
\n:log info (\"could not read GPS coordinates...enabling back WWAN\")\r\
\n/interface ppp-client at-chat ppp-out1 input=\"AT+QGPSCFG=\\\"priority\\\",1\"\r\
\n/interface ppp-client set ppp-out1 disabled=no\r\
\n:delay 1000ms\r\
\n###if dial on demand is enabled\r\
\n/ping 1.1.1.1 count=1\r\
\n:delay 5000ms\r\
\n}\r\
\n\r\
\n##################################Bluetooth###################################\r\
\n:global invertU16 do={\r\
\n    :local inverted 0\r\
\n    :for idx from=0 to=15 step=1 do={\r\
\n        :local mask (1 << \$idx)\r\
\n        :if (\$1 & \$mask = 0) do={\r\
\n            :set \$inverted (\$inverted | \$mask)\r\
\n        }\r\
\n    }\r\
\n    return \$inverted\r\
\n}\r\
\n\r\
\n:global le16ToHost do={\r\
```

```
\n    :local lsb [:pick \$1 0 2]\r\
\n    :local msb [:pick \$1 2 4]\r\
\n\r\
\n    :return [:tonum \"0x\$msb\$lsb\"]\r\
\n}\r\
\n:local from88 do={\r\
\n    :global invertU16\r\
\n    :global le16ToHost\r\
\n    :local num [\$le16ToHost \$1]\r\
\n\r\
\n    # Handle negative numbers\r\
\n    :if (\$num & 0x8000) do={\r\
\n        :set num (-1 * ([\$invertU16 \$num] + 1))\r\
\n    }\r\
\n\r\
\n    # Convert from 8.8. Scale by 1000 since floating point is not supported\r\
\n    :return ((\$num * 125) / 32)\r\
\n}\r\
\n:put (\"[*] Gathering Bluetooth info...\")\r\
\n\r\
\n:global makeRecord do={\r\
\n    :local jsonStr \"{\\\"ts\\\":\$ts,\\\"values\\\":{\\\"KNOT_\$gwName\\\":\\\"\$gwName\
\\\",\\\"temp\\\":\$temp}}\"\r\
\n    :return \$jsonStr\r\
\n}   \r\
\n\r\
\n# array of record strings collected for each advertising MAC address\r\
\n:global macRecords [:toarray \"\"]\r\
\n\r\
\n# process advertisements and update macRecords\r\
\n:local advertisements [/iot bluetooth scanners advertisements print detail as-value where\
\_\\\r\
\naddress ~ \$addressRegex and \\\r\
\ndata ~ \$advertisingDataRegex and \\\r\
\nrssi > \$rssiThreshold]\r\
\n\r\
\n/iot/bluetooth/scanners/advertisements clear\r\
\n\r\
\n:foreach adv in=\$advertisements do={\r\
\n:local address (\$adv->\"address\")\r\
\n:local ad (\$adv->\"data\")\r\
\n:local rssi (\$adv->\"rssi\")\r\
\n:local epoch (\$adv->\"epoch\")\r\
\n:local temp [\$from88 [:pick \$ad 28 32]]\r\
\n               \r\
\n:local recordStr [\$makeRecord ts=\$epoch gwName=\$gwName temp=\$temp]\r\
\n\r\
\n:if ([:len (\$macRecords->\$address)] > 0) do={\r\
\n:local str (\$macRecords->\$address)\r\
\n:local newStr \"\$str,\$recordStr\"\r\
\n:set (\$macRecords->\$address) \$newStr} else={:set (\$macRecords->\$address) \$recordStr\
}}\r\
\n\r\
\n# TODO: add some logic to decide when we want to send data\r\
\n:local sendData true\r\
\n\r\
\n:if (\$sendData) do={\r\
\n:local jsonStr \"{\"\r\
\n\r\
\n:foreach addr,advRec in=\$macRecords do={\r\
\n:set jsonStr \"\$jsonStr\\\"\$addr\\\":[\$advRec],\"}\r\
\n\r\
\n:local payloadlength\r\
\n:set payloadlength [:len (\$jsonStr)]\r\
\n:local remcom\r\
\n:set remcom [:pick \$jsonStr 0 (\$payloadlength-1)]\r\
\n:set jsonStr \"\$remcom}\"\r\
\n:local message\r\
\n:set message \"\$jsonStr\"\r\
\n:log info \"\$message\";\r\
\n:put (\"[*] Message structured: \$message\")\r\
```

```
\n:put (\"[*] Total message size: \$[:len \$message] bytes\")\r\
\n:put (\"[*] Sending message to MQTT broker...\")\r\
\n/iot mqtt publish broker=\"\$broker\" topic=\"\$topic\" message=\$message}"
```

This is where you need to keep in mind the BG77 modem behavior → BG77 cellular modem (used in the KNOT) can not be used to have a cellular CAT-M /NB-IoT ongoing connection and to obtain GPS coordinates at the same time.

When the script is run:

- PPP interface is disabled and the highest priority is set for GPS reception (while the cellular PPP interface is down) for 32 seconds (this time can be altered in the script);
- (a) If the device managed to obtain GPS latitude value during the configured 32 seconds (if the value obtained equals anything other than "none"), the script will structure a JSON message with captured latitude and longitude values, the script will set the highest priority for WWAN (change the priority from GPS to WWAN) and enable PPP interface back (for internet access). After that, the script will send GPS data JSON message via the first MQTT publish and, the script will run the Bluetooth part, where the Bluetooth data JSON message is structured and sent via the second MQTT publish (x2 MQTT messages are sent → GPS data MQTT message and Bluetooth data MQTT message);
- (b) If the device fails to obtain GPS latitude value during the 32-second interval (if the value obtained equals "none"), the script sets the highest priority for WWAN (changes the priority from GPS to WWAN), enables PPP interface back (for internet access) and just processes the Bluetooth part, where the Bluetooth data JSON message is structured and sent via MQTT publish (basically, if GPS data could not be obtained, x1 MQTT message with Bluetooth data is sent).

Scheduler

Apply a scheduler to the script, so that RouterOS periodically initiates the script by itself:

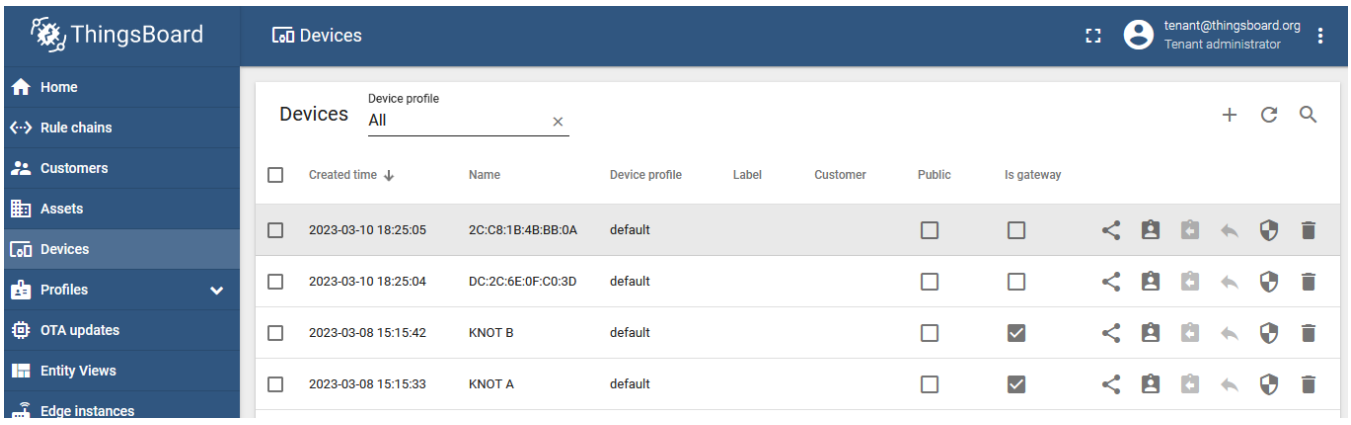
```
/system/scheduler/add name=bluetoothscheduler interval=50s on-event="/system/script/run tracking"
```

You can set up shorter and longer intervals. If you want to send data more often, so that the data is "fresher" → set up shorter time intervals (10-15 seconds). If you want to send fewer messages, less often → you can set up longer time intervals (30min+).

The JSON message structured using the script has a "`ts`" value (timestamp) assigned for each payload received. Meaning that **when the script is run**, for example, **every minute**, 1 tag is used and **the tag broadcasts** 1 payload every 10 seconds (that is **6 payloads per minute**) → ThingsBoard data (GUI) will be updated every minute, and every minute, 6 new entries will appear (each entry will indicate that it was received 10 seconds after the previous one). And if you send the message every 15 minutes when using 1 tag that is broadcasting a payload every 10 seconds (that is 6*15=90 payloads per 15 minutes) → ThingsBoard data (GUI) will be updated every 15 minutes but 90 entries will appear.

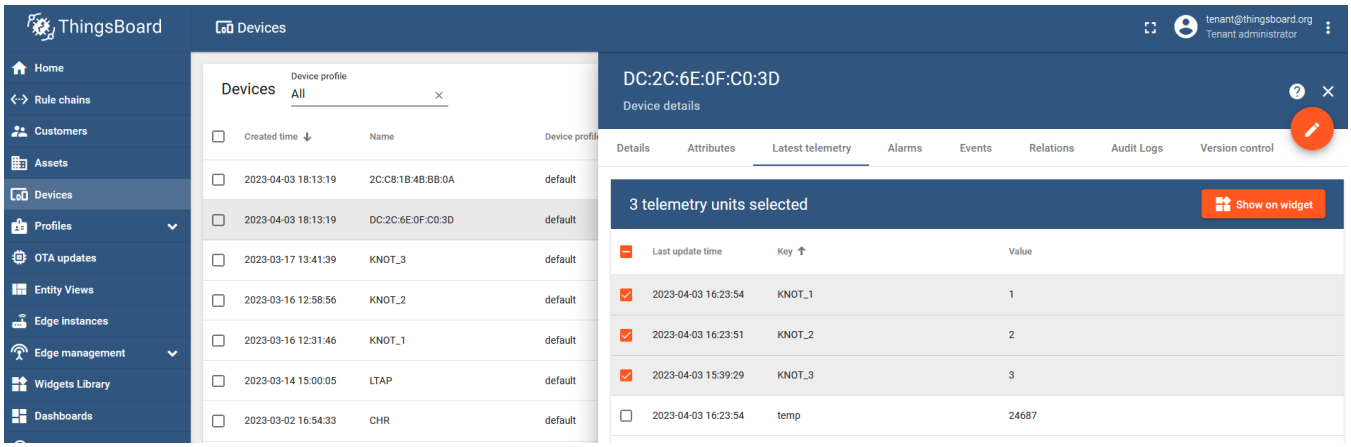
ThingsBoard data visualization and result verification

After you run the script with `/system script run tracking` or via a scheduler and refresh the GUI portal → all MAC addresses (tags) that are found in the JSON message, will be made into new devices under the ThingsBoard GUI:

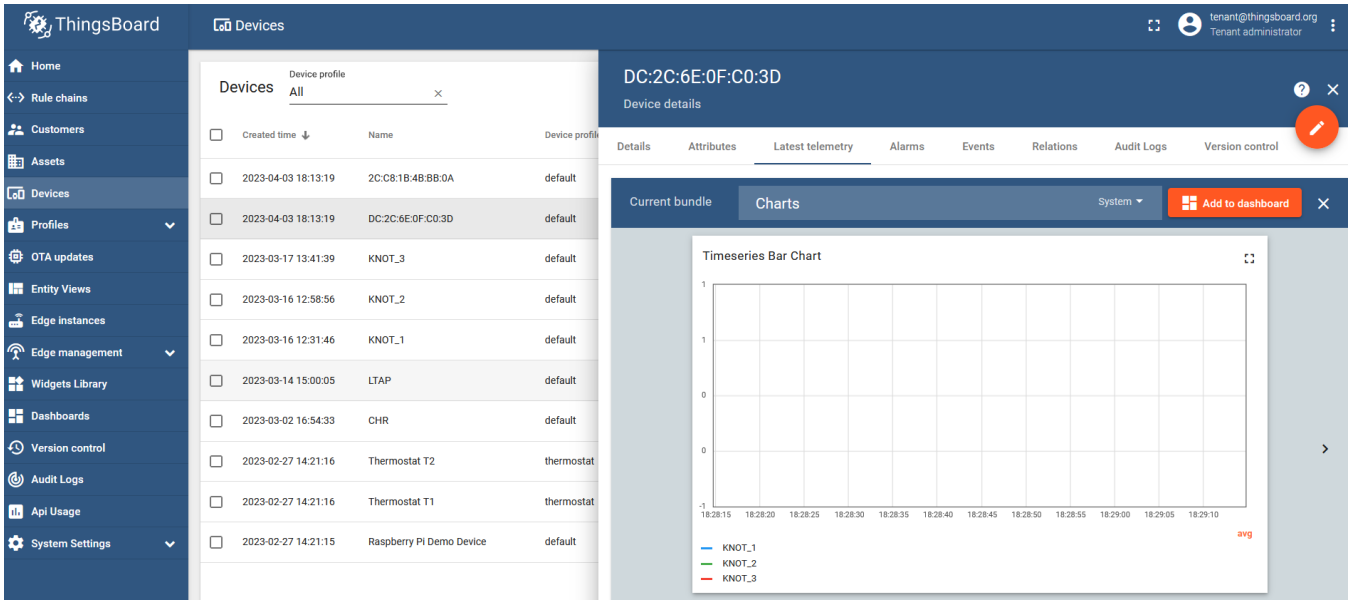


To help you visualize the data, you can use the built-in widgets or create your own one.

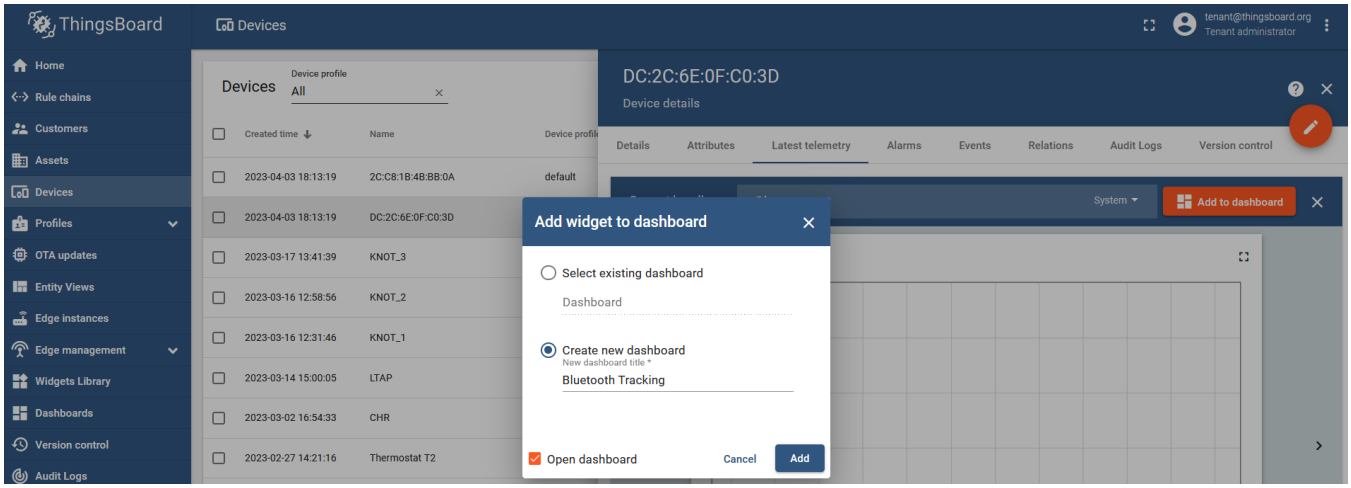
Select the tag's MAC address from the list of devices, go to the "Latest telemetry" section, checkbox KNOT IDs that you wish to monitor, and click on the "Show on widget" button:



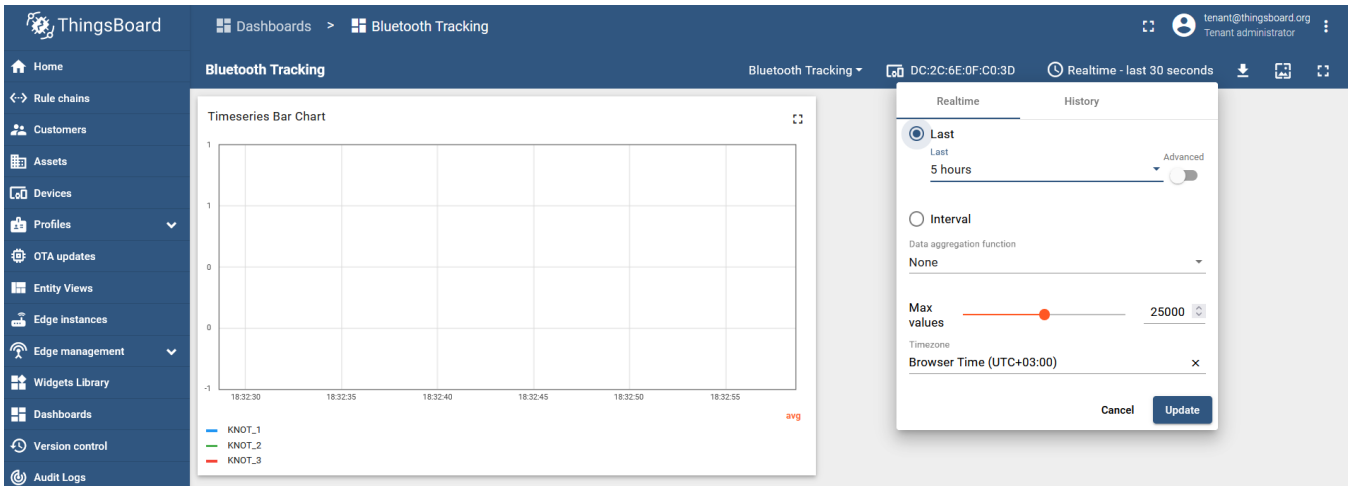
Select a widget that you wish to use, for example under the "Charts" bundle, "Timeseries Bar Chart" and click on "Add to dashboard":



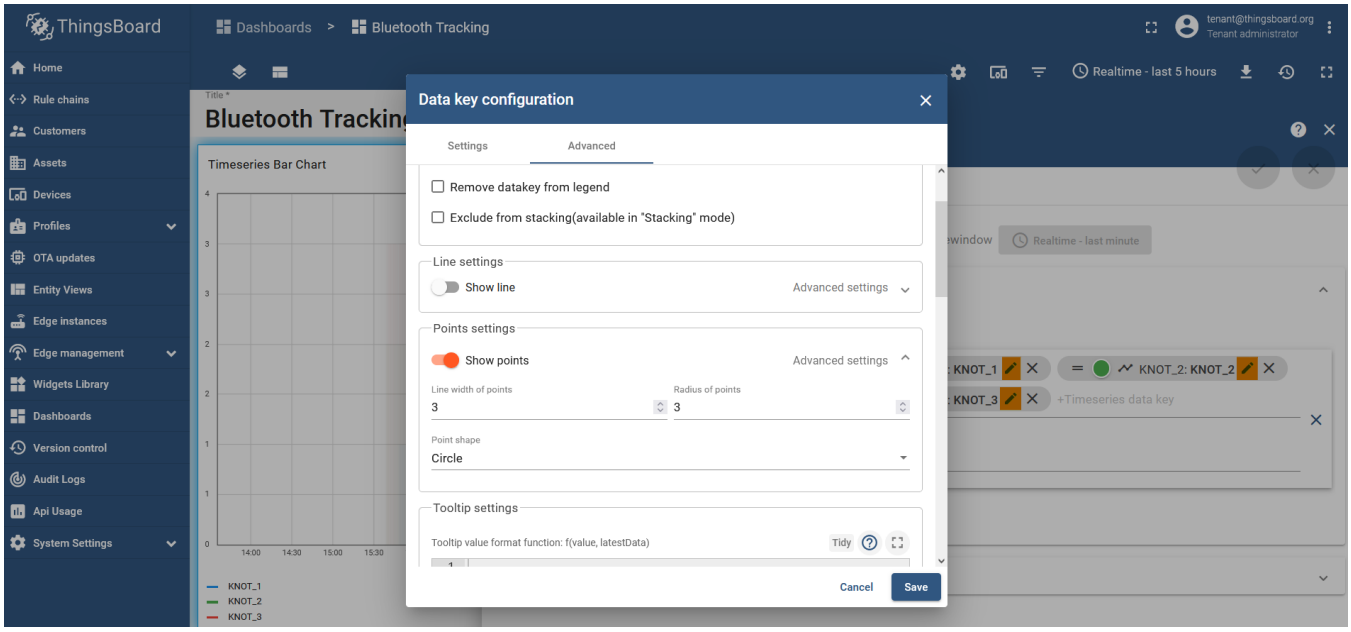
Create a new dashboard and name it, however, you like. Click on "Add":



Change the widget's "Timewindow" from "Realtime-last minute" (which is used by default) to "Realtime-last 5 hours" and disable "data aggregation function" (select "none"):

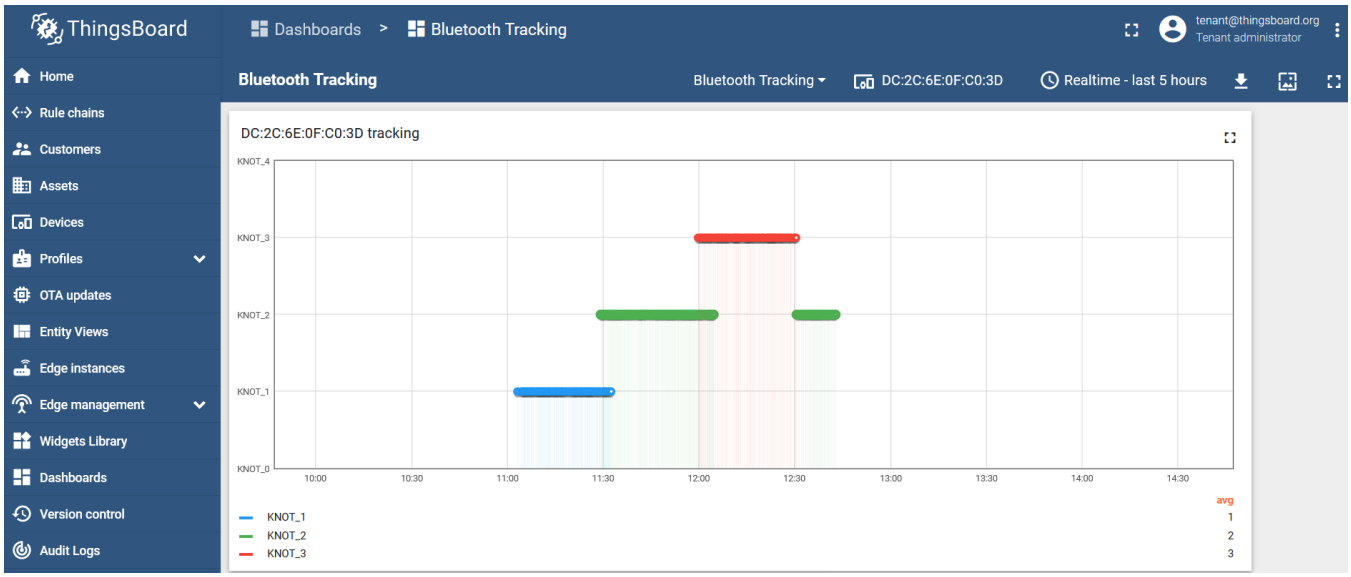


To help you better visualize the result, edit the widget and then edit each "KNOT_X" parameter/key. Enable the "Show points" checkbox for each key:



Check the ThingsBoard widget guide for more options that you have.

The end result would look like this:

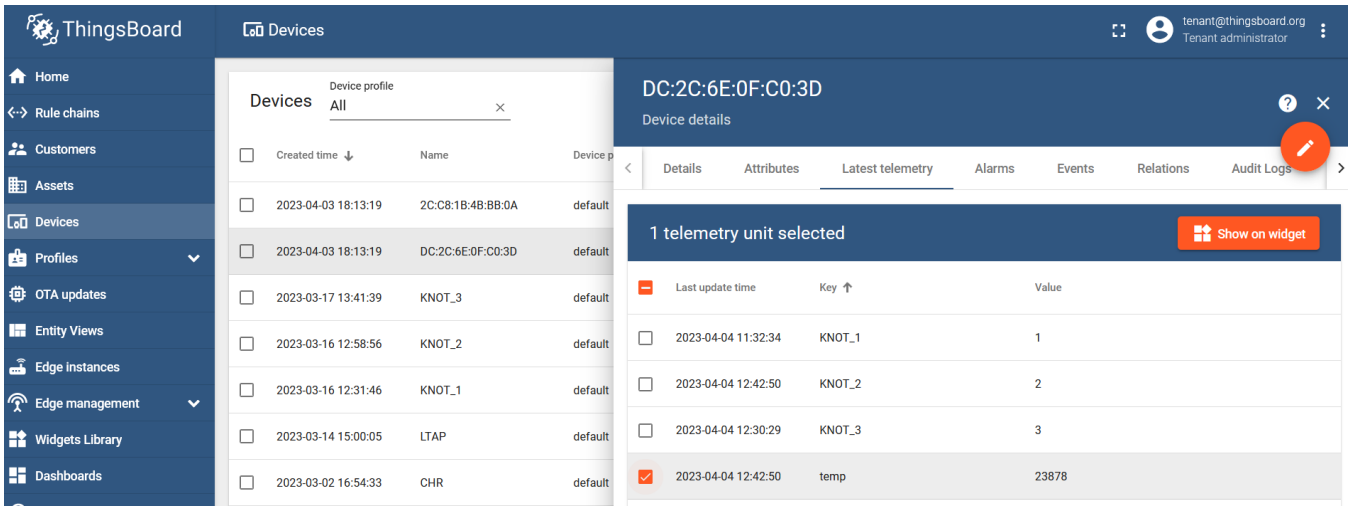


Per the dashboard, we can tell that:

- from ~11:00 to ~11:30, our asset was inside KNOT_1 Bluetooth range (inside warehouse #1);
- from ~11:30 to ~11:35, our asset was relocated to the vehicle (KNOT_2) that was parked near the warehouse (the tag was inside both KNOT's ranges);
- from ~11:35 to ~12:00, the tag was inside the truck (KNOT_2) - traveling to another warehouse;
- from ~12:00 to ~12:05, the asset was parked outside of warehouse #2, and it was inside both KNOT_2 and KNOT_3 ranges at the same time;
- from ~12:05 to 12:30, our asset was stored inside warehouse #2 (KNOT_3);
- from ~12.:30 onwards, the tag was on the road again, inside the truck (KNOT_2).

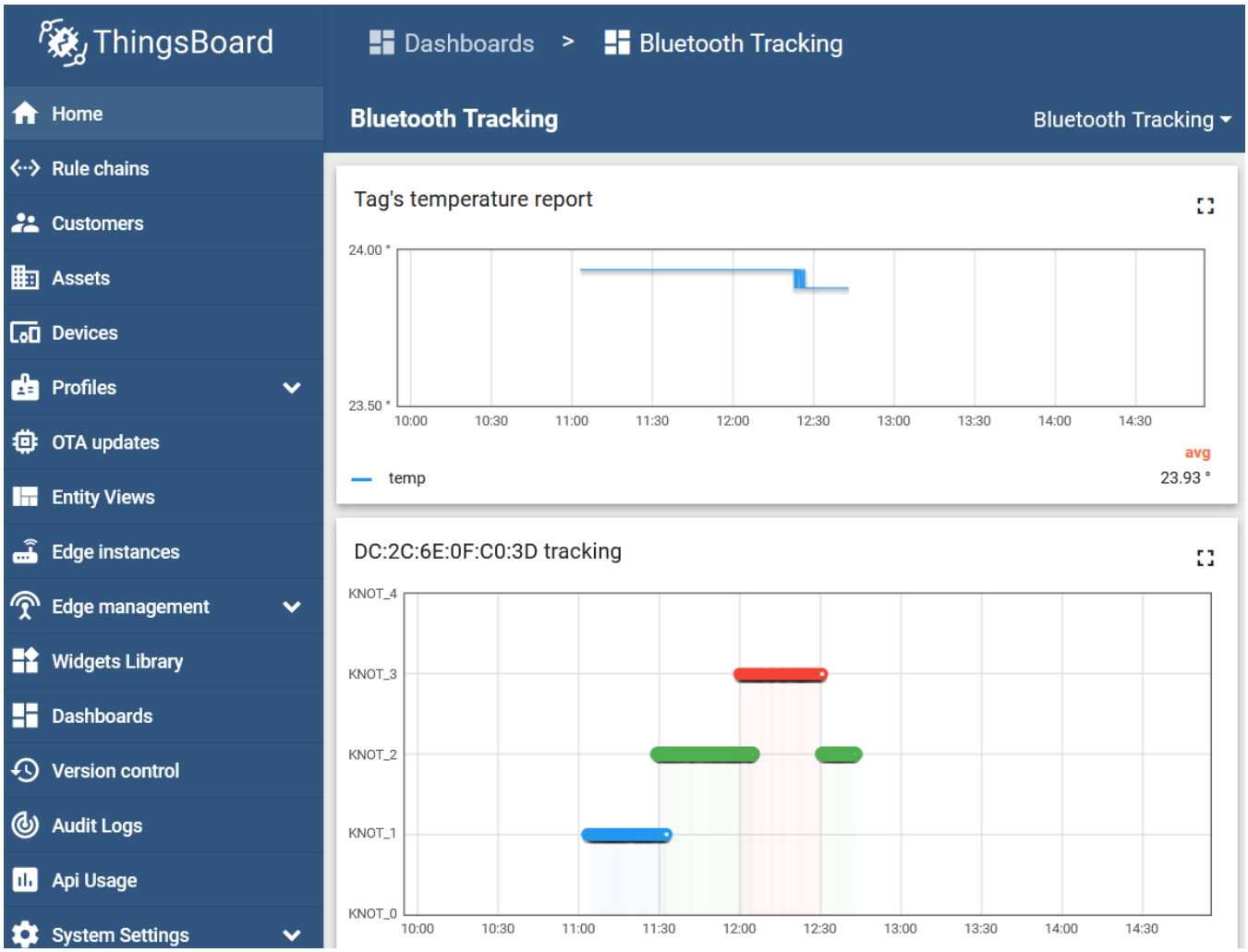
Temperature visualization (optional)

Select the tag's MAC address from the list of devices, go to the "Latest telemetry" section, checkbox "`temp`" parameter, and click on the "Show on widget" button:



Select a widget that you wish to use, for example under the "Charts" bundle, "Timeseries Line Chart". Click on "Add to dashboard", and choose the dashboard where you want to add the widget.

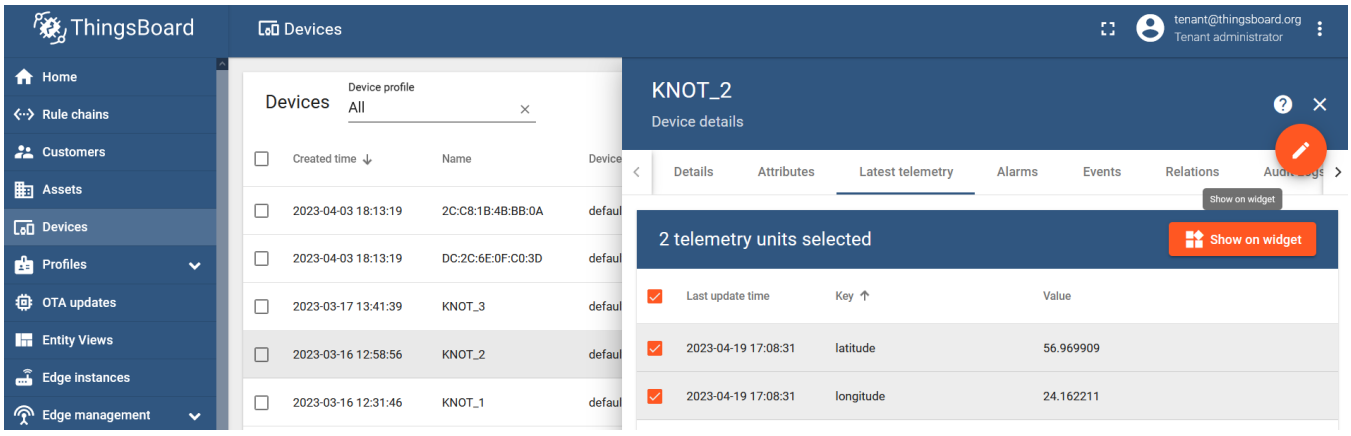
The result would look like this:



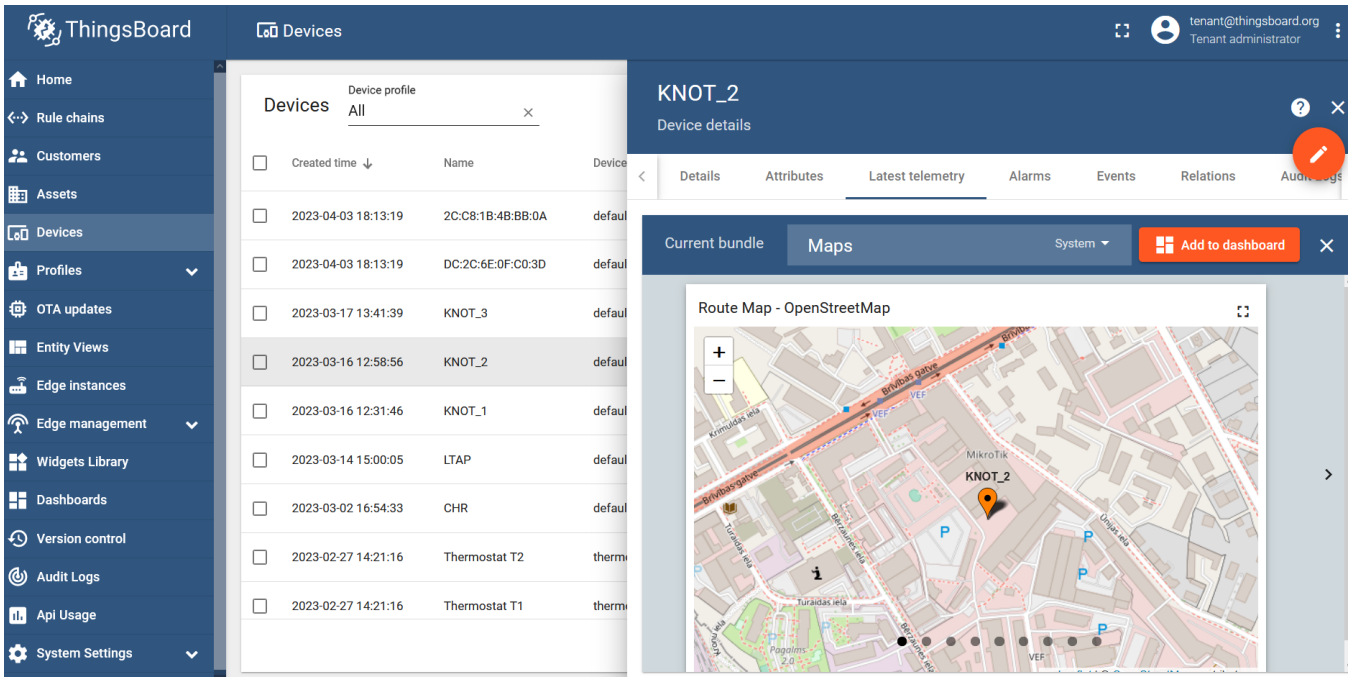
Now you have an additional graph that indicates how the tag's temperature changes during different time intervals.

GPS coordinate visualization (optional)

Per the script in the Script that includes GPS data section, the script sends x2 MQTT messages. Each message is sent to a different MQTT topic. GPS coordinate message will be posted to a topic named "1/devices/me/telemetry", while Bluetooth data will be posted to a topic named "v1/gateway /telemetry". Coordinates will be available to you under the ThingsBoard device list, under the specific gateway:

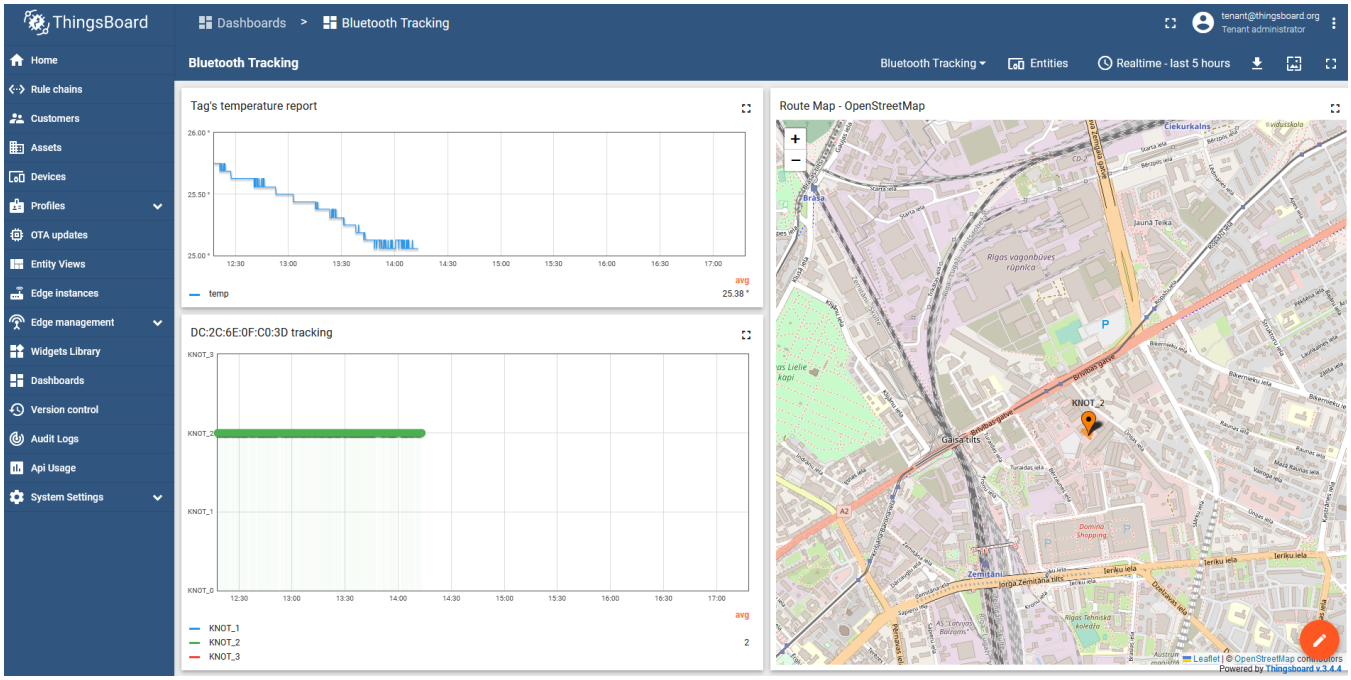


Checkbox both "latitude" and "longitude" parameters, click on the "Show on widget button", select "Current bundle" to "Maps", and choose the "Route Map - OpenStreetMap" widget:



To finish things up, click on the "Add to dashboard" button and choose the dashboard where you want the widget to be shown.

After adding 3 widgets into 1 dashboard (temperature line chart, Bluetooth reporter bar chart, and GPS coordinates map), you would get something similar to this:



- You will have a graph showing temperature changes (the tag's surrounding temperature);
- You will have the chart that indicates which specific KNOT has sent the report, that tells you in which KNOT's Bluetooth range the tag is currently in;
- You will have a map that shows the GPS position of the KNOT.

GPIO

***note**: In order to access GPIO settings, make sure that **iot** package is installed beforehand.*

You can find more information about GPIO following the link.

GPIO stands for General-Purpose Input/Output. It is a digital signal pin/pins on the routerboard that allows you to send/receive the signal. It can be useful in different scenarios, like:

1. Measuring voltage through ADC input
2. Reading 0 and 1 signal received from another device - "dry contact"
3. Controlling connected relays by sending logical 0 or 1 signal to the pin

RouterOS configuration

***note**:* GPIO settings are available only using CLI.

Sub-menu: `/iot gpio`

GPIO settings are divided into:

- analog (/iot gpio analog)
- digital (/iot gpio digital)

***note**:* in our examples, we are using KNOT as a reference device. Other devices may have a different pinout but the same principles apply.

> Please note that long-term (6.47.10) and stable (6.48.3) versions, you have to use "/system gpio", command structure remains the same as in " /iot gpio" examples, aside from "analog" and "digital" sub-menus, which were added in later versions. Versions 6.49beta54+ and RouterOS v7, use "/iot gpio" sub-menu.

/iot gpio analog

***note**:* please check on a product page whether your hardware supports analog input or not.

In the "analog" setting you can measure voltages on the analog input/ADC input pins:

```
[admin@device] /iot gpio analog> print
 # NAME                                                                  VALUE       OFFSET
 0 pin2                                                                  0mV         0mV
 1 pin3                                                                 32mV         0mV
```

"OFFSET" can be used to manually compensate voltage drop on the wires. "VALUE" is measured with:

```
value = adc_input + offset
```

, where adc_input is the voltage on the pin.

"OFFSET" configuration example is shown below:

```
[admin@device] /iot gpio analog> set pin2 offset

Offset ::= [-]Num[mV]
  Num ::= -2147483648..2147483647    (integer number)


[admin@device] /iot gpio analog> set pin2 offset 2
[admin@device] /iot gpio analog> print
 # NAME                                                                                                                                VALUE
OFFSET
 0 pin2
2mV          2mV
 1 pin3
0mV          0mV
```

/iot gpio digital

In the "digital" section you can send/receive a logical 0 or 1 signal using the digital output/input pins (output pins are "open drain"):

```
[admin@device] /iot gpio digital> print
Flags: X - disabled
 #   NAME                                        DIRECTION OUTPUT INPUT
SCRIPT
 0   pin5                                        input     0      0
 1   pin4                                        output    0
 2   pin6                                        output    0
```

"DIRECTION" for the pin can be either "input" (a pin that can receive the signal) or "output" (a pin that can send the signal).

> KNOT pin's "DIRECTION" for pin4 and pin6 can not be changed. Both pins are meant to be used only as "output" pins.

When the pin's direction is set to "output", you can configure the "OUTPUT" value. Changing the "OUTPUT" value sends the signal to the pin.

```
[admin@device] /iot gpio digital> set pin4 output=

Output ::= 0 | 1


[admin@device] /iot gpio digital> set pin4 output=1
[admin@device] /iot gpio digital> print
Flags: X - disabled
 #   NAME                                        DIRECTION OUTPUT INPUT
SCRIPT
 0   pin5                                        input     0      0
 1   pin4                                        output    1
 2   pin6                                        output    0
```

The "SCRIPT" field allows you to configure a script, that will be initiated whenever the "INPUT" or "OUTPUT" value changes (from 0 to 1 or from 1 to 0).

```
[admin@device] /iot gpio digital> set pin4 script=script1
[admin@device] /iot gpio digital> set pin5 script="/system .."
[admin@device] /iot gpio digital> print
Flags: X - disabled
 #   NAME                                    DIRECTION OUTPUT INPUT
SCRIPT
 0   pin5                                    input     0      0     /system
..
 1   pin4                                    output    1
script1
 2   pin6                                    output    0
```

Different scenarios

Controlling relays

One of the scenarios for the GPIO implementation is "controlling other relays" using digital output pins. Basically, sending "0" or "1" signal to the unit that is connected to the pin. To automate the process, you can use a scheduler, which will run the script at specific times.

For example, you can add the first script (a single line shown below) and name it "output=0":

> */iot gpio digital set pin4 output=0*

Then add a second script (a single line shown below) and name it "output=1":

> */iot gpio digital set pin4 output=1*

Having both scripts, you can configure a schedule:

```
[admin@device] /system scheduler> add name=run-30s interval=30s on-event="output=0"
```

The schedule configuration shown above will run the script with the name "output=0", every 30 seconds.

```
[admin@device] /system scheduler> add name=run-45s interval=45s on-event="output=1"
```

The schedule configuration shown above will run the script with the name "output=1", every 45 seconds.

As a result, the device will automatically send a signal to the 4th pin (digital output pin) with output value=0 every 30 seconds and a signal with output value=1 every 45 seconds.

You can change the scheduled time as you see fit (depending on the requirements).

Monitoring input signal

Another scenario is to "monitor input signal" using the digital input pins. You need a script that will initiate e-mail notification or MQTT/HTTPS (fetch) publish whenever the "INPUT" value changes for the pin with the direction="input" (whenever the RouterOS device receives a signal "0 or 1" from another device connected to the pin).

E-mail notification script:

> */tool e-mail send to=config@mydomain.com subject=[/system identity get name] body="$[/iot gpio digital get pin5 input]"*

After creating a script, apply/set it to the "input" pin:

```
[admin@device] /iot gpio digital> set pin5 script=script1
[admin@device] /iot gpio digital> print
Flags: X - disabled
 #   NAME                    DIRECTION OUTPUT INPUT SCRIPT
 0   pin5                    input     0      0     script1
 1   pin4                    output    0            script1
 2   pin6                    output    0
```

In the example above, the e-mail notification script is named "script1".

As a result, whenever the input value changes (from 0 to 1 or from 1 to 0), the script automatically initiates an e-mail notification that will display the input value in the e-mail body.

Do not forget to change the script line and configure the e-mail settings (/tool e-mail) accordingly:

```
/tool e-mail send to="config@mydomain.com" subject="[/system identity get name]"  body="$[/iot gpio digital get pin5 input]"
```

Configure the actual e-mail address that you use. You can also change the subject and the body for the mail as you see fit.

MQTT publish script:

```
:local broker "name"

:local topic "topic"

:local message "{\"inputVALUE\":$[/iot gpio digital get pin5 input]}"
/iot mqtt publish broker=$broker topic=$topic message=$message
```

This script works the same way as the "*e-mail notification*" script, only when the input value changes the script initiates MQTT publish (instead of e-mail notification) and sends the input value received on the pin in the JSON format.

Do not forget to set up MQTT broker (*/iot mqtt brokers add* ..) and alter a few script lines beforehand:

```
:local broker "name"
```

The broker's "name" should be changed accordingly (you can check all created brokers and their names using CLI command /*iot mqtt brokers print*).

```
:local topic "topic"
```

The topic should be changed as well. The topic itself is configured on the server-side, so make sure that the correct topic is used.

Do not forget to apply/set the script to pin5 (/iot gpio digital set pin5 script=script_name), as shown in the "email notification" example above.

If the mechanical switch is used to send the signal to the GPIO pin, it is suggested to use the following script instead (in case the script is initiated more than once when the signal is received on the pin):

```
:global gpioscriptrunning;
if (!$gpioscriptrunning) do={:set $gpioscriptrunning true;
:log info "script started - GPIO changed";
:do {if ([/iot gpio digital get pin5 input] = "0") do={/tool e-mail send to="config@mydomain.com" subject=[/system identity get name]
body="pin5 received logical 0"} else {/tool e-mail send to="config@mydomain.com" subject=[/system identity get name]  body="pin5
received logical 1"};
:delay 1s;
:set $gpioscriptrunning false} on-error={:set $gpioscriptrunning false;
:log info "e-mail error, resetting script state..."}}
```

If the GPIO pin state changes more than once within mili/microseconds - the script above is going to make sure that e-mail notification is not sent more than once.

Monitoring voltage

Last but not least - is to "monitor voltage" using the analog pins. You need a script that will read/monitor voltage on schedule and then send the data via e-mail, MQTT or HTTPS (fetch).

Create a script, as shown below. In this example, we will be using MQTT publish (but you can create a similar script with "/tool e-mail .." to use e-mail notifications):

```
:local broker "name"

:local topic "topic"

:local message "{\"voltage(mV)\":$[/iot gpio analog get pin3 value]}"
/iot mqtt publish broker=$broker topic=$topic message=$message
```

The script will read/measure the voltage on pin3 and publish the data to the MQTT broker.

Do not forget to set up MQTT broker (*/iot mqtt brokers add* ..) and alter a few script lines beforehand:

```
:local broker "name"
```

The broker's "name" should be changed accordingly (you can check all created brokers and their names using CLI command /*iot mqtt brokers print*).

```
:local topic "topic"
```

The topic should be changed as well. The topic itself is configured on the server-side, so make sure that the correct topic is used.

Save the script and name it, for example, "voltagepublish". To automate the process, you can use the scheduler.

```
[admin@device] /system scheduler> add name=run-45s interval=45s on-event="voltagepublish"
```

The schedule configuration shown above will run the script every 45 seconds.

Lora

General Properties

Every RouterBOARD with a miniPCI-e slot which supports LTE modems can also be used as a LoRaWAN gateway by installing **R11e-LoRa8** or **R11e-LoRa9** card. Both UDP and LNS (starting with **v7.12rc1** testing version) protocols are supported.

In order to work with Lora, IoT package should be installed. You can find the package for your device architecture in extra packages archive on the downloa page.

Starting with v**7.11** (stable), LoRa functionality is moved into the **IoT package** that is available on the download page under extra packages. A separate Lora package is still available for download.

When using IoT package, LoRa functionality will move to a `/iot lora` sub-menu. When using LoRa package, LoRa functionality will be possible via `/lora` sub-menu.

LoRa package is not obligatory anymore and is left only for compatibility reasons.

***note**:* RouterOS does not support 3rd party LoRaWAN gateway cards.

Properties

Sub-menu: `/iot lora`

Property	Description
antenna-gain (*integer [-128..127]*; Default: **0**)	Antenna gain in dBi.
channel-plan (*as-923 | au-915 | custom | eu-868 | in-865 | kr-920 | ru-864 | ru-864-mid | us-915-1 | us-915-2*; Default: **eu-868**)	Frequency plans for various regions.
disabled (*yes | no*; Default: **yes**)	Whether LoRaWAN gateway is disabled.
forward (*crc-disabled | crc-error | crc-valid*; Default: **crc-valid, crc-error**)	Defines what kind of packets should be forwarded to Network server: • crc-disabled - forward packets which CRC code isn`t checked • crc-error - forward packets with incorrect CRC code • crc-valid - forward valid packets with correct CRC.
gateway-id (string)	Gateway ID or Gateway EUI, is used when registering the gateway with the server.
lbt-enabled (*yes | no*; Default: **no**)	Whether gateway should use LBT (Listen Before Talk) protocol.
listen-time (*integer [0us..4294967295us]*; Default: **5000us**)	Time in microseconds to track RSSI before TX (used when **lbt-enabled=yes**).
name (*string*; Default:)	Name of LoRaWAN gateway.
network (*private | public*; Default: **public**)	Whether sync word should (network=private) or should not (network=public) be used.
rssi-threshold (*integer [-32,768 .. 32,767]*; Default: **-65dB**)	RSSI value to determine whether forwarder may use specific channel to talk. If RSSI value is below **rssi-threshold**, channel could be used (used when **lbt-enabled=yes**).
servers (*list of string*; Default:)	Name or names of servers from /lora servers.
src-address (*IP*; Default:)	Specifies uplink packet source address if necessary (address should match an address configured on the RB).

spoof-gps (*string*; Default:)	Set custom GPS location: • Latitude [-90..90] • Longitude [-180..180] • Altitude(**m**) [-2147483648..2147483647]

Channels

Sub-menu: `/iot lora channels`

Property	Description
bandwidth (*7.8_kHz* / *15.6_kHz* / *31.2_kHz* / *62.5_kHz* / *125_kHz* / *250_kHz* / *500_kHz*; Default: **125_kHz**)	Bandwidth of specific channel, predefined when any of channel-plan preset is used, but could be manually changed when channel-plan is set to custom.
disabled (*yes* / *no*; Default: **no**)	Disable or enable the channel.
freq-off (*integer* [-400000..400000]; Default:)	Channel frequency offset against radio central frequency, it makes possible to adjust channel frequencies so that channels does not overlap.
radio (*radio0* / *radio1*; Default:)	Defines which radio uses selected channel.
spread-factor (*SF7* / *SF8* / *SF9* / *SF10* / *SF11* / *SF12*; Default:)	Defines the Spread Factor for a channel with type=LoRa. Lower Spread Factor means higher data rate.

To view current channels, issue the command `/iot lora cannels print`:

```
/iot lora channels print
Columns: NAME, TYPE, RADIO, FREQ-OFF, BANDWIDTH, FREQ, SPREAD-FACTOR, DATARATE
# NAME       TYPE  RADIO   FREQ-OFF  BANDWIDTH  FREQ   SPREAD-FACTOR  DATARATE
0 gateway-0  MSF   radio1  -400000   125_kHz    868.1
1 gateway-0  MSF   radio1  -200000   125_kHz    868.3
2 gateway-0  MSF   radio1  0         125_kHz    868.5
3 gateway-0  MSF   radio0  -400000   125_kHz    867.1
4 gateway-0  MSF   radio0  -200000   125_kHz    867.3
5 gateway-0  MSF   radio0  0         125_kHz    867.5
6 gateway-0  MSF   radio0  200000    125_kHz    867.7
7 gateway-0  MSF   radio0  400000    125_kHz    867.9
8 gateway-0  LoRa  radio1  -200000   250_kHz    868.3  SF7
9 gateway-0  FSK   radio1  300000    125_kHz    868.8                    50000
```

Channels are created using `freq-off` and radio's `center-freq` frequencies. To view **radios** center frequencies use the command `/iot lora radios print`.

To understand how each channel's frequency is calculated, check the example below:

```
# NAME       TYPE  RADIO   FREQ-OFF  BANDWIDTH  FREQ   SPREAD-FACTOR  DATARATE
0 gateway-0  MSF   radio1  -400000   125_kHz    868.1
```

`radio1` is selected to be used for channel #0 and it is configured with `center-freq=868500000` (868500000 Hz or 868.5 MHz).

By using frequency offset, `freq-off=-400000` (-400000 Hz or -0.4 MHz), we define channel #0 to be `868500000-400000=868100000` Hz or 868.1 MHz.

To configure custom channels, select "custom" channel profle with the help of the command:

```
/iot lora set [find] channel-plan=custom
```

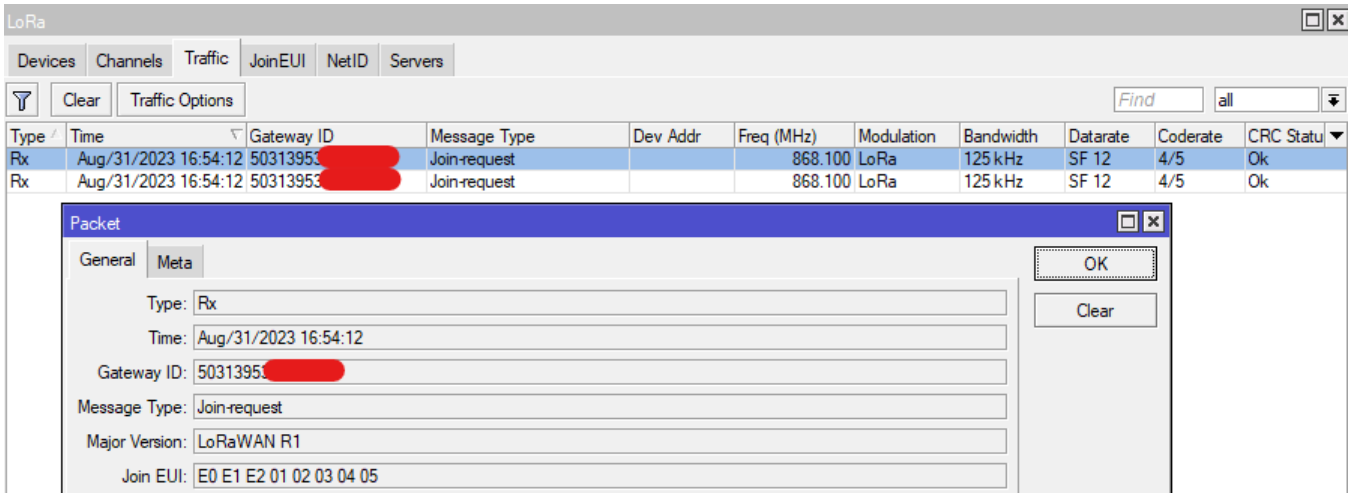
Join EUI

The gateway will forward to the server every single LoRaWAN payload it receives. That includes neighboring LoRaWAN node's payloads as well. It might not be ideal to forward everything, as, for example, it can increase the data amount used (and directly impact ISP plan cost).

The Join EUI menu allows you to specify a range of Join EUI's that the gateway will forward. After adding the range, make sure to apply it to the server settings.

If the Join EUI of the packet does not match the configuration, the packet is not forwarded to the server.

You can find the Join EUI used by your node with the help of RouterOS GUI. Go to the "LoRa" section and to the "Traffic" sub-menu (which is only available using the graphical interface). After you power your LoRaWAN node, the node should send a "Join-request" packet. Double-click on it to inspect it:



Sub-menu: `/iot lora joineui`

Property	Description
joineui (*string*; Default:)	Define a range of Join EUI's.
name (*string*; Default:)	Define the name for the range.

An example of Join EUI would look like this **E0 E1 E2 01 02 03 04 05**. It consists of 8 octets in HEX format.

To add a range that allows **every possible** Join EUI, add a filter like this:

```
/iot lora joineui add name=ALL joineuis=0000000000000000-ffffffffffffffff
```

To add a range that allows "nothing" (basically "restrict" all Join EUI's), add a filter like this:

```
/iot lora joineui add name=NONE joineuis=0000000000000000-0000000000000000
```

For a specific single Join EUI, add a filter like this:

```
/iot lora joineui add name=SINGLE joineuis=E0E1E20102030405-E0E1E20102030405
```

Network ID

The gateway will forward to the server every single LoRaWAN payload it receives. That includes neighboring LoRaWAN node's payloads as well. It might not be ideal to forward everything, as, for example, it can increase the data amount used (and directly impact ISP plan cost).

The NetID menu allows you to specify a list of NetIDs that the gateway will forward. After adding the list, make sure to apply it to the server settings.

If the NetID (DevAddr range) of the packet does not match the configuration, the packet is not forwarded to the server.

NetIDs define the ranges of Device Addresses (DevAddr) that were assigned to different operators/servers by the LoRaWAN Alliance. A list with most ranges can be found in the TTN guide.

DevAddr is assigned to the LoRaWAN node by the LoRaWAN server after the communication with the server takes place. For example, TTN will assign your node an address from within the range 26000000 - 27FFFFFF. You can find it under the LoRaWAN server dashboard or using RouterOS GUI, under the "Traffic" sub-menu (after "join-request" and "join-accept" communication takes place) in the Dev Addr column/field.

Let's say TTN assigned **26 1B D8 D1** Dev Addr to your node. Based on the TTN guide, it falls under the 26000000 - 27FFFFFF DevAddr range and it belongs to the **000013 NetID**.

Sub-menu: `/iot lora netid`

Property	Description
netids (*string*; Default:)	Define the NetIDs
name (*string*; Default:)	Define the name for the ID.

To add a filter for a specific NetId, use the command (you can add more than one using a "comma" separator):

```
/iot lora netids add name=TTN netids=000013
```

Servers

Sub-menu: `/iot lora servers`

There are a few predefined servers that can be used (it requires to make an The Things Network account to use them):

```
[admin@MikroTik] /iot/lora/servers> print
Columns: NAME, UP-PORT, DOWN-PORT, ADDRESS
#  NAME              UP-PORT  DOWN-PORT  ADDRESS
0  TTN-EU               1700       1700  eu.mikrotik.thethings.industries
1  TTN-US               1700       1700  us.mikrotik.thethings.industries
2  TTS Cloud (eu1)      1700       1700  eu1.cloud.thethings.industries
3  TTS Cloud (nam1)     1700       1700  nam1.cloud.thethings.industries
4  TTS Cloud (au1)      1700       1700  au1.cloud.thethings.industries
5  TTN V3 (eu1)         1700       1700  eu1.cloud.thethings.network
6  TTN V3 (nam1)        1700       1700  nam1.cloud.thethings.network
7  TTN V3 (au1)         1700       1700  au1.cloud.thethings.network
```

Custom servers can be added as well. Data forwarding to multiple servers can work simultaneously if the first server does not change "DevAdress" part of the packet and under the condition that all servers are able to decode the packet.

Property	Description
address (*domain name or IP address*; Default:)	Defines LoRaWAN Network server address.
name (*string*; Default:)	Defines server name.
protocol (*UDP | LNS | CUPS*; Default: **UDP**)	Specify whether to use UDP, LNS or CUPS protocol for the communication with the LoRaWAN server.

down-port (*integer [0..65535]*; Default: **1700**)	Parameter that is used when UDP protocol is selected. Defines port for down-link communication (from server to node) with LoRaWAN Network server. Most of known open source servers uses port 1700 as default, but it can change if multiple servers are configured on the same machine.
up-port (*integer [0..65535]*; Default: **1700**)	Parameter that is used when UDP protocol is selected. Defines port for up-link communication (from node to server) with LoRaWAN Network server. Most of known open source servers uses port 1700 as default, but it can change if multiple servers are configured on the same machine.
netid (*list of string*; Default:)	Parameter that is used when UDP protocol is selected. Applies a filter to only send LoRaWAN payloads that match the Network ID (Net ID) filter configured.
joineui (*list of string*; Default:)	Parameter that is used when UDP protocol is selected. Applies a filter to only send LoRaWAN payloads that match the Join EUI filter configured.
port (*integer [0..65535]*; Default: **8887**)	Parameter that is used when LNS or CUPS protocol is selected. For LNS, defines the WSS (WebSocket) port and, for CUPS, defines HTTPS port.
key (*string*; Default:)	Parameter that is used when LNS or CUPS protocol is selected. Specify the LoRa Basics Station LNS Authentication Key or CUPS API KEY (both generated on the server).
ssl (*yes or no*; Default: no)	Parameter that is used when LNS or CUPS protocol is selected. Specify whether to use or not to use SSL (if the server supports TLS server authentication). When this option is choosen, root SSL certificate(s) must be uploaded under the certificates menu.
certificate (*list of string*; Default: **none**)	Parameter that is used when LNS or CUPS protocol is selected. Select an uploaded client certificate (if the server awaits TLS client authentication). If TLS client authentication is not required by the server, use the default "**none**" setting.
interval (*integer [0..65535]*; Default:)	Parameter that is used when CUPS protocol is selected. Specify the interval with which the LoRa Basics Station will query CUPS server for configuration updates/changes.

Debugging

If you have issues with the connection, make sure to enable logs:

```
/system/logging/add topics=debug,lora
```

This will enable debug logging and help you pin point where the potential issue could be. Logs can be viewed using:

```
/log/print
```

A successful connection would look like this:

```
 13:50:33 lora,info gateway-0 forwarder started
 13:50:38 lora,info [LNS] connecting to wss://eu1.cloud.thethings.network:8887/router-info
 13:50:39 lora,info [LNS] eu1.cloud.thethings.network discovered
 13:50:39 lora,info [LNS] eu1.cloud.thethings.network disconnected
 13:50:39 lora,info [LNS] connecting to wss://eu1.cloud.thethings.network:8887/traffic/eui-xxxx
 13:50:39 lora,info [LNS] eu1.cloud.thethings.network configured
 13:50:52 lora,info gateway-0 forwarder is ready
```

More logging information can be found in our Log guide.

Setup

Step by step installation

AWS LoRaWAN configuration

The Things Network

The Things Stack

AWS LoRaWAN configuration

> This scenario will work starting with RouterOS version **7.14beta8**.

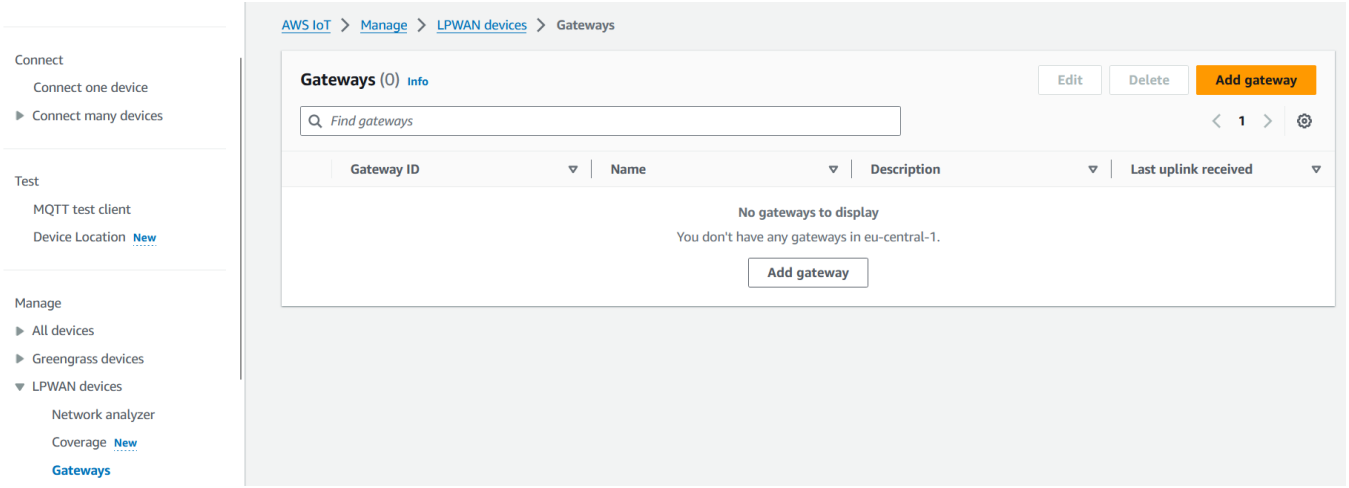
Before we proceed with the settings, you need to create an account in the AWS system. You can find more information on how to do that following this link.

After you are logged-in, go to **Services>IoT Core** section on the portal.

AWS - Registering the gateway

The first step is to register the LoRaWAN gateway.

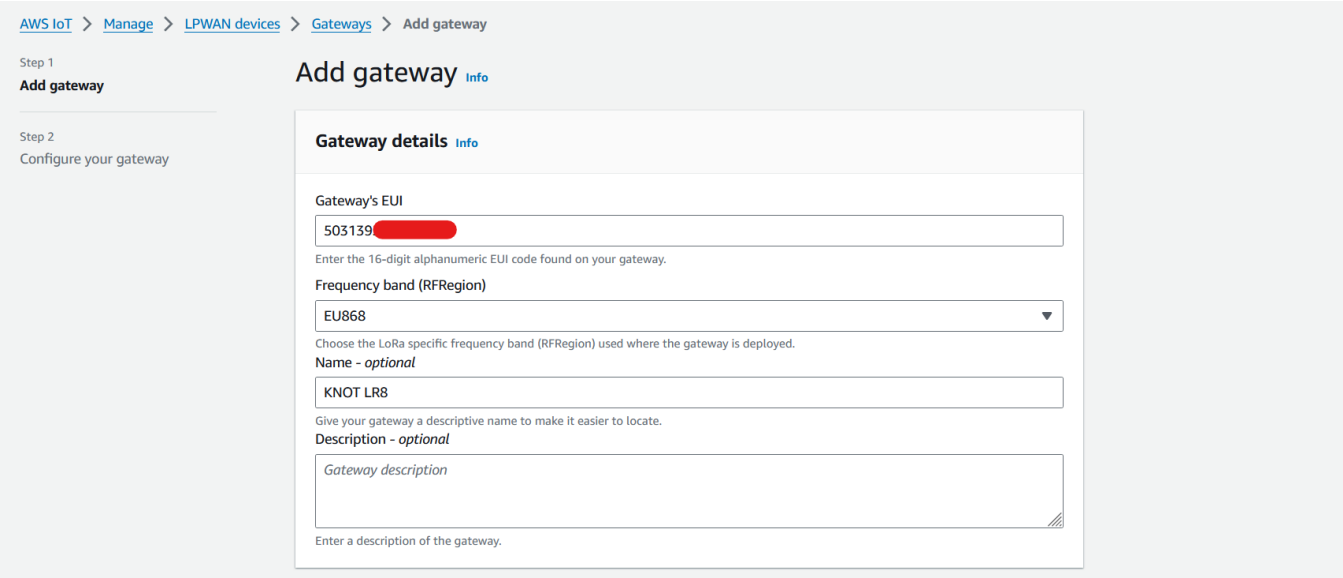
Navigate to the Gateways section (under LPWAN devices).



Click on the "**Add gateway**" button.

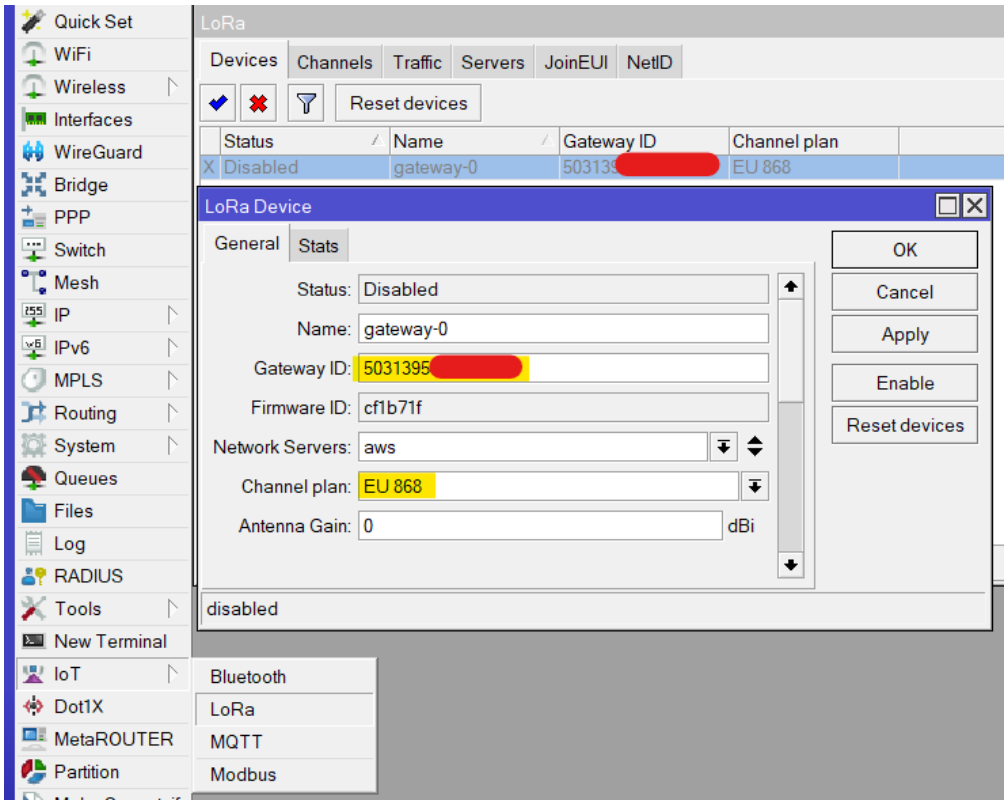
Step 1 - add gateway

- Input the gateway's EUI;
- Select device's frequency band;
- Configure optional fields if required;



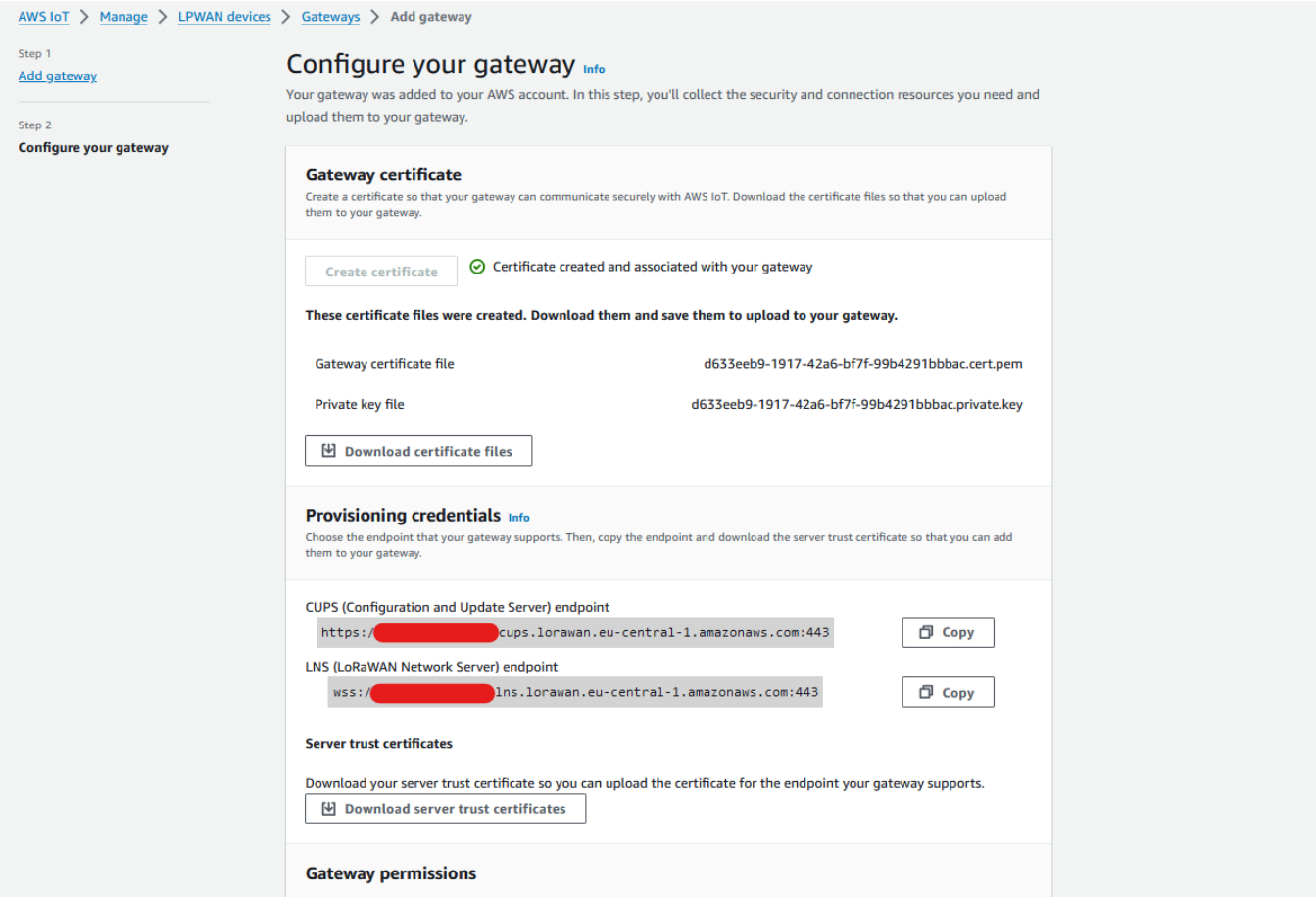
Finish the step by clicking on the "**Add gateway**" once again.

In RouterOS settings, gateway's EUI and frequency plan can be checked under **IoT>LoRa>Devices** tab:



Step 2 - configure your gateway

- Generate a gateway certificate ("**Create certificate**" button), and download the certificate file and private key files ("**Download certificate files**" button);
- Copy CUPS and LNS endpoints and download server trust certificates ("**Download server trust certificates**" button);
- Add suggested gateway permissions;



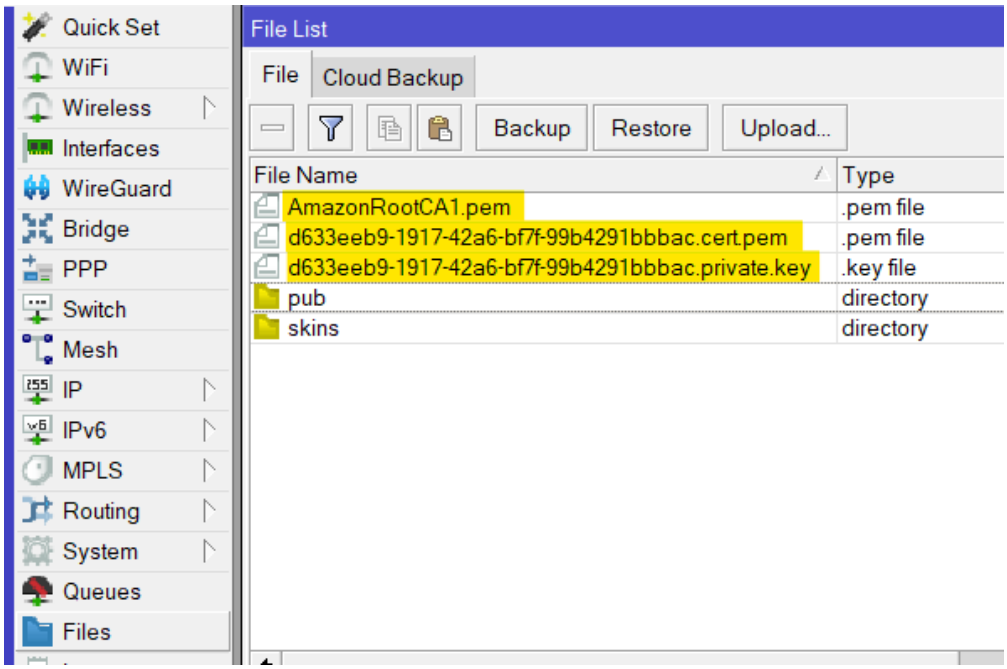
Finish the step by clicking on "**Submit**".

You will be redirected to the page where your newly created gateway should appear.

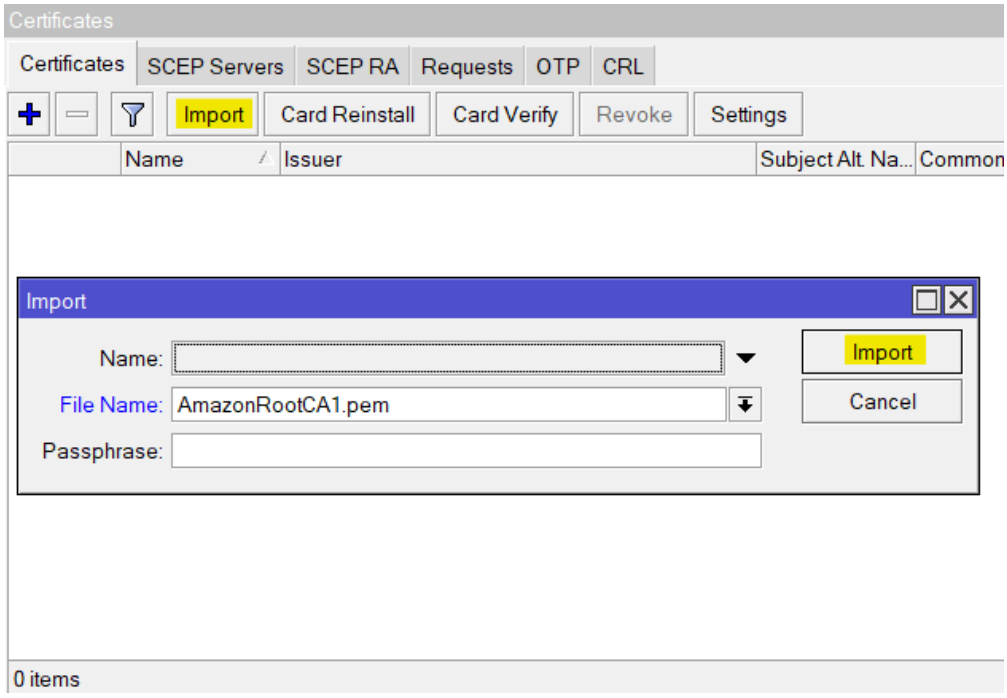
RouterOS - Connecting the gateway

Uploading and importing certificates

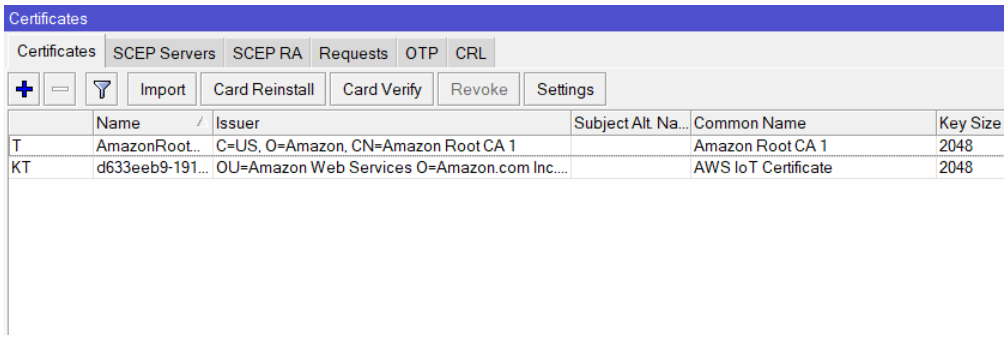
Before we proceed with the setup, you need to download Amazon Root CA and upload it, together with the gateway certificate file and its key, into the RouerOS file list menu:



After the files were uploaded, import the certificates, one by one (under **System>Certificates**):



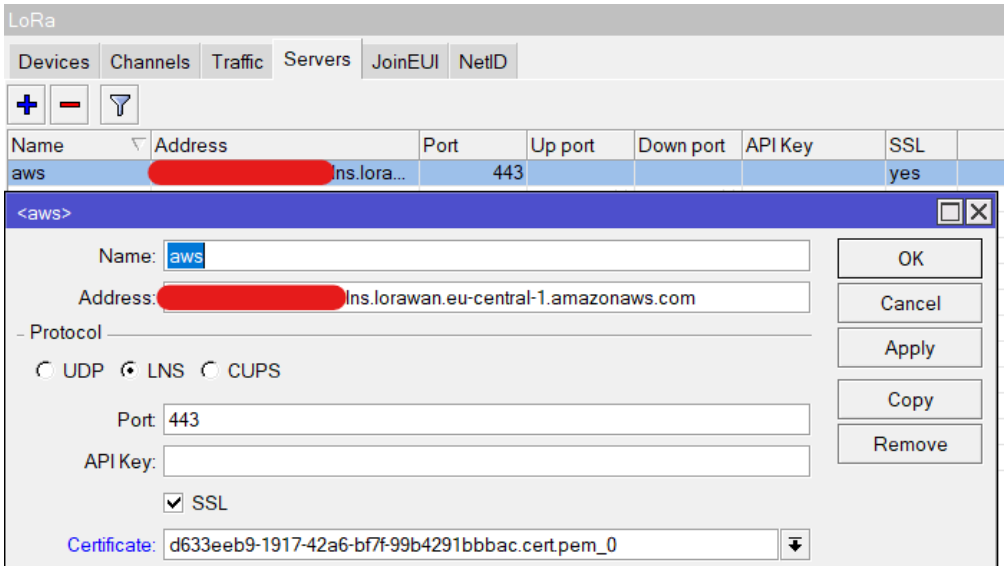
Make sure to upload the gateway certificate first and then its key (so that the gateway certificate has both K-key and T-trusted flags present). In the end, you should have all 3 file imported, like so:



Server configuration

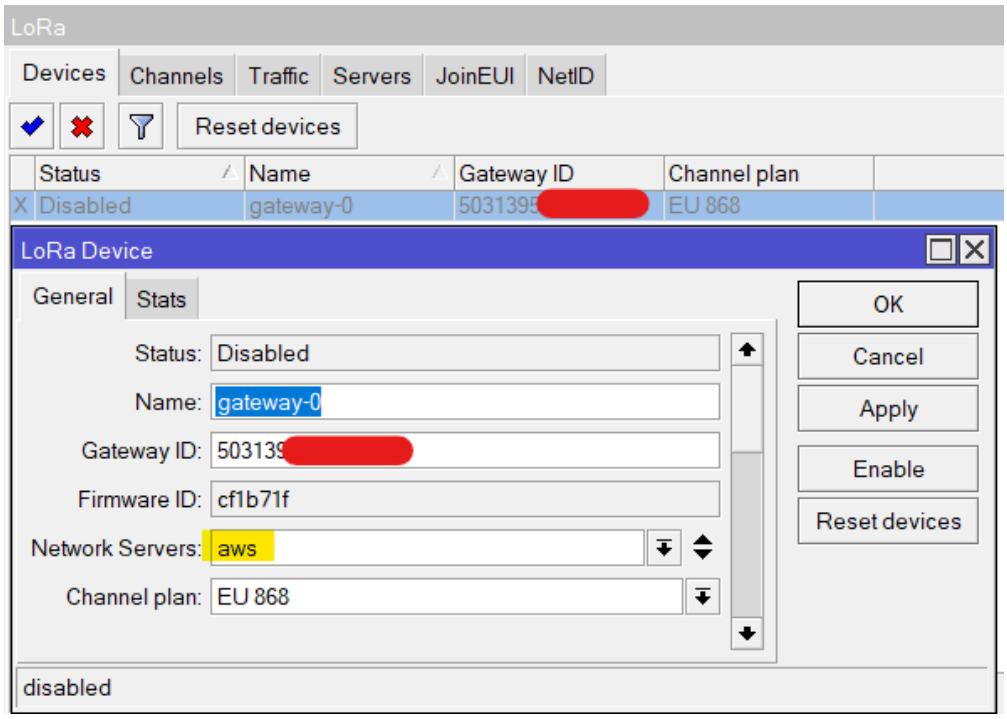
LNS scenario

Navigate to the **IoT>LoRa>Servers** tab and add a new server:



- Name the server;
- Input LNS endpoint address (without "`wss://`" and "`:443`");
- Select LNS protocol;
- Change port to "443";
- Enable SSL checkbox;
- Select gateway certificate.

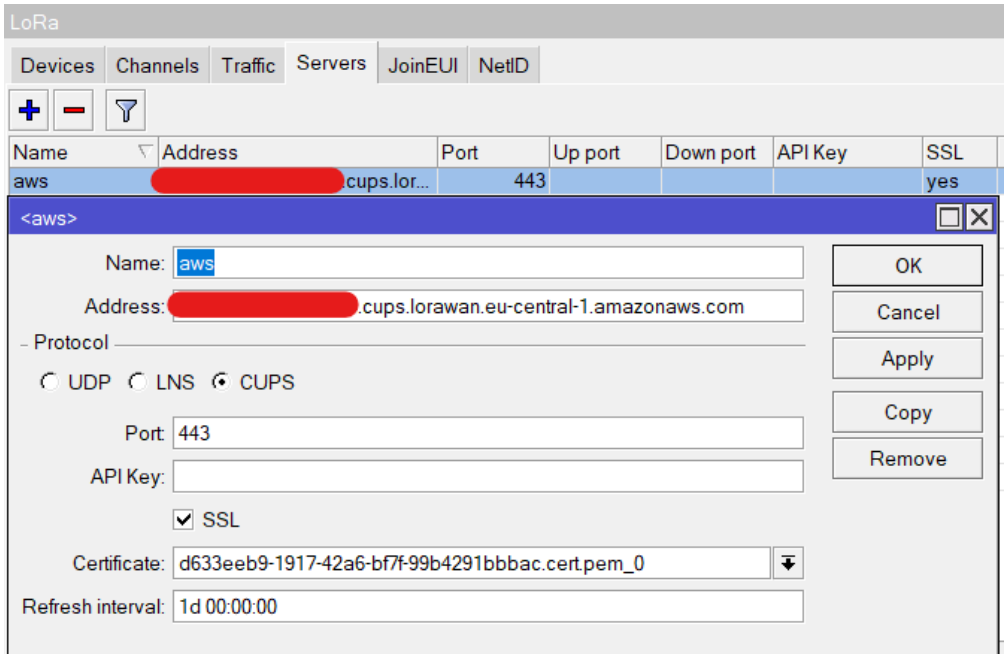
Make sure to apply newly configured server under **IoT>LoRa>Devices** tab:



And then, **enable** the LoRa interface.

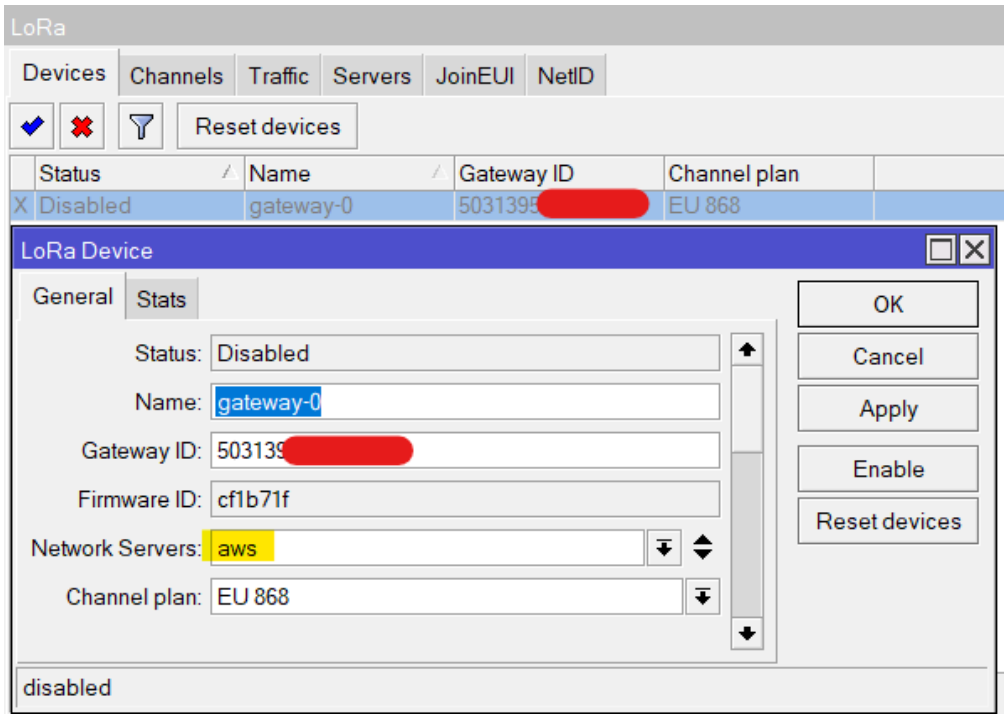
CUPS scenario

Navigate to the **IoT>LoRa>Servers** tab and add a new server:



- Name the server;
- Input CUPS endpoint address (without "`https://`" and "`:443`");
- Select CUPS protocol;
- Change port to "443";
- Enable SSL checkbox;
- Select gateway certificate.

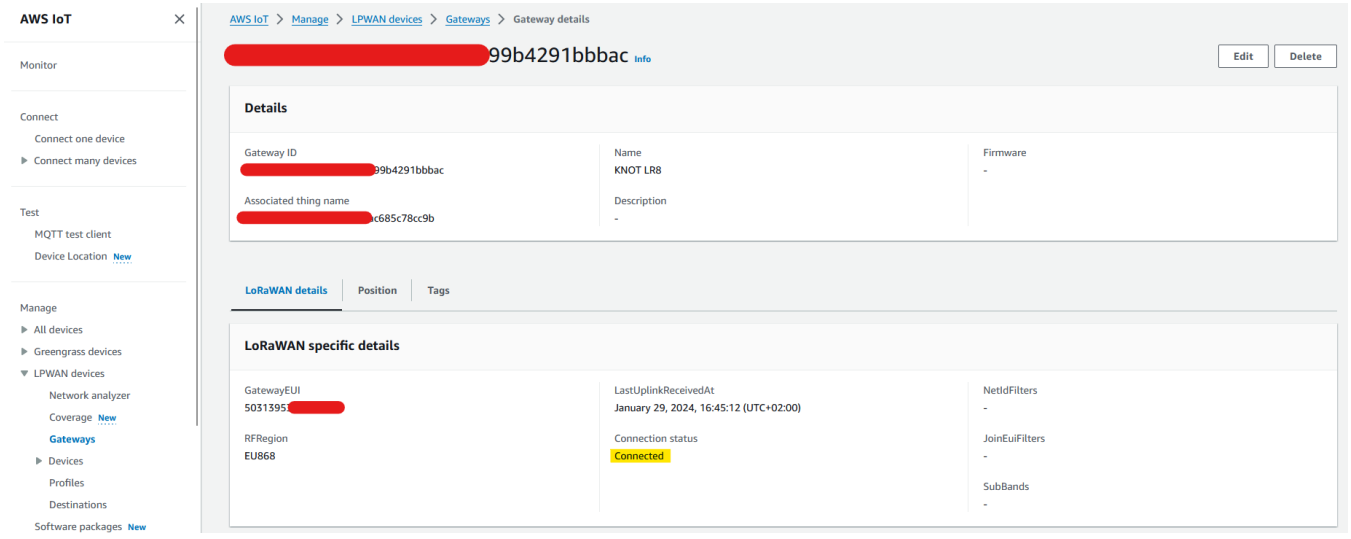
Make sure to apply newly configured server under **IoT>LoRa>Devices** tab:



And then, **enable** the LoRa interface.

Connection verification

If everything is configured correctly, you should see "connected" status on the AWS portal:



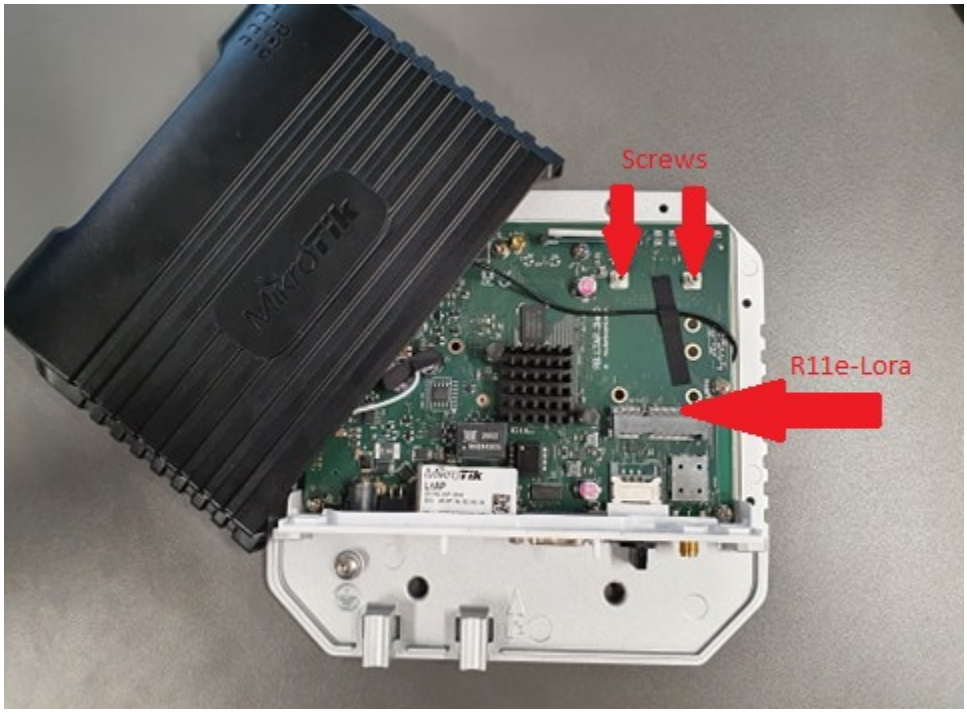
Step by step installation

LoRa card installation

LtAP LTE kit will be used as example in this section.

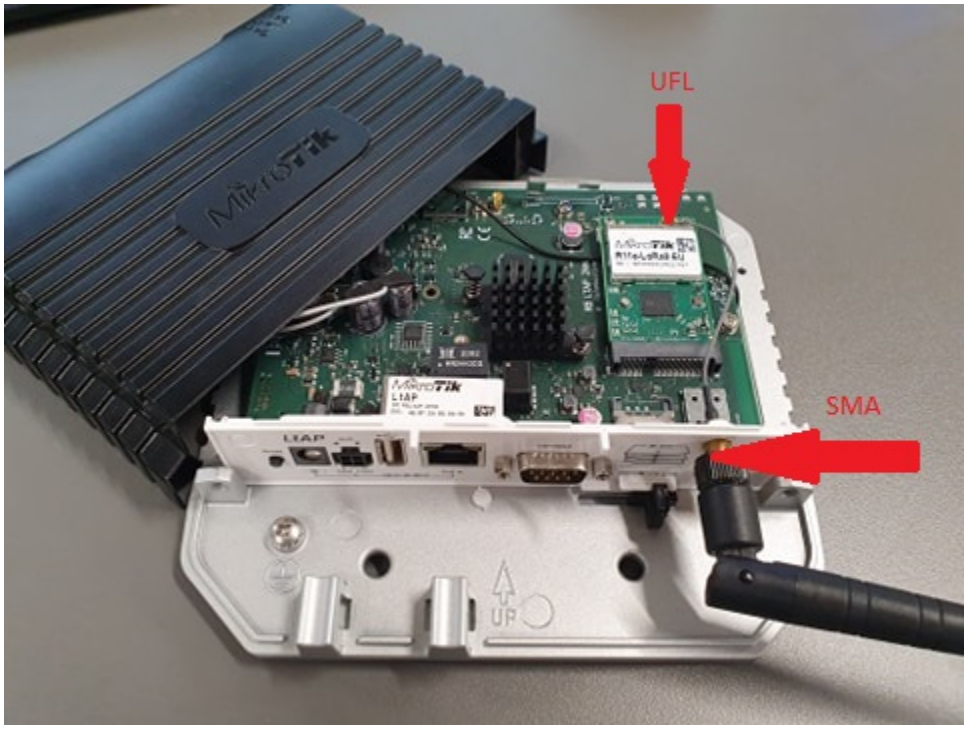
Open your routers case. Once you have removed all the screws carefully move the upper case to the left side, as the LTE antennas are attached to the inner side of it.

Insert R11e-LoRa card into the mini-PCIe slot and apply two screws to the threaded inserts.



Attach antenna to the card (UFL connector)

In this case UFL → SMA cable is also used, as the LtAP's case has a specific slot for it.



Once the previous steps are done, you can close the routers case and move on to configuration.

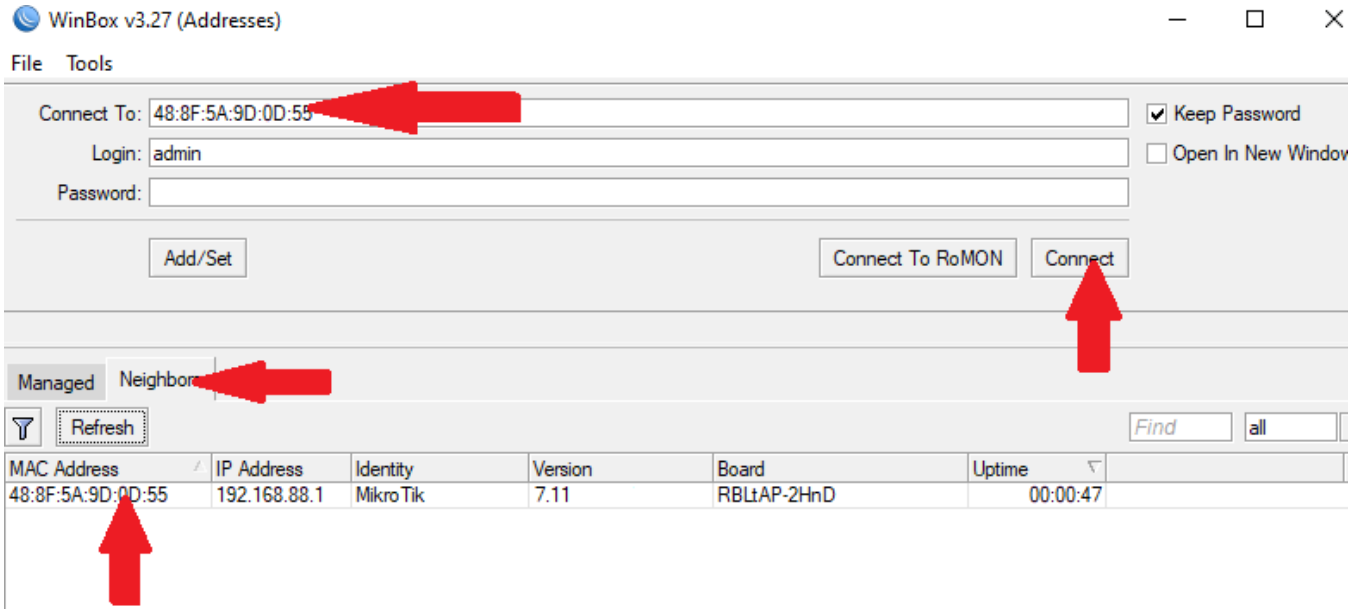
Configuration

GUI setup

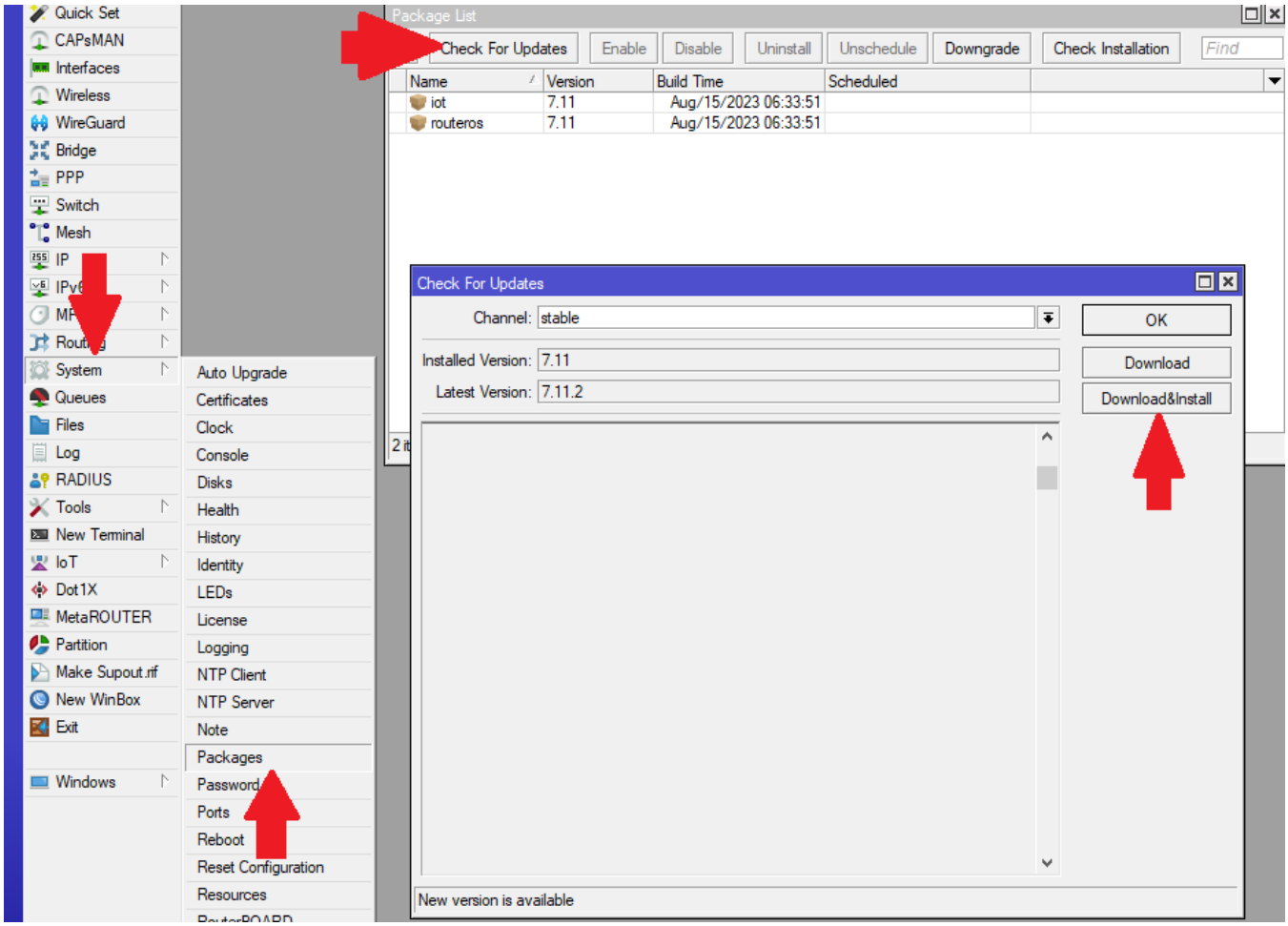
Connect to your router via Winbox or WebFig.

Winbox can be downloaded in the link given below:

https://mikrotik.com/download

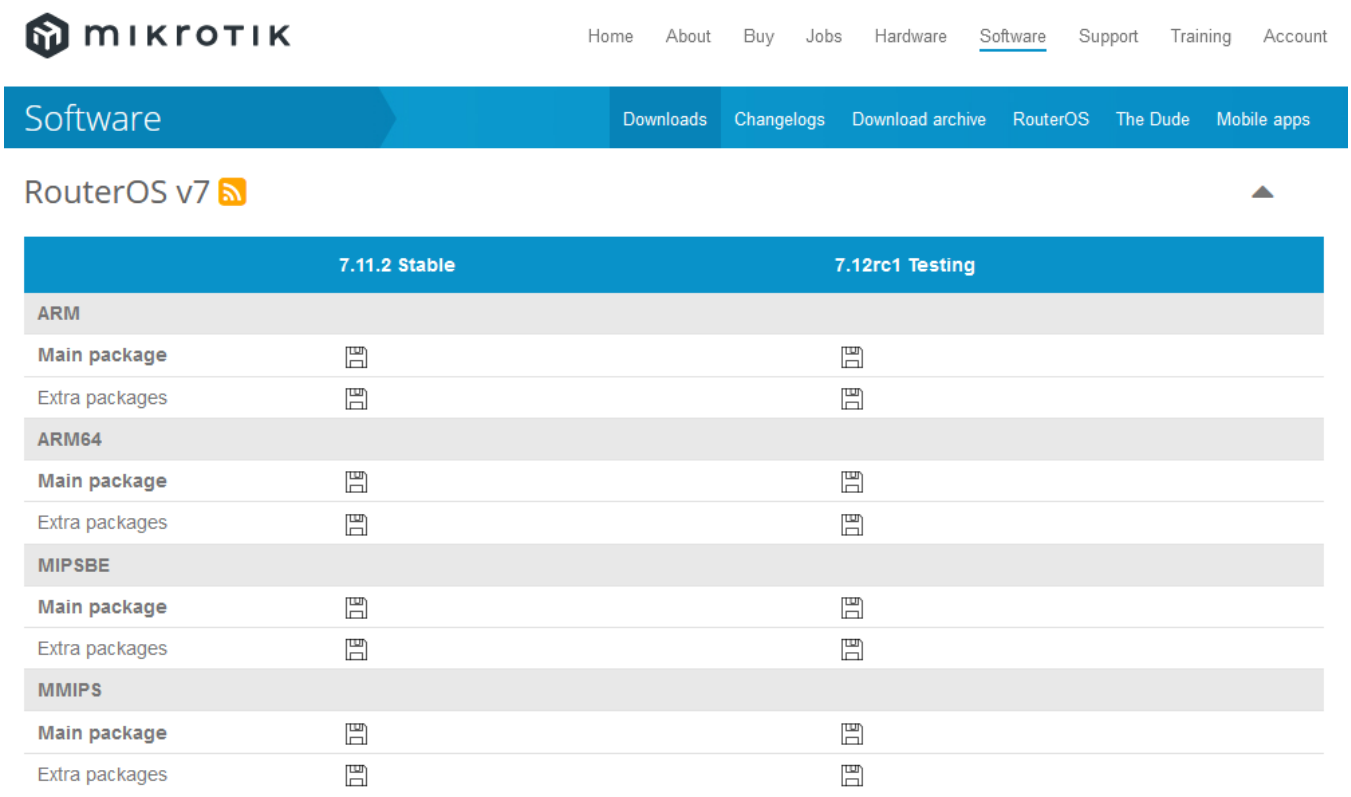


It is Highly recommended to upgrade your RouterOS version to the latest available. Installing the version will perform a reboot:

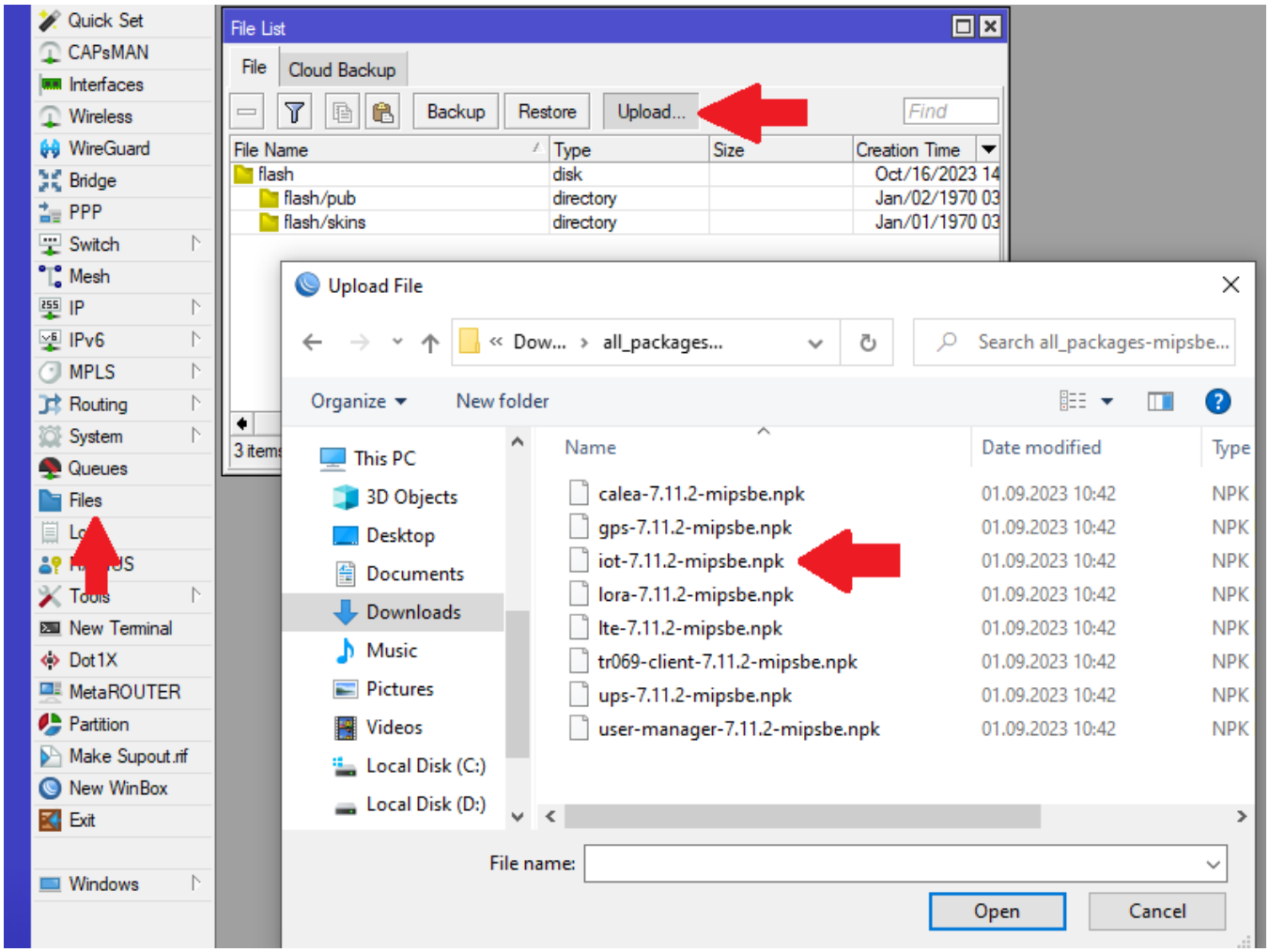


If your device does not have **IoT>LoRa** menu, download "**Extra packages**" specifically for your routers architecture and rOS version. You can see the type of your routers architecture at the top of Winbox window or in System → Resources → Architecture Name.

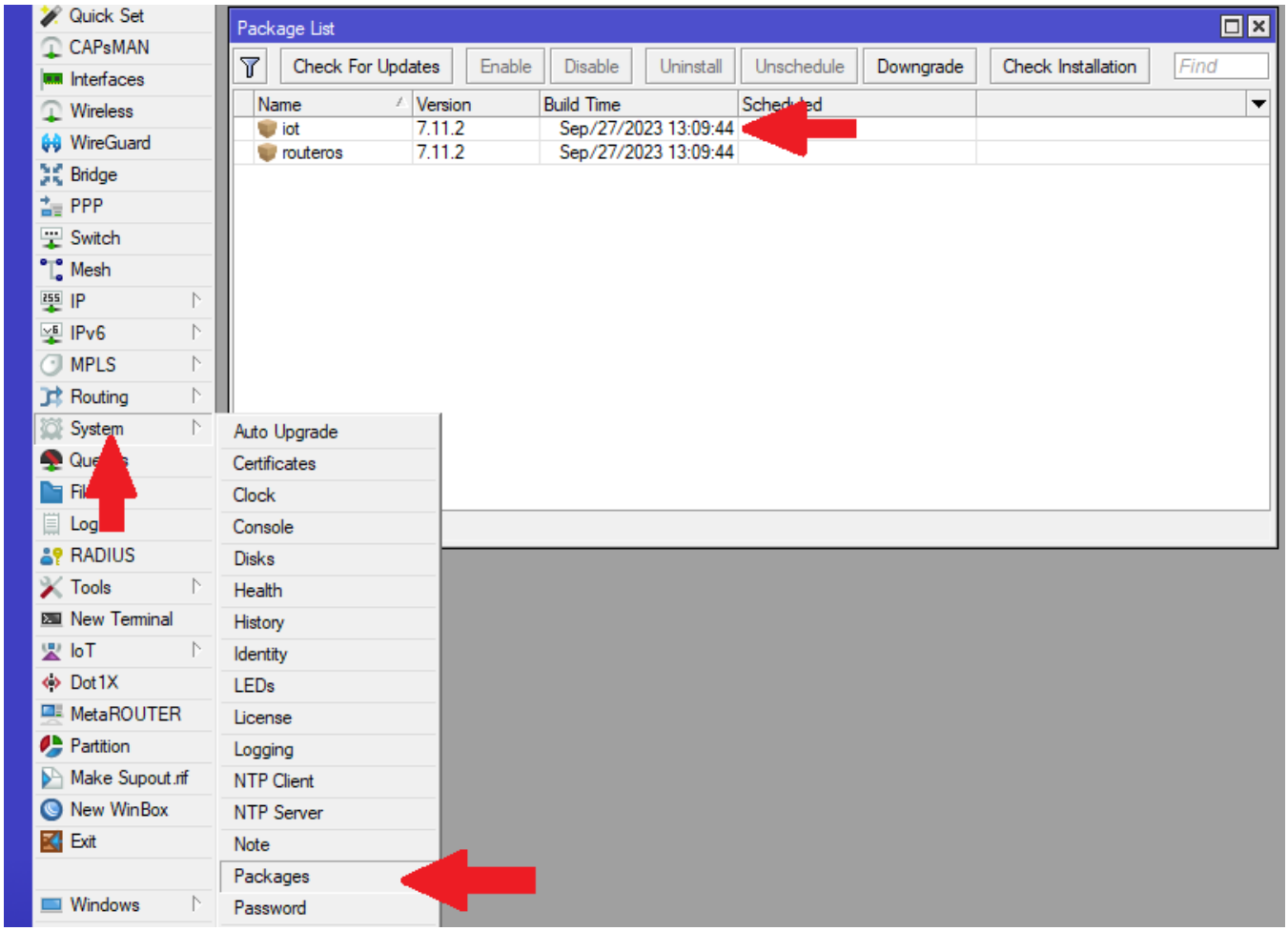
https://mikrotik.com/download



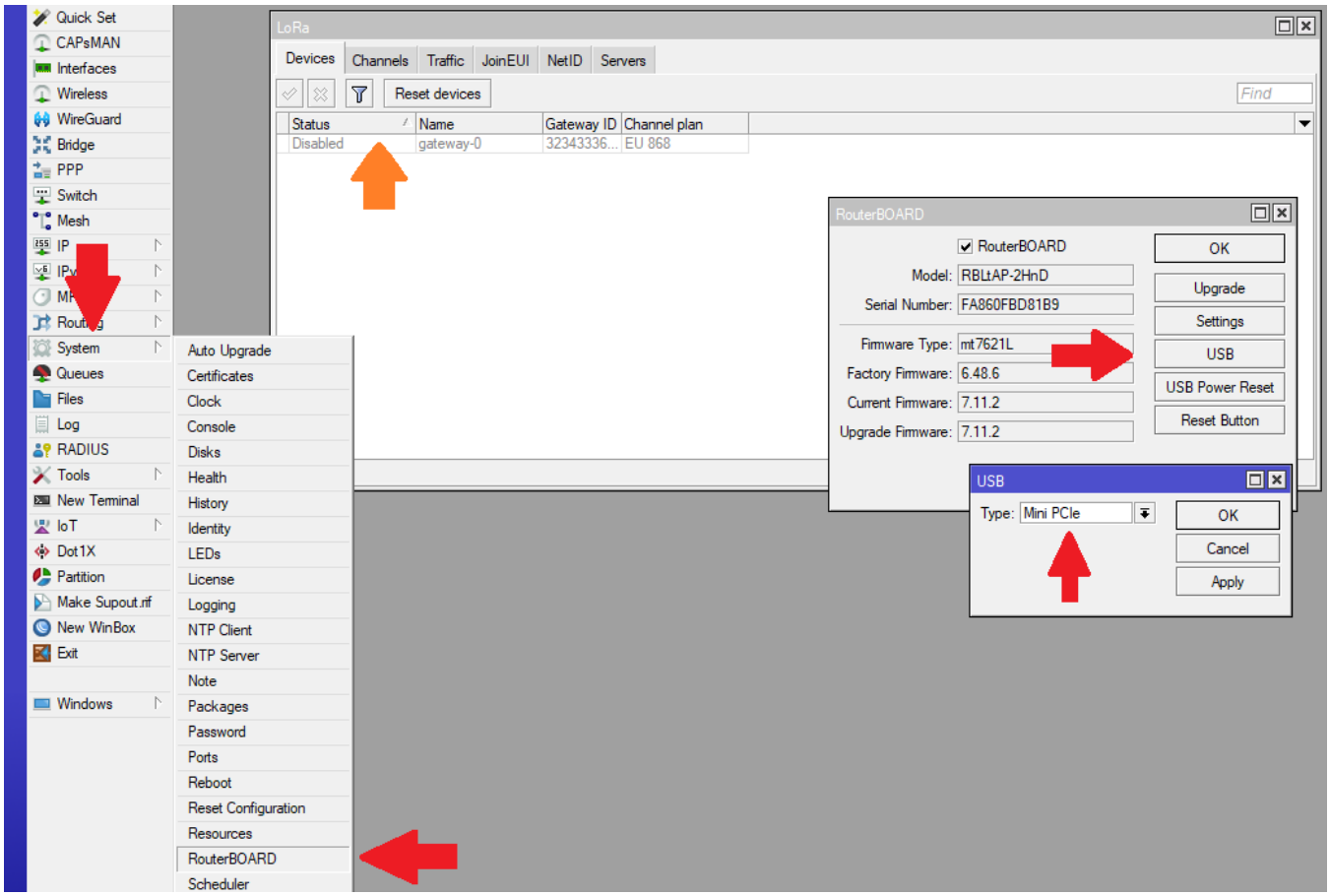
Once the package is downloaded and extracted, upload the **IoT** package to your router. It can be done via drag & drop as well. It should appear in the files folder after the upload is complete, reboot your router (System → Reboot) to install the package:



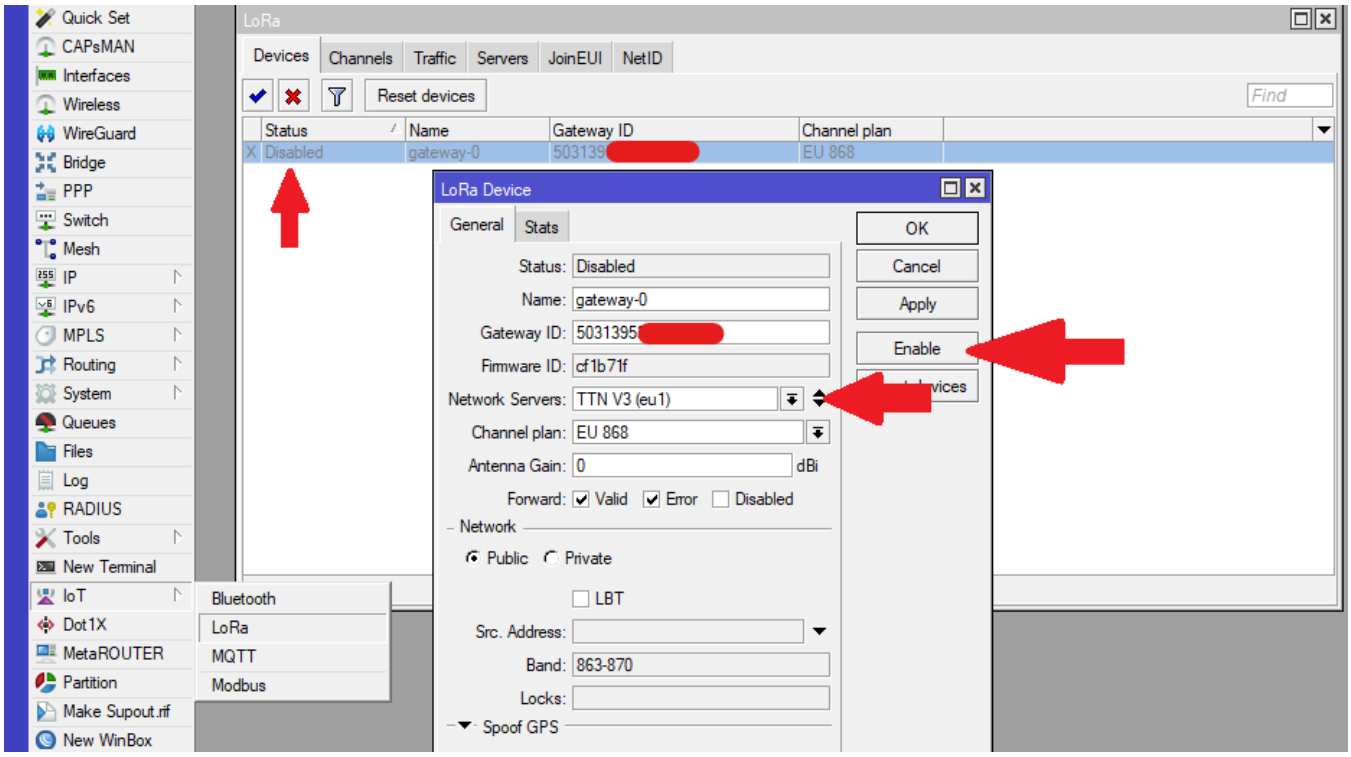
After the reboot, the package should be visible in the Package list:



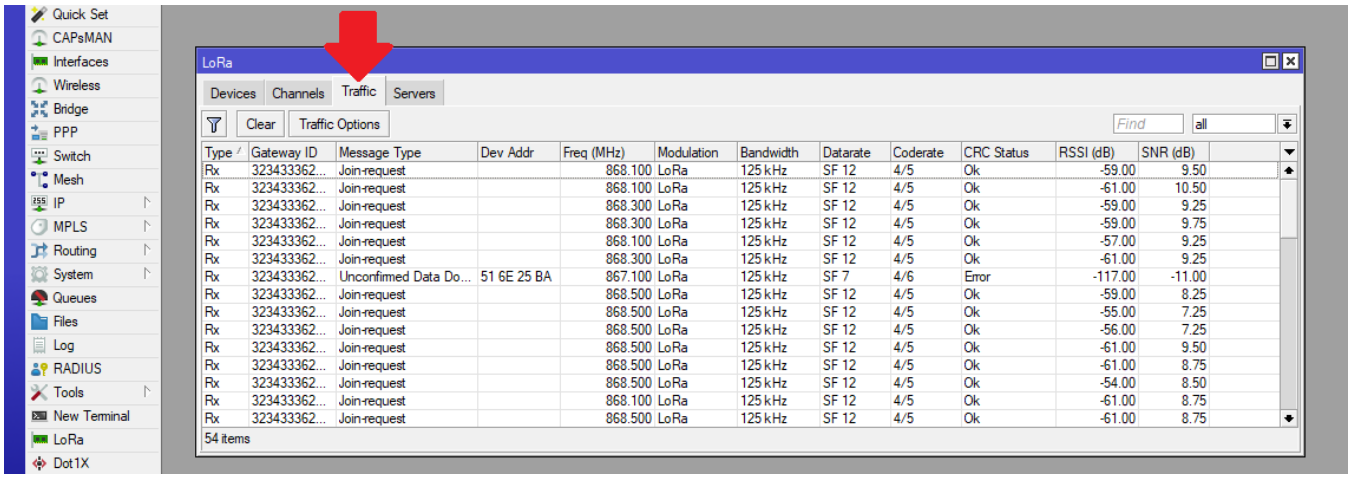
Check if the LoRa gateway has initialized under **IoT>LoRa>Devices**. If it is LtAP model, make sure to set USB Type to Mini-PCIe:



Once the gateway has shown up (under **IoT>LoRa>Devices**) select it, choose Network Servers from the default ones or add your own (under **IoT>LoRa>Servers**) and enable it:



Navigate to Traffic tab to monitor the surrounding nodes sending requests:

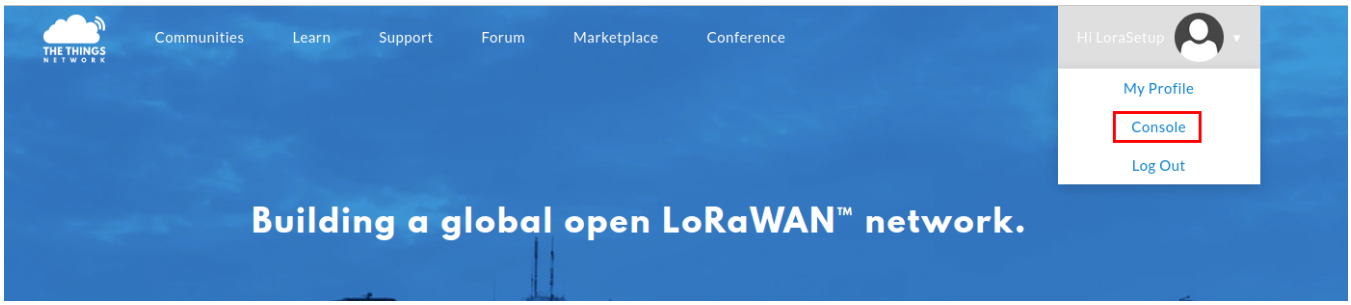


This concludes basic installation and configuration of LoRa mini-PCIe cards. For additional settings check: General Properties

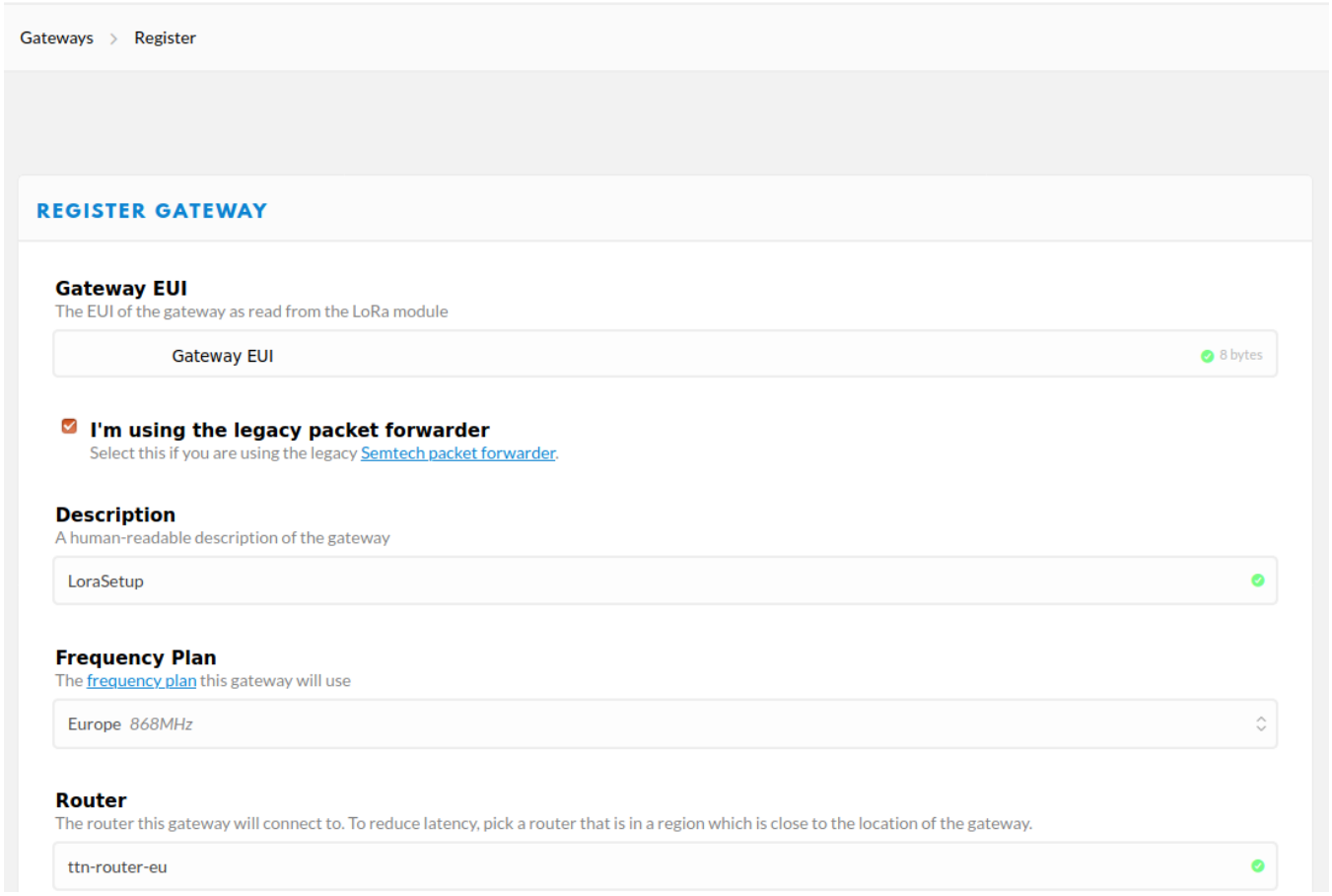
The Things Network

Once you have installed the lora package on your router and created an account on `The Things Network` you can set up a running gateway

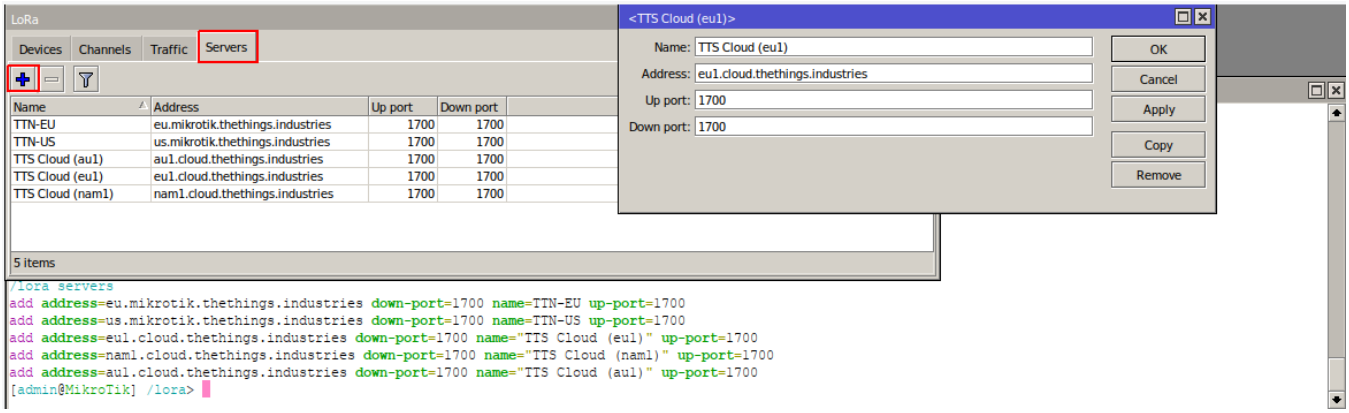
- Login into your account and go to Console and select Gateways



- Select *register gateway* and fill in the blank spaces. Gateway EUI can be found in your lora interface

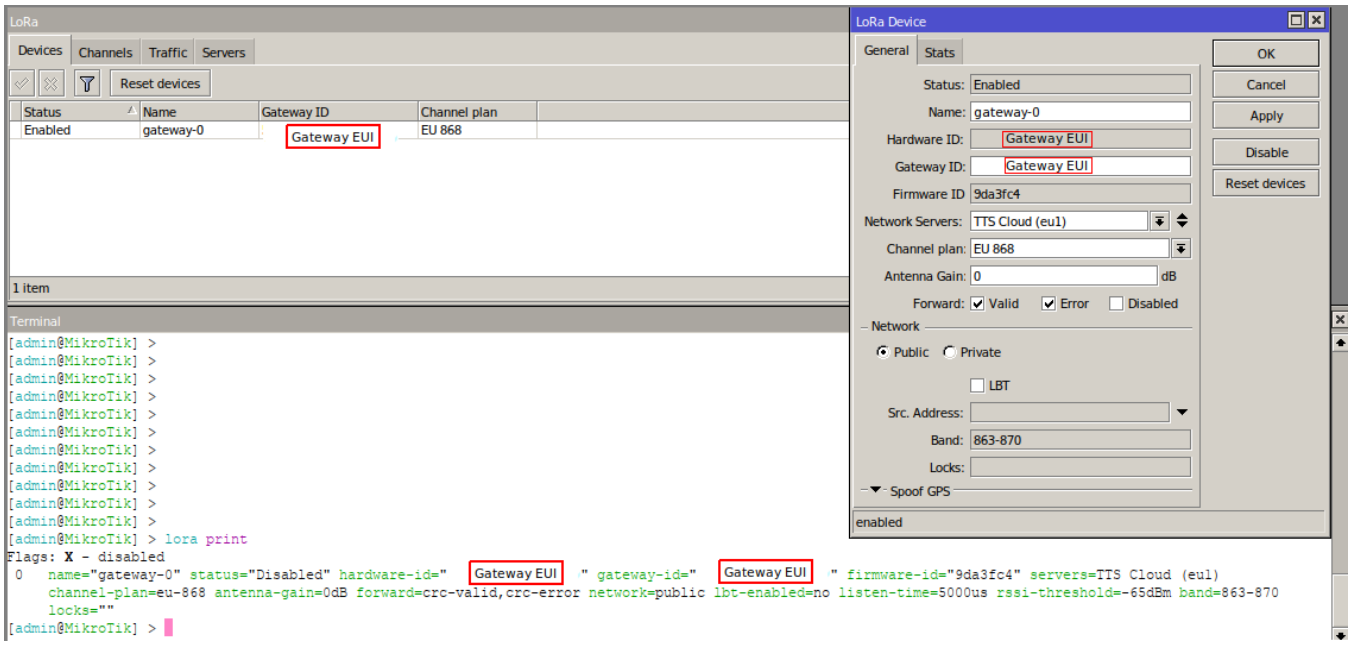


- You will have to manually add the Network Servers, or you can upgrade your router to the stable version **6.48.2** and these servers will be added automatically (highly recommended)
  https://wiki.mikrotik.com/wiki/Manual:Upgrading_RouterOS

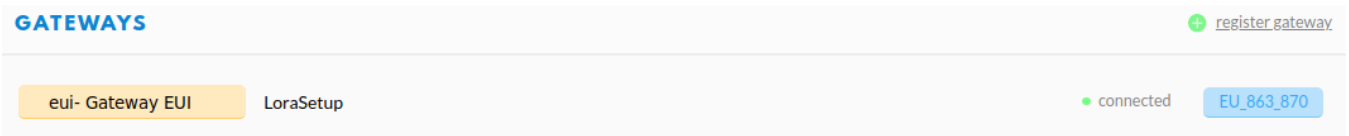


/lora servers

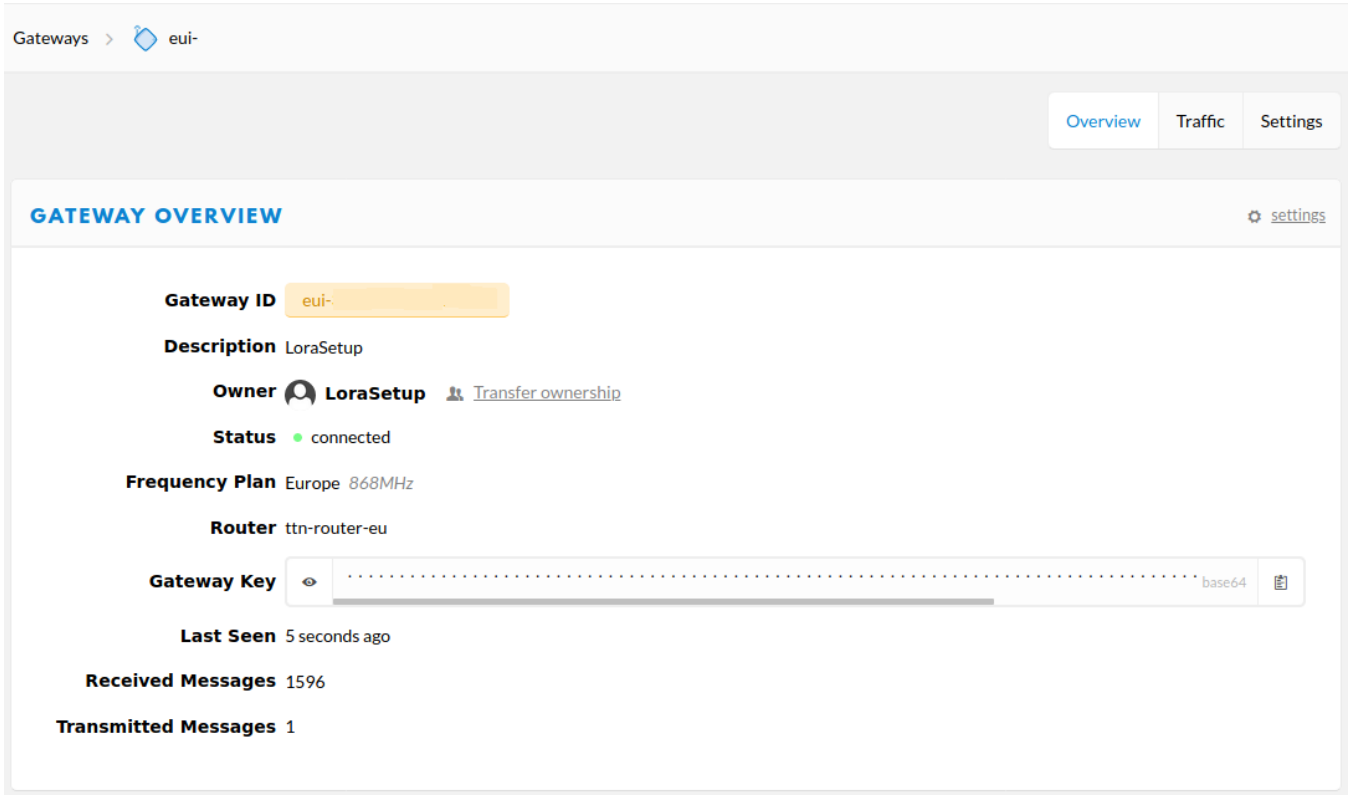
add address=eu1.cloud.thethings.industries down-port=1700 name="TTS Cloud (eu1)" up-port=1700
add address=nam1.cloud.thethings.industries down-port=1700 name="TTS Cloud (nam1)" up-port=1700
add address=au1.cloud.thethings.industries down-port=1700 name="TTS Cloud (au1)" up-port=1700

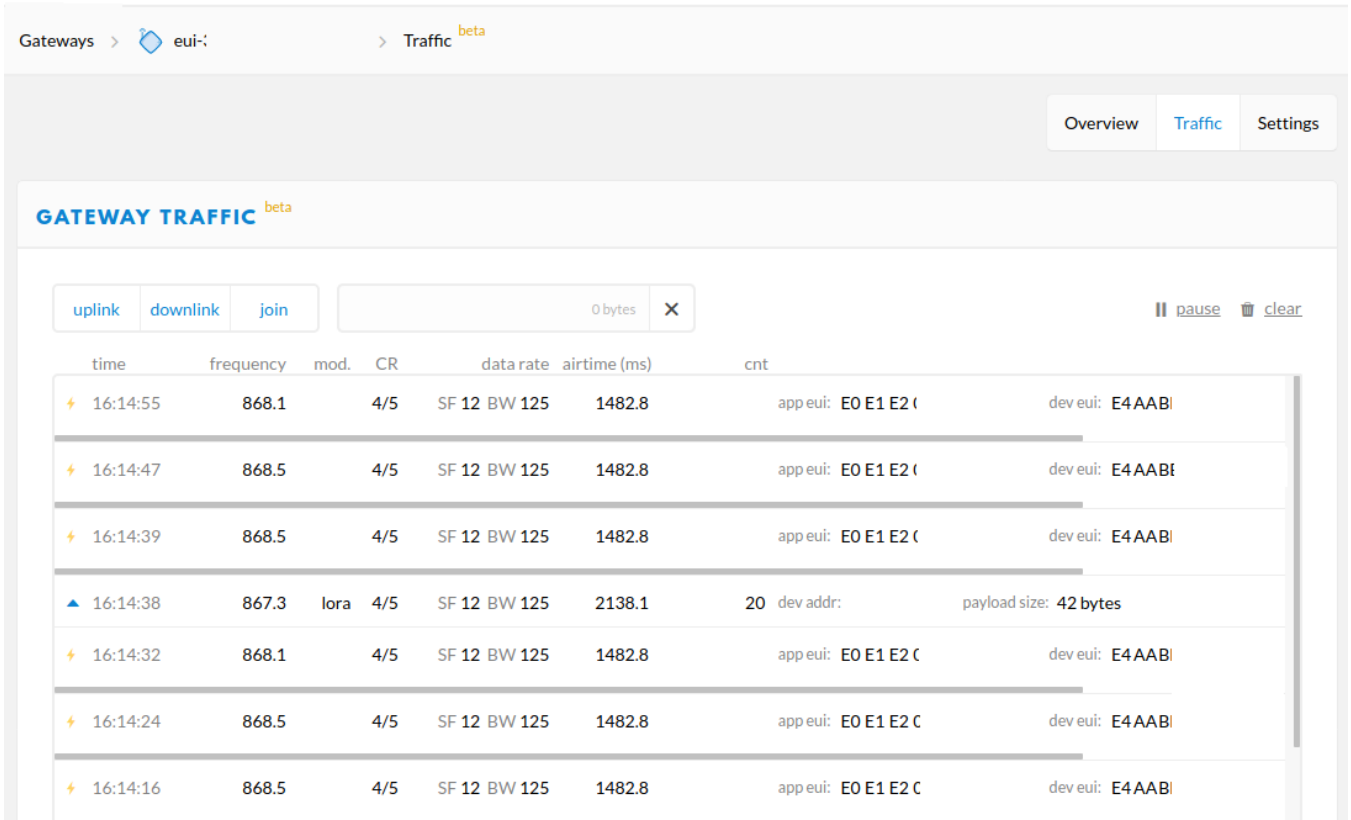


- After everything is filled press Register Gateway at the bottom of the page. If you have set everything accordingly to the previous steps you should see that your lora gateway is now connected



- At this point everything is set and you have a working lora gateway. You can monitor incoming packets in Traffic section



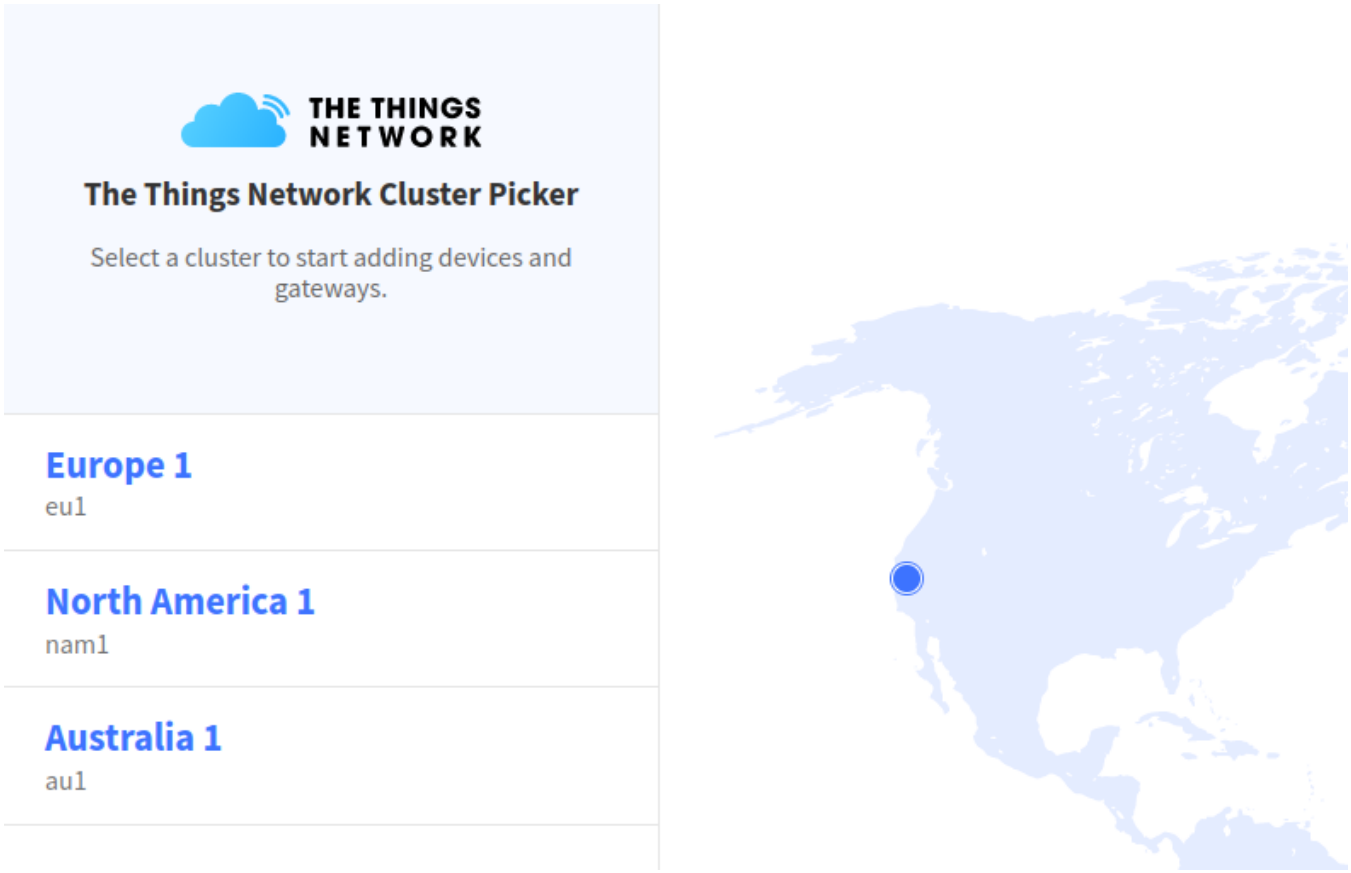


*Later this year, The Things Network will be migrating to a new version of network server, called The Things Stack.

The Things Stack

Selecting a region

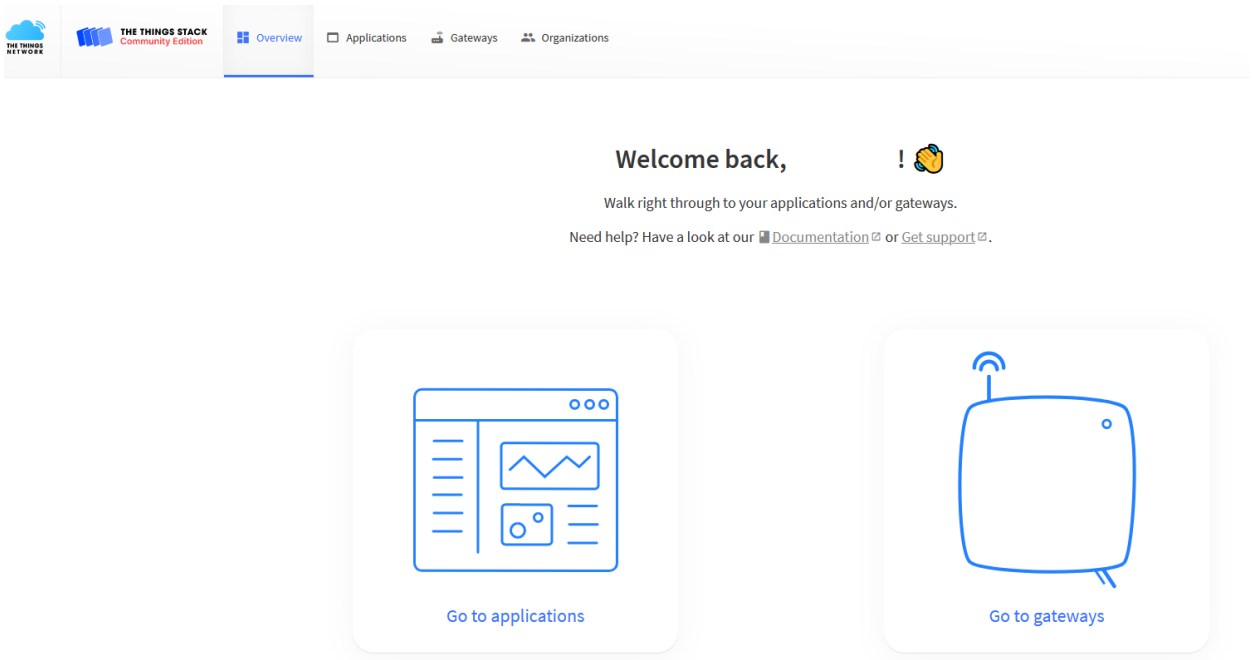
The Things Stack is a new version of The Things network.



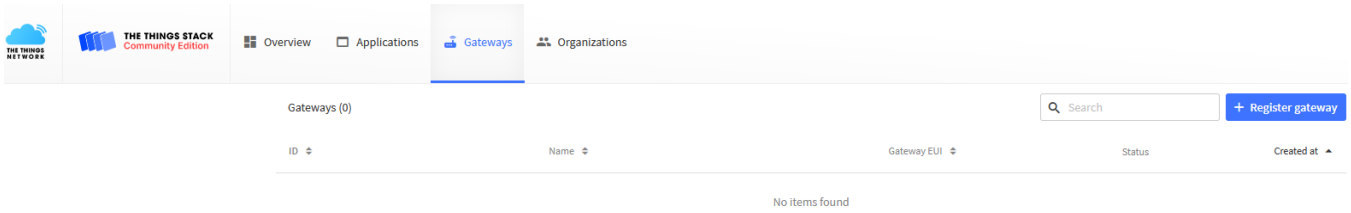
Choose your region and login with The Things network account or other credentials.

Registering gateway

Once logged in, navigate to "Go to gateways":

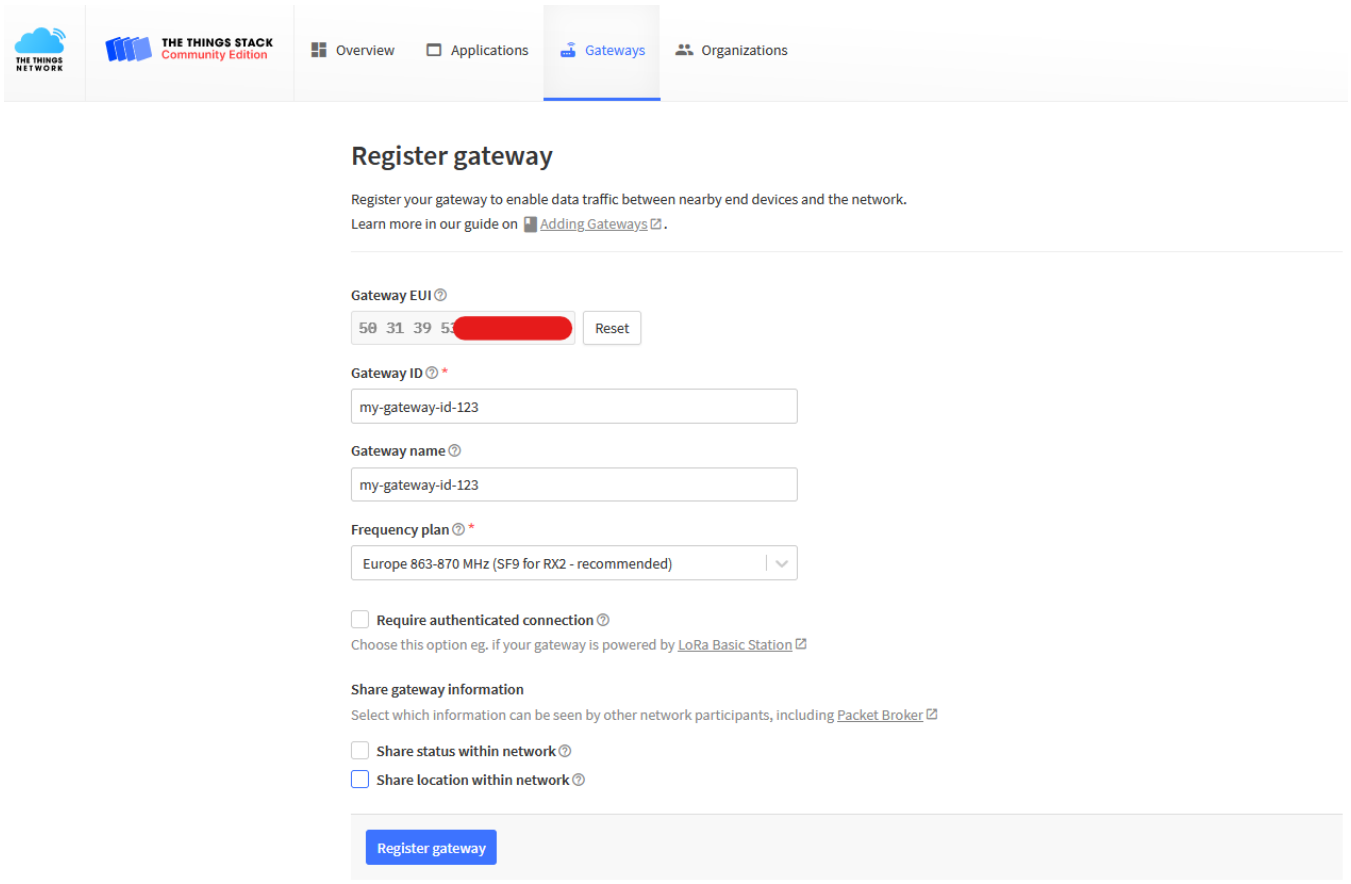


Register a gateway by clicking on the "**+ Register gateway**" button:

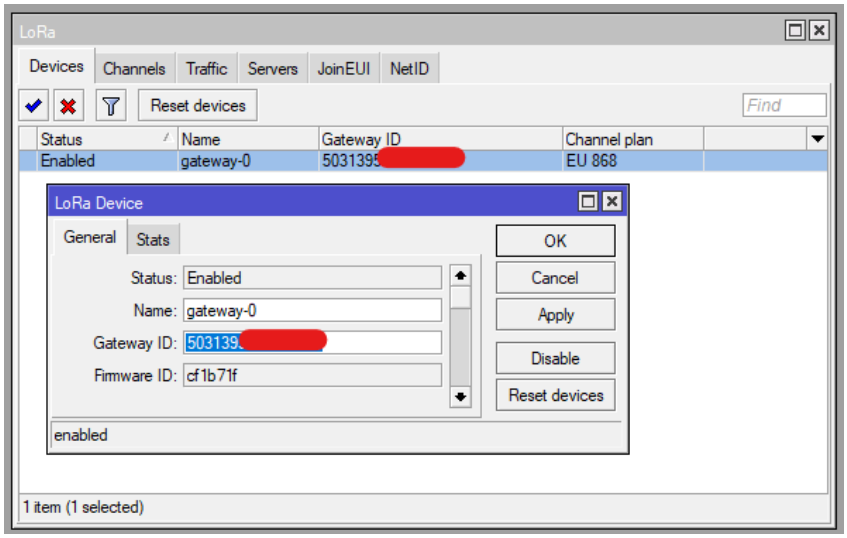


UDP protocol gateway registration

Fill in the blank spaces. Input **Gateway EUI**. Make sure to select a correct frequency plan. Do not enable "**Require authenticated connection**" option (it is used for LNS and CUPS)!



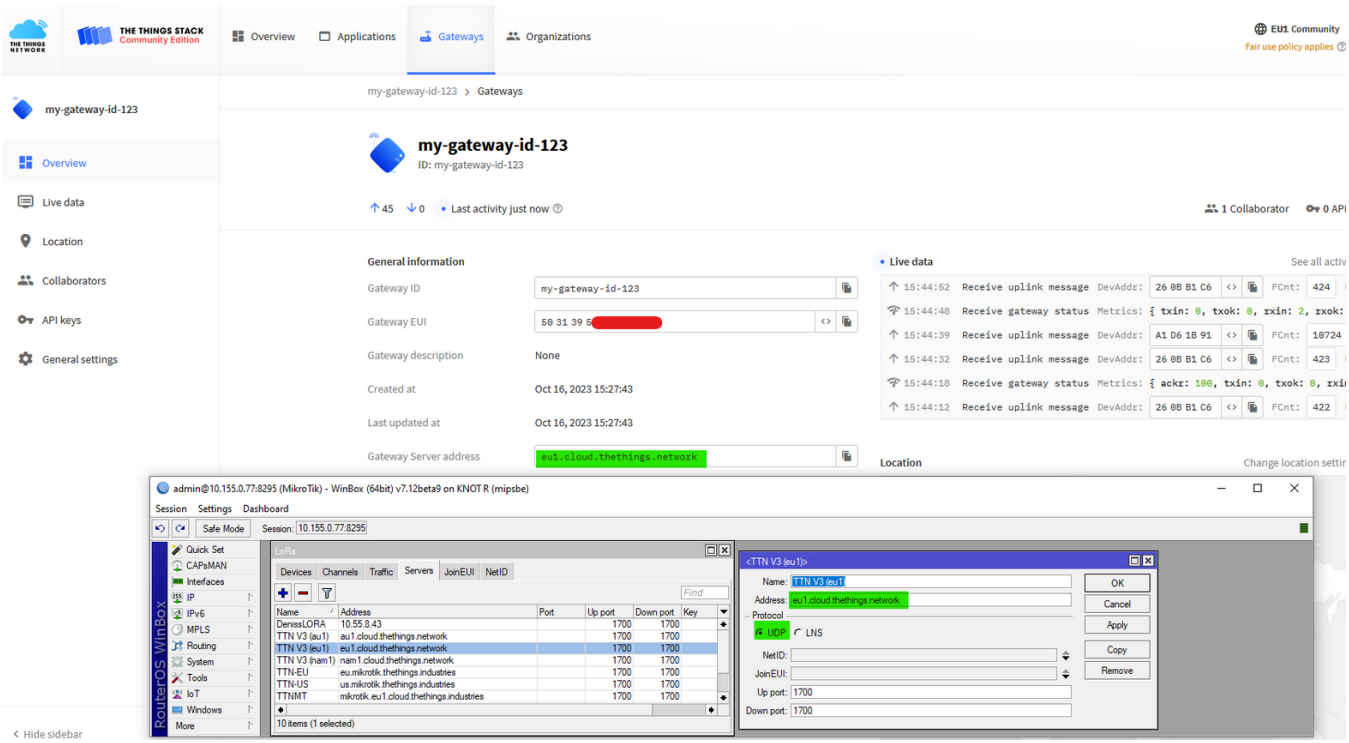
In RouterOS, **Gateway EUI** value can be found under "**IoT>LoRa>Devices>Gateway ID**"):



For additional information check their documentation page.

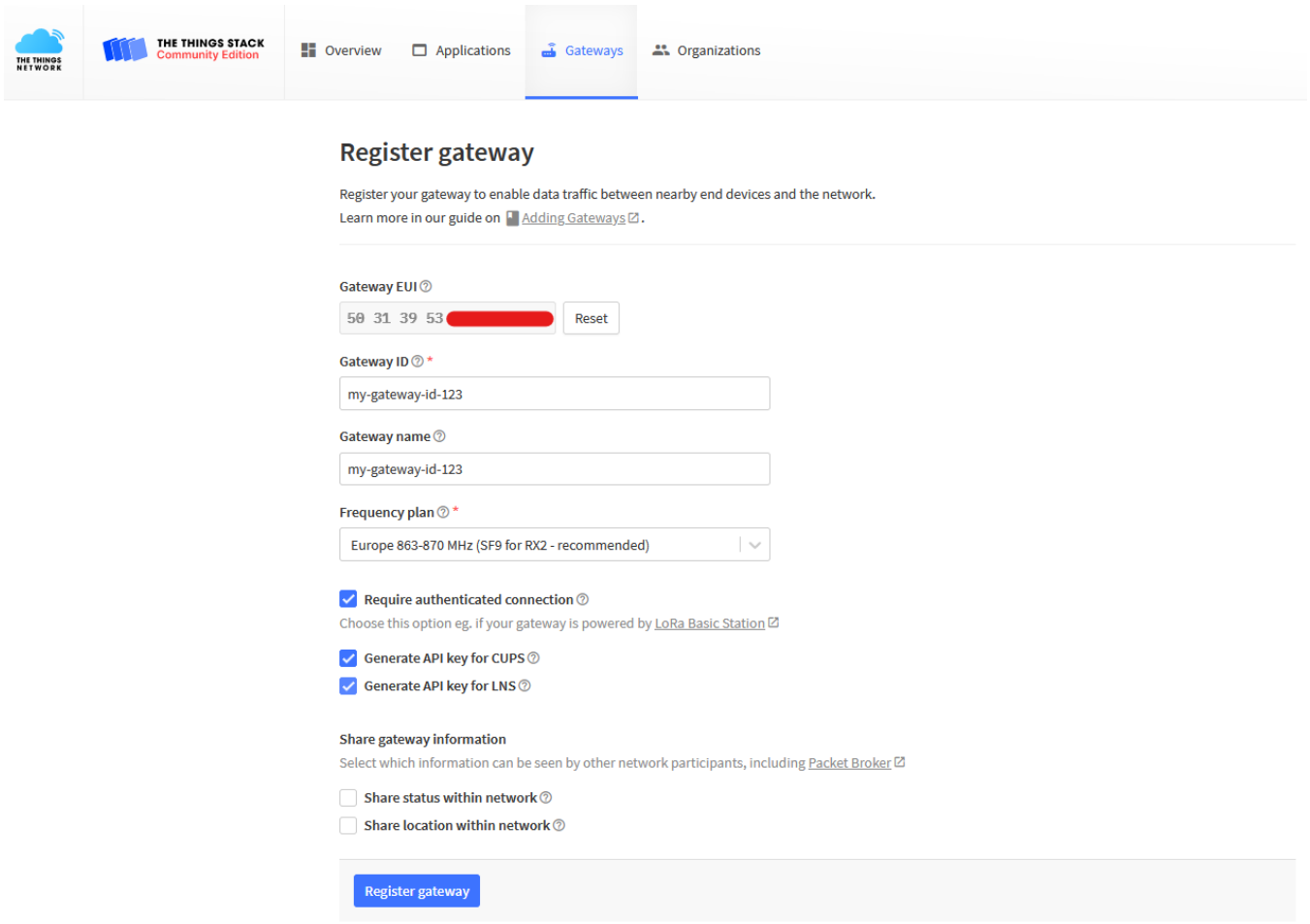
UDP scenario RouterOS settings

Double-check that the correct TTN server is selected by the LoRa device (in RouterOS) and that the server setting uses "UDP" protocol:

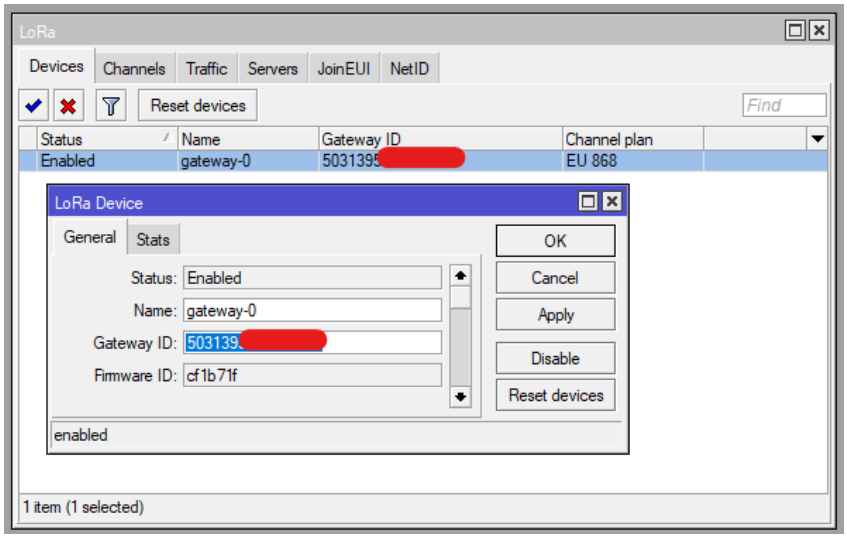


LNS and CUPS protocol gateway registration

Fill in the blank spaces. Input **Gateway EUI**. Make sure to select a correct frequency plan. Enable "**Require authenticated connection**" and the follow-up (for LNS) "**Generate API key for LNS**" and (for CUPS) "**Generate API key for CUPS**" options:

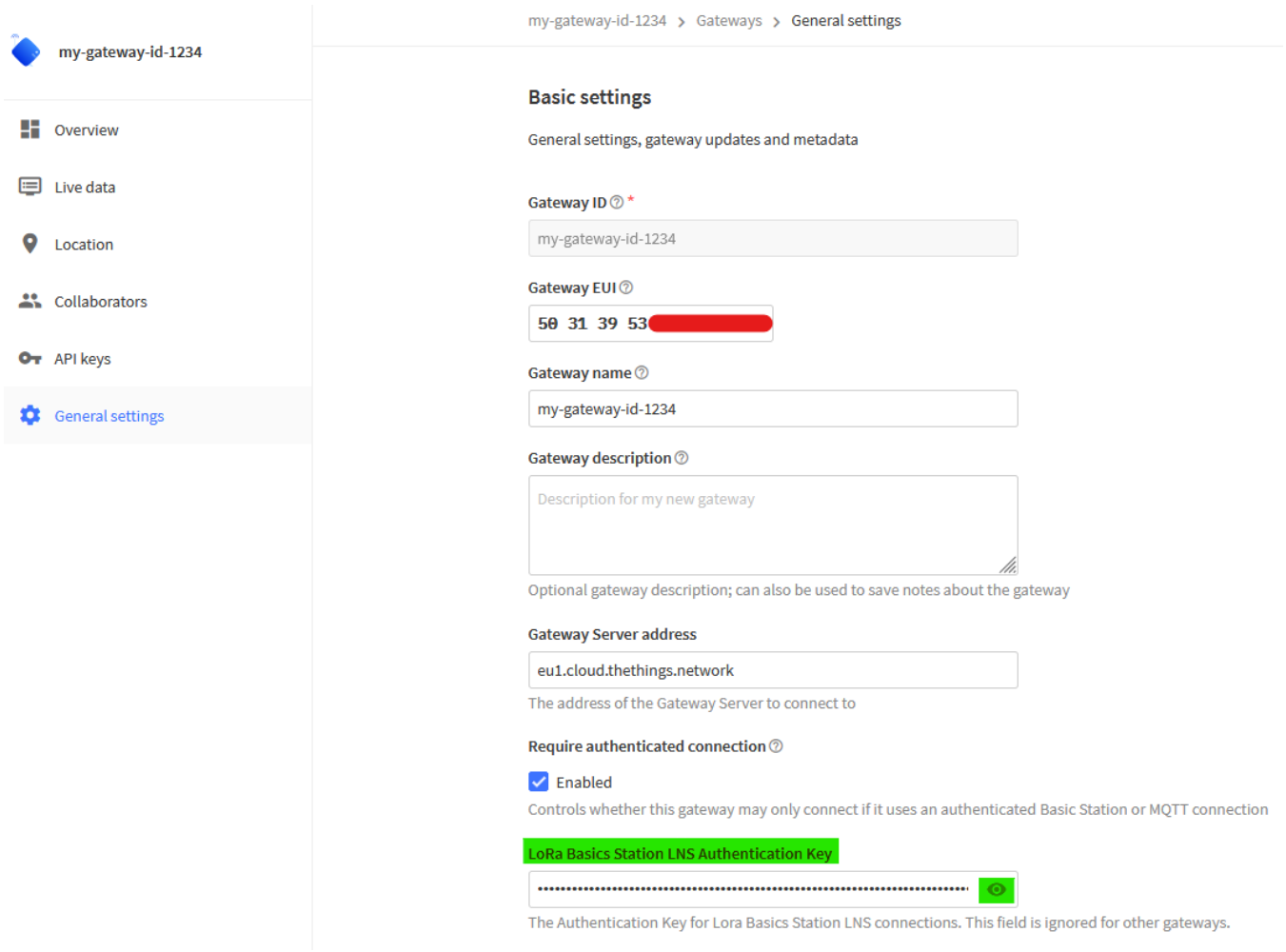


In RouterOS, **Gateway EUI** value can be found under "**IoT>LoRa>Devices>Gateway ID**":



After clicking on "**Register gateway**" button, you should be prompted to download the keys. Download them.

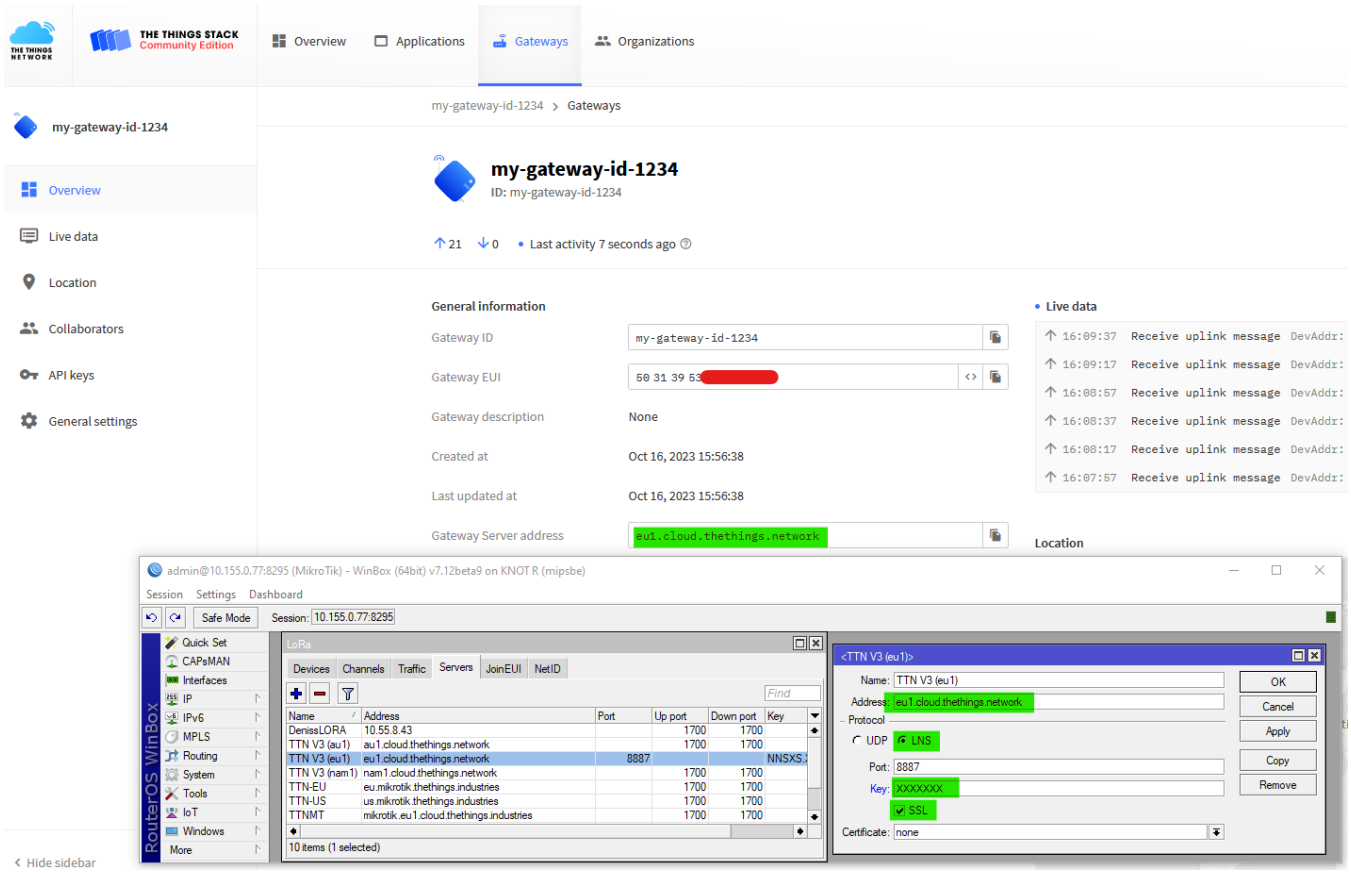
To view LNS and CUPS keys, inspect download files. LNS key should also be visible under the "**LoRa Basics Station LNS authentication Key**" field:



For additional information check their documentation page.

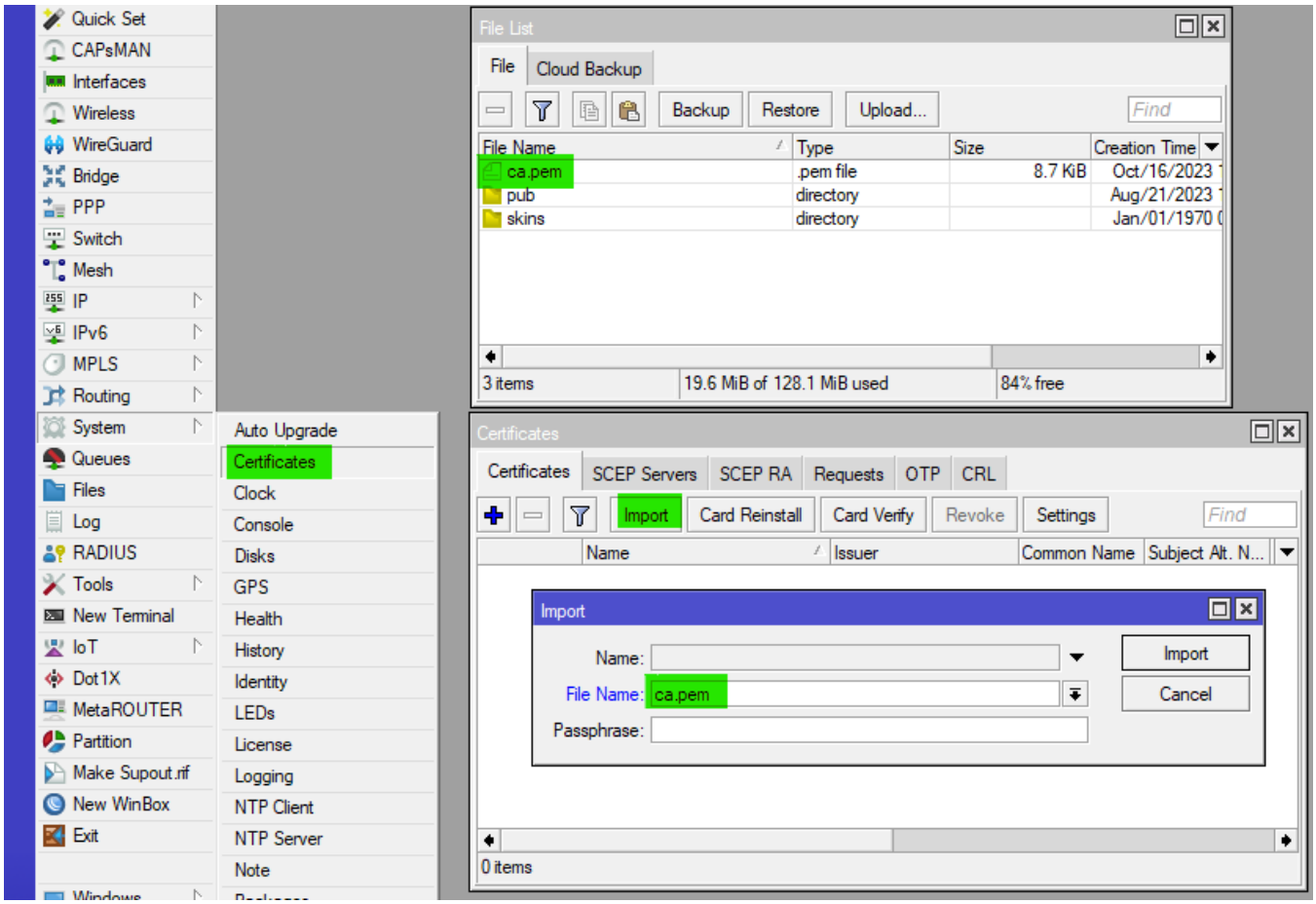
LNS scenario RouterOS settings

Make sure that the correct TTN server is selected, that the correct port is configured (TTN expects LNS over 8887), that LNS protocol is chosen, that the LNS key (from the "**LoRa Basics Station LNS authentication Key**" field) is input and that "**SSL**" checkbox is enabled:

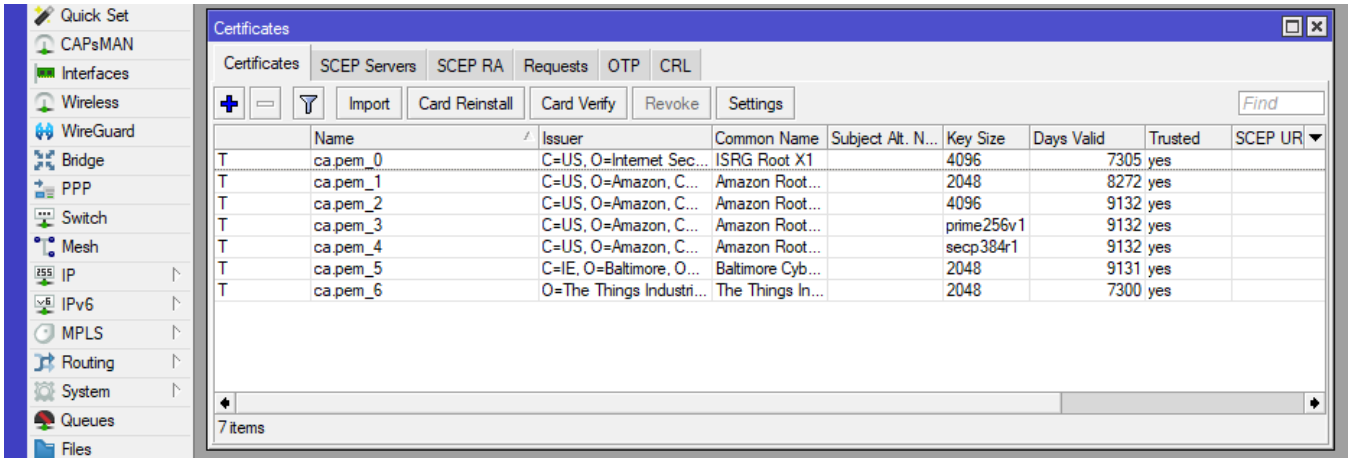


The last step is to download and import Root Certificates. The page has links to the required file.

After the certificate file was downloaded, drag and drop it into the RouterOS file menu and import the certificate list:

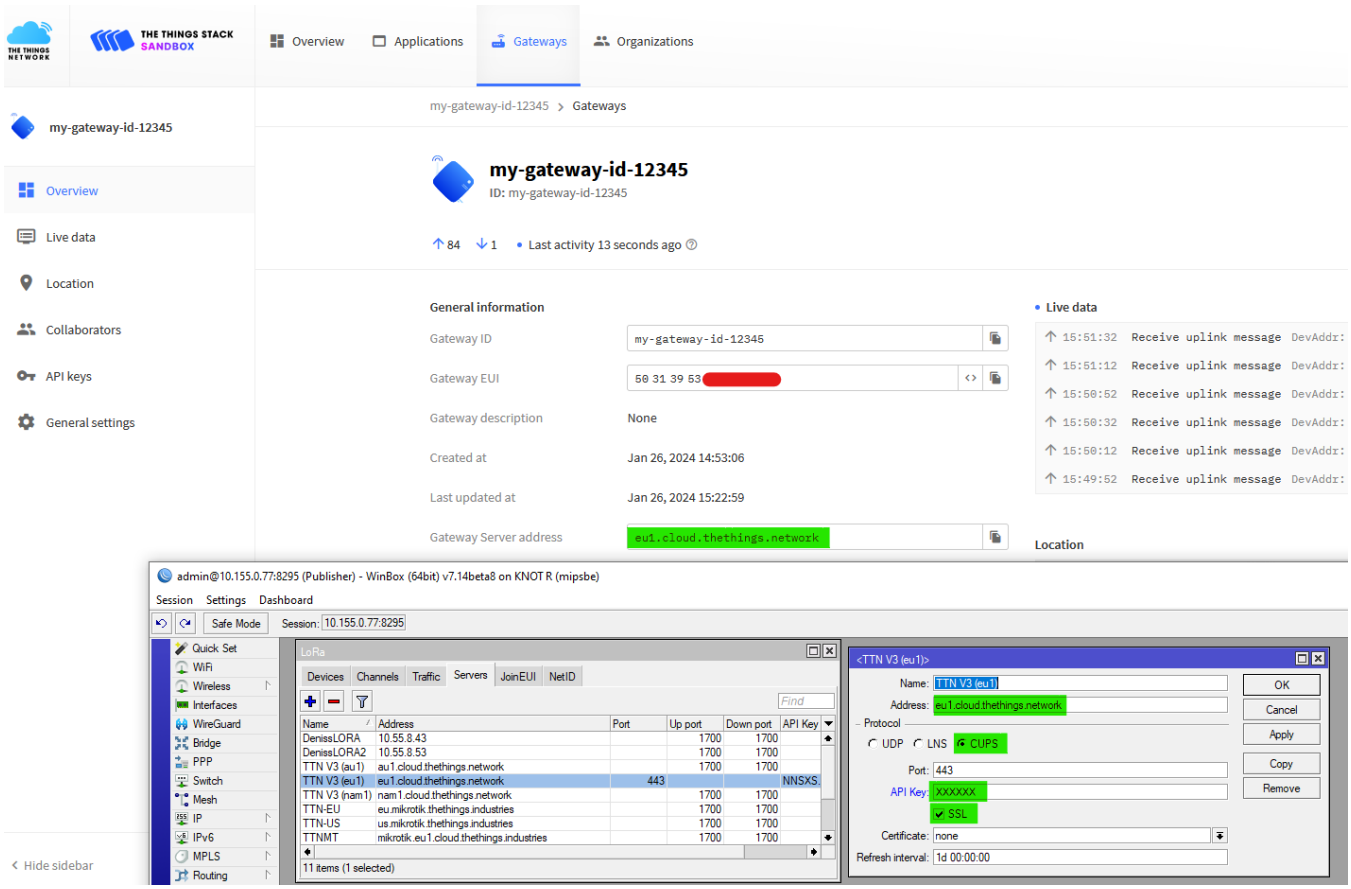


This should make the certificate list trusted:



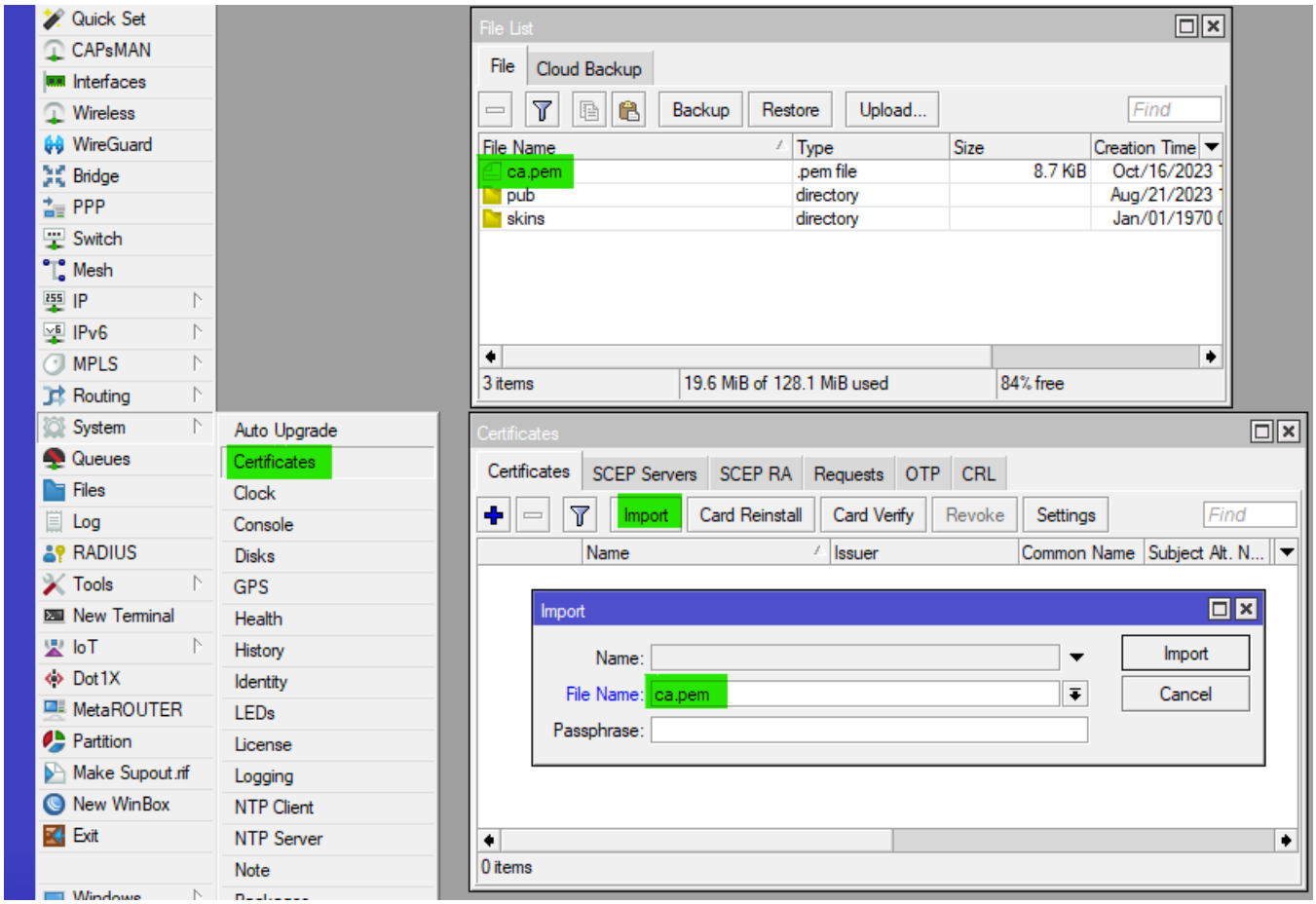
CUPS scenario RouterOS settings

Make sure that the correct TTN server is selected, that the correct port is configured (TTN expects CUPS over 443), that CUPS protocol is chosen, that the CUPS key is input and that "**SSL**" checkbox is enabled:

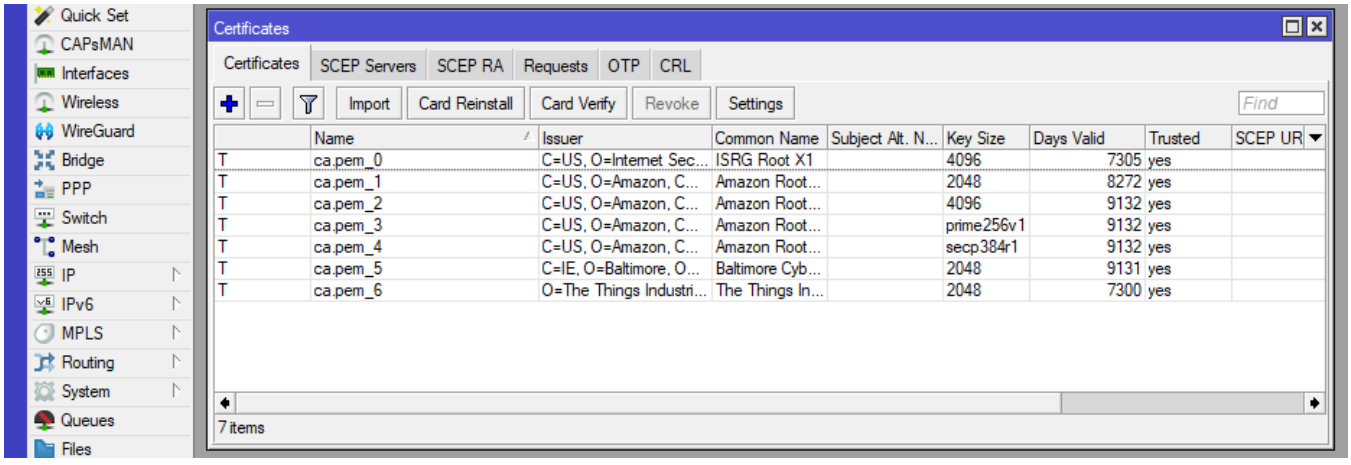


The last step is to download and import Root Certificates. The page has links to the required file.

After the certificate file was downloaded, drag and drop it into the RouterOS file menu and import the certificate list:

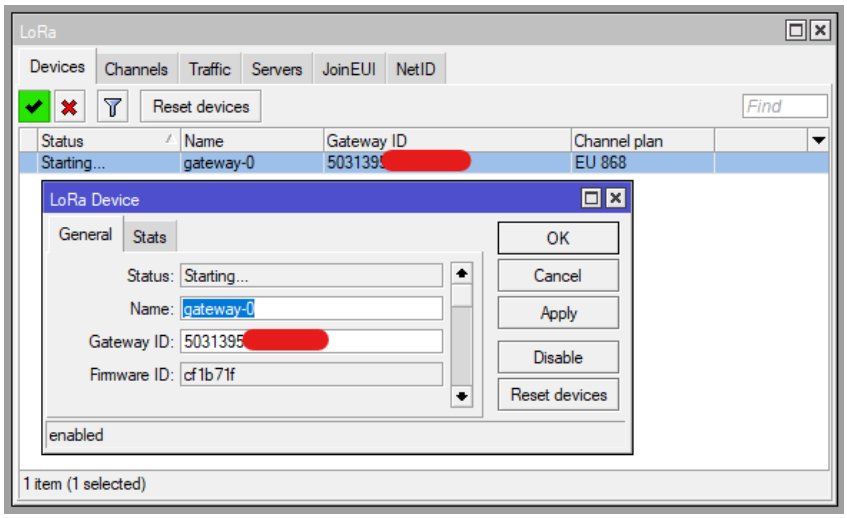


This should make the certificate list trusted:

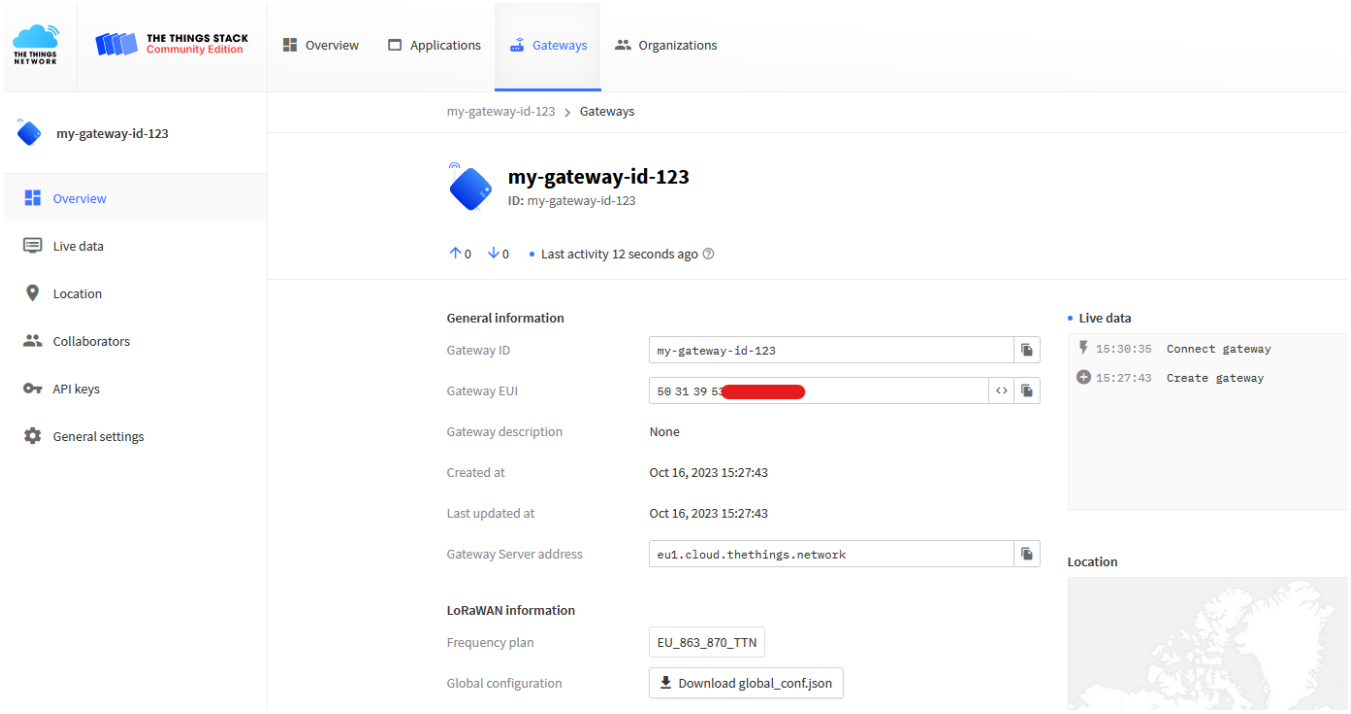


Verification

If everything is configured in correctly, right after you enable the LoRa interface in RouterOS ("**IoT>LoRa>Devices>Enable**"):



You should see the gateway connection "Live data" update:



MQTT

Summary

MQTT is an open OASIS and ISO standard lightweight, publish-subscribe network protocol that transports messages between devices. A typical MQTT communication topology consists of:

- an MQTT publisher → a device that sends information to the server;
- an MQTT broker → a server where the data is stored;
- an MQTT subscriber → a device that reads/monitors the data published on the server.

RouterOS can act as an MQTT publisher and subscriber (starting with **7.11beta2**). You can also run an MQTT broker/server via the container feature. For Mosquitto MQTT broker configuration visit the link here.

You can find application examples for MQTT publish scenarios below:

a) MQTT/HTTPS example with AWS cloud platform

b) MQTT example with Azure cloud platform

c) MQTT and ThingsBoard configuration

Please note that AWS and Azure examples (scripts) showcase publishing Bluetooth tag data. Currently, only the KNOT has a Bluetooth chip built-in.

Configuration

Sub-menu: `/iot mqtt`

note: **iot** package is required.

IoT package is available with RouterOS version 6.48.3. You can get it from our download page - under "Extra packages".

Property	Description
brokers	A list of configured MQTT brokers.
connect	A command that specifies, which broker to connect to.
disconnect	A command that specifies, which broker to disconnect from.
publish	A command that defines the MQTT message that needs to be published.
subscribe	A command that defines MQTT topics to subscribe to.
subscriptions	A list of subscribed topics and received messages.
unsubscribe	A command that specifies, which topic to unsubscribe from.

Brokers

To add a new MQTT broker (or an MQTT server), run the following command:

```
/iot mqtt brokers add
```

Configurable properties are shown below:

Property	Description
address (*IP|hostname*; Default:)	IP address or hostname of the broker.
auto-connect (*yes | no*; Default: **no**)	When enabled, after the connection with the MQTT broker goes down/gets interrupted, RouterOS will try to re-establish the connection over and over again.
certificate (*string*; Default:)	The certificate that is going to be used for the SSL connection.
client-id (*string*; Default:)	A unique ID used for the connection. The broker uses this ID to identify the client.
keep-alive (*integer:30..64800*; Default: **60**)	A parameter that defines the time (in seconds), after which the client should "ping" the MQTT broker that it is "alive", to ensure the connection stays ongoing. This value should be set according to MQTT broker settings.
name (*string*; Default:)	Descriptive name of the broker.
parallel-scripts-limit (*integer:3..1000;* Default: off)	A parameter that defines how many scripts the on-message feature for this broker is allowed to run at the exact same time. Can be useful to reduce CPU, in cases when a large number of messages are constantly published.
password (*string*; Default:)	Password for the broker (if required by the broker).
port (*integer:0..4294967295*; Default: **1883**)	Network port used by the broker.
ssl (*yes | no*; Default: **no**)	Secure Socket Layer configuration.
username (*string*; Default:)	Username for the broker (if required by the broker).

An example of adding a broker:

```
/iot mqtt brokers add name="broker" address="192.168.88.33" port=1883 ssl=no client-id="test-client" auto-
connect=no keep-alive=60
```

The result:

```
/iot mqtt brokers print
 0 name="broker" address="192.168.88.33" port=1883 ssl=no client-id="test-client" auto-connect=no keep-alive=60
connected=no
```

Connect

To connect to the pre-configured MQTT broker, issue the command:

```
/iot mqtt connect broker="broker"
```

If the connection is successful, the "**connected**" parameter should change to "**yes**":

```
/iot mqtt brokers print
 0 name="broker" address="192.168.88.33" port=1883 ssl=no client-id="test-client" auto-connect=no keep-alive=60
connected=yes
```

Disconnect

To disconnect from the MQTT broker, issue the command:

```
/iot mqtt disconnect broker="broker"
```

To confirm that the broker was disconnected, issue the command below and it should indicate "**connected=no**":

```
/iot mqtt brokers print
 0 name="broker" address="192.168.88.33" port=1883 ssl=no client-id="test-client" auto-connect=no keep-alive=60
connected=no
```

Publish

Publish menu is used to send MQTT messages to the MQTT broker.

Property	Description
broker (*string*; Default:)	Select the broker, where to publish the message.
disconnect-after (*yes* / *no*; Default: **no**)	Parameter, that ensures that the connection with the broker will be automatically disconnected after the publish message is sent.
force (*yes* / *no*; Default: **yes**)	If set to "yes", when the connection with the broker is not yet established ("**connected=no**"), and the message is attempted to be published, RouterOS will try to establish an MQTT connection with the specified broker first and then publish the message. If set to "no", RouterOS will not be able to send the message, unless the connection is already established beforehand ("**connected=yes**").
message (*string*; Default:)	The message that you wish to publish to the broker.
qos (*integer: 0..4294967295*; Default: **0**)	Quality of service parameter, as defined by the broker.
retain (*yes* / *no*; Default: **no**)	Whether to retain the message or to discard it if no one is subscribed to the topic. This parameter is defined by the broker.
topic (*string*; Default:)	Topic, as defined by the broker.

An example of publishing the message:

```
/iot mqtt publish message="test-message" broker="broker" topic="my/test/topic"
```

Subscribe

> Please remember that if you have an on-going connection with the broker (the connection is in the "**connected=yes**" status) and you subscribe to the topic via that broker, you have to re-establish the connection!

This menu is used to subscribe to MQTT topics from the broker.

Property	Description

broker (*string*; Default:)	Select the broker, where to subscribe to.
force (*yes* / *no*; Default: **yes**)	If set to "yes", when the connection with the broker is not yet established ("**connected=no**"), and subscription is attempted, RouterOS will try to establish an MQTT connection with the specified broker first and then subscribe to the topic. If set to "no", RouterOS will not be able to subscribe to the topic, unless the connection is already established beforehand ("**connected=yes**").
qos (*integer: 0..4294967295*; Default: **0**)	Quality of service parameter, as defined by the broker.
topic (*string*; Default:)	Topic, as defined by the broker, where to subscribe to.

An example of a subscription:

```
/iot mqtt subscribe broker="broker" topic="my/test/topic"
```

Wildcard (single level "**+**" and multi-level "**#**") subscriptions are also supported (RouterOS **does not allow publishing** to wildcard topics **but allows subscribing** to them):

```
/iot mqtt subscribe broker="broker" topic="my/test/#"
/iot mqtt subscribe broker="broker" topic="my/test/+"
```

This means that if you subscribe to `topic="my/test/#"`, you will be able to receive messages published to any topic that begins with the pattern before the wildcard symbol "**#**" (e.g., `"my/test/topic"`, `"my/test/topic/something"`).

And, if you subscribe to `topic="my/test/+"`, you will be able to receive messages published on the topic +1 level (e.g., `"my/test/topic"`, `"my/test/something"`).

Subscriptions

This section is used to manage already-added subscriptions (that were previously added via the Subscribe section).

It has the same properties as the Subscribe section.

Starting with **v7.12beta9**, this menu allows you to add the "**on-message**" setting to your subscriptions.

Property	Description
on-message (*string*; Default:)	Configure a script that will be automatically initiated/run whenever a new message is received in the subscribed topic.

To check already subscribed topics, issue the command:

```
/iot mqtt subscriptions print
0 broker=broker topic="my/test/topic" qos=0
```

After you publish a test message as shown in the **Publish** section above:

```
/iot mqtt publish message="test-message" broker="broker" topic="my/test/topic"
```

You should be able to check the received message under:

```
/iot mqtt subscriptions recv print
 0 broker=broker topic="my/test/topic" data="test-message" time=2023-05-22 16:57:00
```

Received message list is limited to 1024 entries. After which, older entries will get overwritten with the new ones.

To clear stored messages, issue the command:

```
/iot mqtt subscriptions recv clear
```

To run a script (for example, a basic "log" script) whenever any new message appears in the subscribed topic, you can use the `on-message` feature:

```
/iot mqtt subscriptions set on-message={:log info "Got data {$msgData} from topic {$msgTopic}"} broker=broker 0
```

The script can use **$msgData** and **$msgTopic** variables. **$msgData** defines the MQTT message that was published and **$msgTopic** defines the MQTT topic, where the message was published. Both variables are automatically generated when a new message appears.

- **$msgData** and **$msgTopic** variables will not work when used in the "**System>Script**" section created scripts, meaning, they will not work inside "/iot mqtt subscriptions set **on-message={/system script run x}**" added scripts. Both variables will work only when they are used inside the "**on-message={}**" written script, like, for example, "**on-message={:log info "Got data {$msgData} from topic {$msgTopic}"}**".
- The same applies to global variable usage. If there are global variables that are "generated" using other scripts (variables that appear under System>Script>Environment section), they will not work inside the "on-message" script.

After you publish a new MQTT message to the subscribed topic, a new log entry should appear:

```
/log print
10:19:15 script,info Got data {test-message} from topic {my/test/topic}
```

A second example shows how to run a script whenever a specific message (keywords from the message) appears. To achieve a scenario, where we want to run a script only when the MQTT message has specific content or a keyword, we can utilize the **if** condition statement:

```
/iot mqtt subscriptions set 0 on-messag={:if ($msgData~"\\{\"test\":\"123\"\\}") do={:log info "Got data
{$msgData} from topic {$msgTopic}"}}
```

Or:

```
/iot mqtt subscriptions set 0 on-messag={:if ($msgData~"test") do={:log info "Got data {$msgData} from topic
{$msgTopic}"}}
```

As a result, on every received MQTT message, the script will check whether the if condition is true. If it is true (if **$msgData** contains the JSON string **{"test": "123"}** or if **$msgData** contains the string "**test**"), the log entry will be generated. Otherwise, nothing will happen.

Meaning, the script will be run only when you publish a message like this:

```
/iot mqtt publish broker=broker topic="my/test/topic" message="{\"test\":\"123\"}"
```

When you receive a message from a topic that falls under multiple subscriptions with `on-message` configuration, only **x1** `on-message` script will be run. RouterOS will choose which `on-meesage` script to run using the following logic/priority:

1. If the topic configured for the subscription is an exact match → first priority;
2. If the topic name is not an exact match (wildcard is used) → the second priority is for single 1v1 wildcard topics;
3. If the topic does not fall under the single 1v1 wildcard category → the third priority is for multi-level wildcard topics based on the topic level.

An example:

```
/iot mqtt subscriptions print
 0 broker=broker topic="some/sort/of/topic" qos=0 on-message="/system script run script1"

 1 broker=broker topic="some/#" qos=0 on-message="/system script run script2"

 2 broker=broker topic="some/sort/of/+" qos=0 on-message="/system script run script3"

 3 broker=broker topic="some/thing/#" qos=0 on-message="/system script run script4"
```

When you publish the data to `some/sort/of/topic`, script1 will be initiated → because the topic is an exact match.

When you publish the data to `some/sort/of/thing`, scrtip3 will be initiated → because it falls under the single 1v1 wildcard topic name.

When you publish the data to `some/name`, script2 will be initiated → because it falls under the multi-level wildcard topic name.

When you publish the data to `some/thing/else`, script 4 will be initiated → because it falls under the multi-level wildcard topic name (even though it is also matched by `some/#` wildcard, it is a level closer to `some/thing/#` entry).

Unsubscribe

Property	Description
broker (*string*; Default:)	Select the broker to unsubscribe from.
topic (*string*; Default:)	Select a topic, as defined by the broker, to unsubscribe from.

An example of unsubscribing from the broker and the topic is shown below:

```
/iot mqtt unsubscribe broker="broker" topic="my/test/topic"
```

Publishing RouterOS statistics using scripts

You can also use scripts to structure MQTT messages that contain RouterOS statistics. Then, you can apply the scheduler to run the script whenever you like.

For example, you can run a script like this (copy the content of the RouterOS code shown below into a new terminal and press "enter"):

```
/system script add dont-require-permissions=no name=mqttpublish owner=admin policy=\
    ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="#\
    \_Required packages: iot\r\
    \n\r\
    \n################################ Configuration ##########################\
    #######\r\
    \n# Name of an existing MQTT broker that should be used for publishing\r\
    \n:local broker \"broker\"\r\
    \n\r\
    \n# MQTT topic where the message should be published\r\
    \n:local topic \"my/test/topic\"\r\
    \n\r\
    \n################################### System ############################\
    #######\r\
    \n:put (\"[*] Gathering system info...\")\r\
    \n:local cpuLoad [/system resource get cpu-load]\r\
    \n:local freeMemory [/system resource get free-memory]\r\
    \n:local usedMemory ([/system resource get total-memory] - \$freeMemory)\r\
    \n:local rosVersion [/system package get value-name=version \\\r\
    \n\A0 \A0 [/system package find where name ~ \"^routeros\"]]\r\
    \n:local model [/system routerboard get value-name=model]\r\
    \n:local serialNumber [/system routerboard get value-name=serial-number]\r\
    \n:local upTime [/system resource get uptime]\r\
    \n\r\
    \n################################### MQTT ##############################\
    #######\r\
    \n:local message \\\r\
    \n\A0 \A0 \"{\\\"model\\\":\\\"\$model\\\",\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"sn\\\":\\\"\$serialNumber\\\",\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"ros\\\":\\\"\$rosVersion\\\",\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"cpu\\\":\$cpuLoad,\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"umem\\\":\$usedMemory,\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"fmem\\\":\$freeMemory,\\\r\
    \n\A0 \A0 \A0 \A0 \A0 \A0 \A0 \A0 \\\"uptime\\\":\\\"\$upTime\\\"}\"\r\
    \n\r\
    \n:log info \"\$message\";\r\
    \n:put (\"[*] Total message size: \$[:len \$message] bytes\")\r\
    \n:put (\"[*] Sending message to MQTT broker...\")\r\
    \n/iot mqtt publish broker=\$broker topic=\$topic message=\$message\r\
    \n:put (\"[*] Done\")"
```

The script collects the data from the RouterOS (model name, serial number, RouterOS version, current CPU, used memory, free memory, and uptime) and publishes the message (the data) to the broker in the JSON format:

```
/system script run mqttpublish
[*] Gathering system info...
[*] Total message size: 125 bytes
[*] Sending message to MQTT broker...

[*] Done
```

You can subscribe to the topic to check the results:

```
/iot mqtt subscriptions recv  print
 0 broker=broker topic="my/test/topic" data="{"model":"RB924i-2nD-BT5&BG77","sn":"E9C80EAEXXXX","ros":"7.9","
cpu":13,"umem":47476736,
     "fmem":19632128,"uptime":"02:21:18"}"
   time=2023-05-22 17:03:52
```

Do not forget to change the "Configuration" part of the script (topic and the broker) based on your settings.

MQTT and ThingsBoard configuration

Introduction

One of the many cloud services that you can use to monitor information that is sent by an MQTT publisher is Thingsboard. This article will demonstrate how to configure both Thingsboard and RouterOS to publish the data using the MQTT protocol. RouterOS, in this scenario, is going to act as a gateway and publish the data from the RouterBoard to the Thingsboard's server. Thingsboard, in this scenario, will act as an MQTT broker (server, where data will be posted).

Before we proceed with the settings, you need to either:

- a) Create an account in the Thingsboard's system. You can do so by following this link. This will allow you to use the ThingsBoard cloud solution for free for a limited/test time period.
- b) Set up your own server by following the guides. There is a community edition that can be installed and used free of charge.

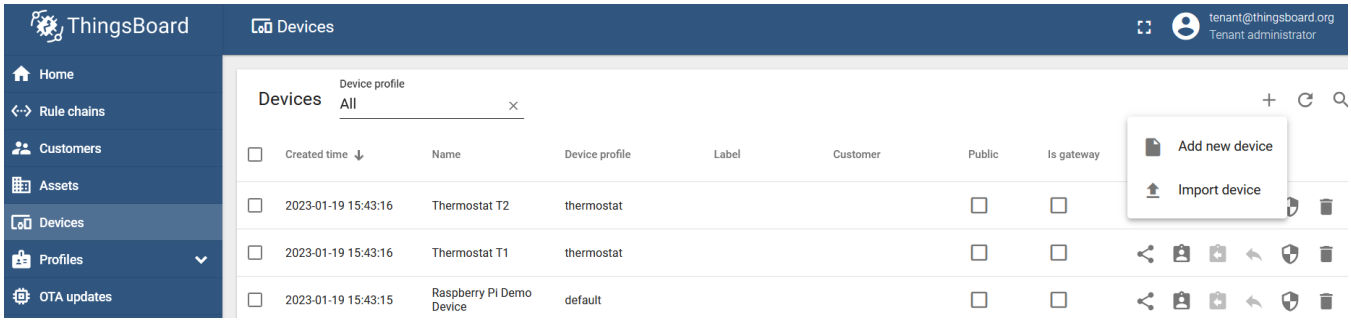
Please consider using **SSL MQTT (TCP port 8883 and certificates)**, instead of non-SSL MQTT (TCP port 1883). If you use non-SSL MQTT, the communication between the client (MQTT publisher) and the server (MQTT broker) can be easily sniffed/packet captured, and that will compromise authentication data (such as client-ids, usernames and passwords).

Thingsboard configuration

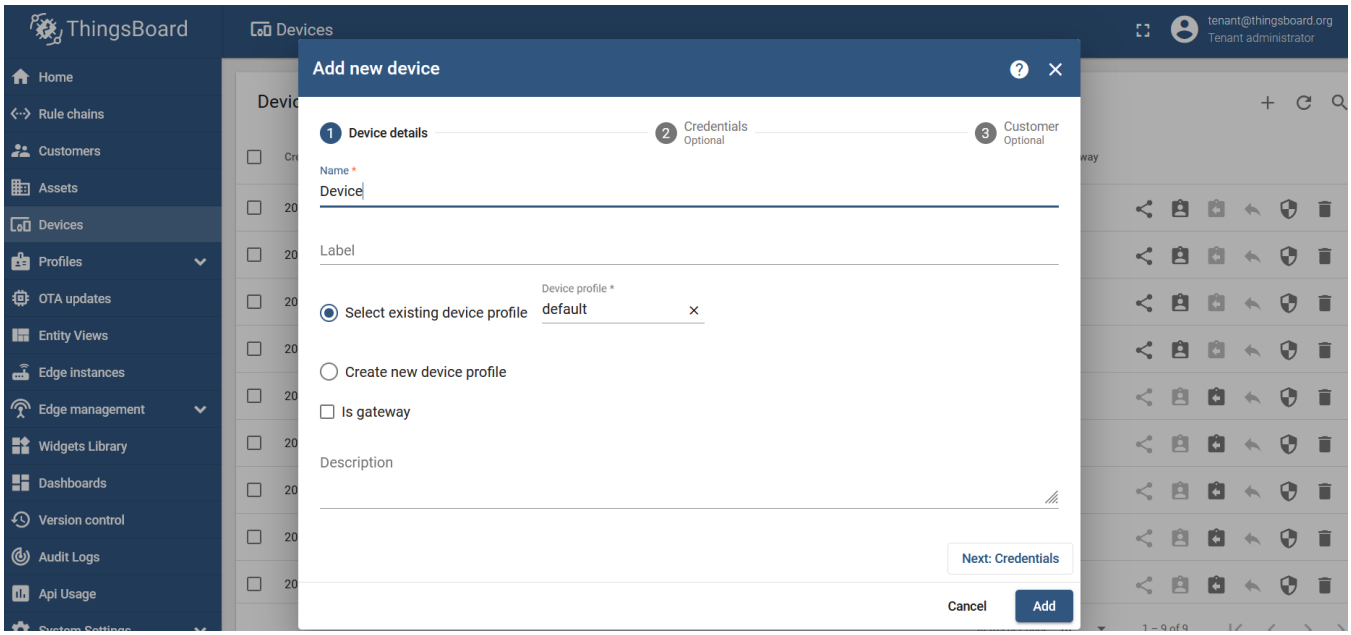
In this guide, we will showcase local instance/server installation configuration, but the same principles apply to the cloud option.

Access the login page via your browser and log in. Go to the "**Devices**" menu.

Create a new device by clicking on the add button "+" and "Add new device":



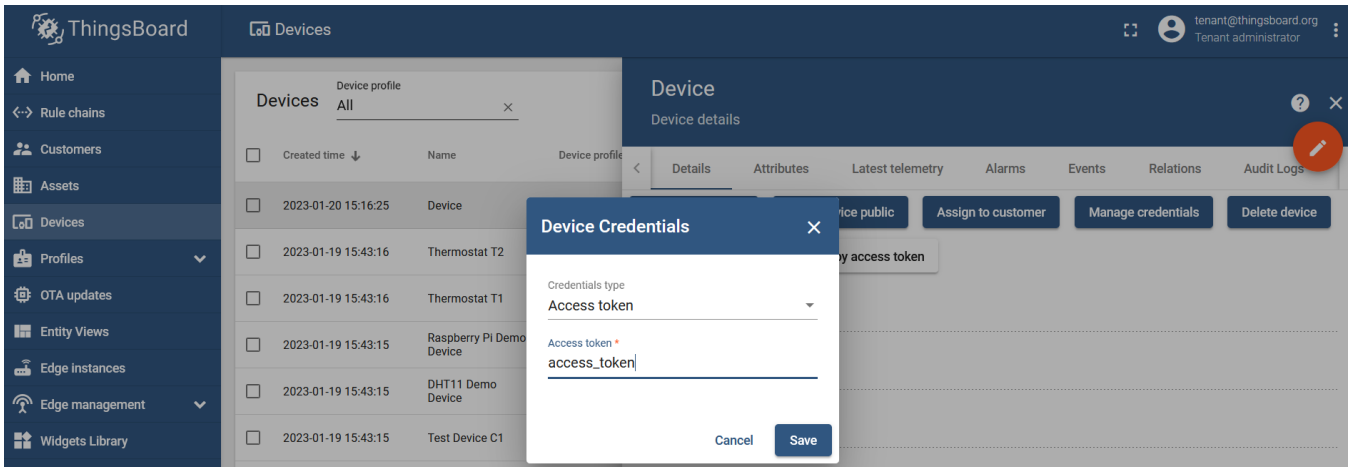
Enter the name of the device and click on "Add":



By default, access token authentication is selected for the newly created device.

Access token scenario

You can change the token by clicking on the created device and entering the "**Manage Credentials**" settings (in the "**Details**" section):

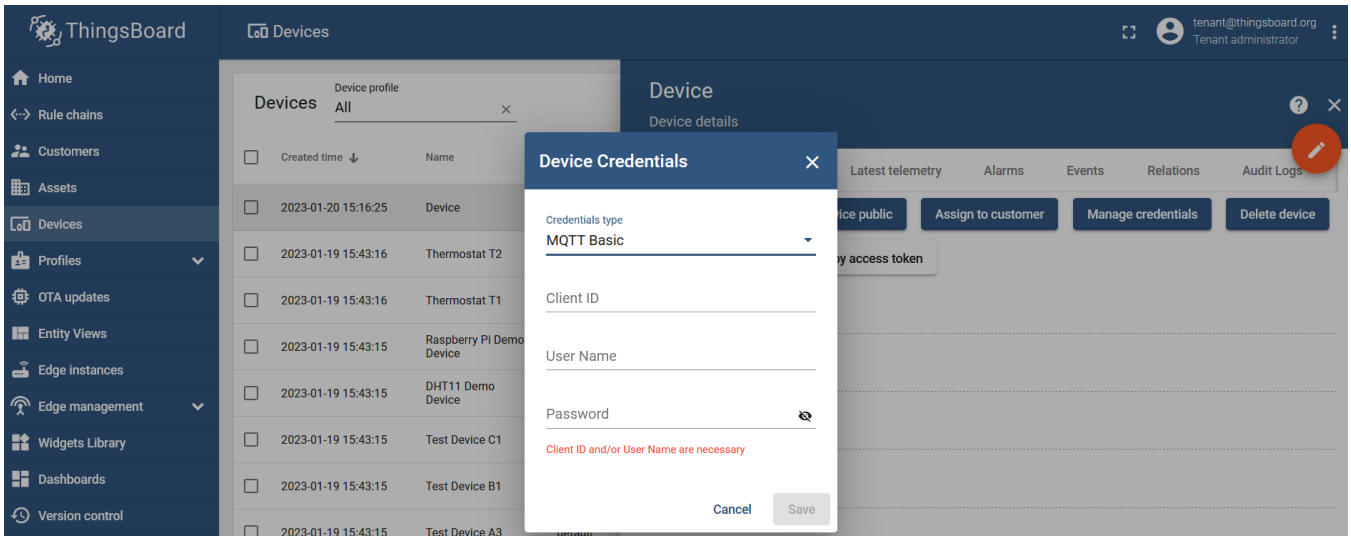


This token will be used as a "username" for the MQTT publisher (in RouterOS settings).

You can find more information by following the link.

MQTT Basic scenario

You can change the credentials type in the "**Device Credentials**" section for the specific device:



MQTT Basic scenario allows you to specify the Client ID, Username, and Password for the MQTT authentication.

You can find more information by following the link.

One-way SSL communication scenario

Recommended scenario to use!

This type of authentication requires you to use a server certificate for SSL communication. A server certificate must be generated and uploaded to the ThingsBoard instance.

To generate a server certificate, use this guide as a reference → generate the certificate (for example, using OPENSSL tool), install/upload it into the correct folder, and enable MQTT SSL in the ThingsBoard configuration file.

The configuration will be the same as shown in the **Access token** and **MQTT Basic scenarios** shown above. So choose either one.

The only difference, in this case, is the communication between the device and the server (you will only have to slightly change MQTT broker configuration in RouterOS settings which will be shown later on).

When using this scenario, the communication is going to be encrypted (using SSL).

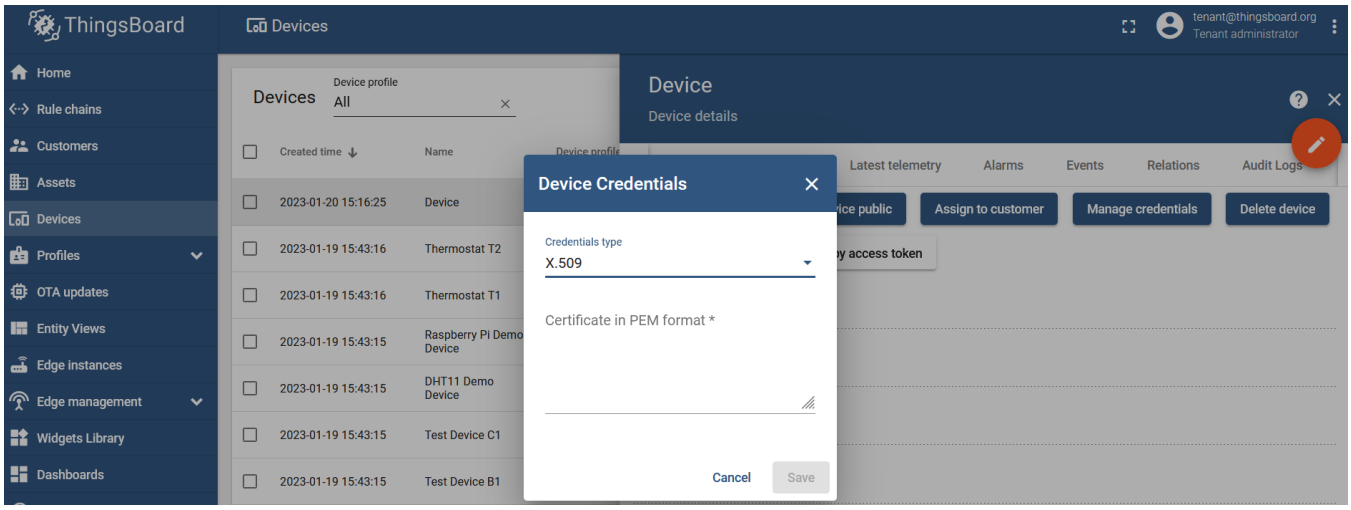
X.509 (two-way SSL communication) scenario

This type of authentication requires you to use a server certificate and a client certificate for SSL communication. A server certificate must be generated and uploaded to the ThingsBoard instance.

To generate a server certificate, use this guide as a reference → generate the certificate (for example, using OPENSSL tool), install/upload it into the correct folder, and enable MQTT SSL in the ThingsBoard configuration file.

To generate a client certificate, use this guide as a reference.

You can change the credentials type in the "**Device Credentials**" section for the specific device:



X.509 scenario uses a client certificate for authentication.

Once the certificate is generated (for example, using OPEN SSL), copy the RSA public key into the field and click on the "Save" button.

RouterOS configuration

***note**: In order to configure MQTT, make sure that **iot** package is installed beforehand.*

MQTT Broker

Access token scenario

Add an MQTT broker as shown below:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=1883 username=access_token
```

- Change the "`address`" to the actual IP/domain address of your ThingsBoard server;
- Change the "`username`" to the access token that you've used in the ThingsBoard settings.

MQTT Basic scenario

Add an MQTT broker as shown below:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x client-id=clientid password=password username=username
```

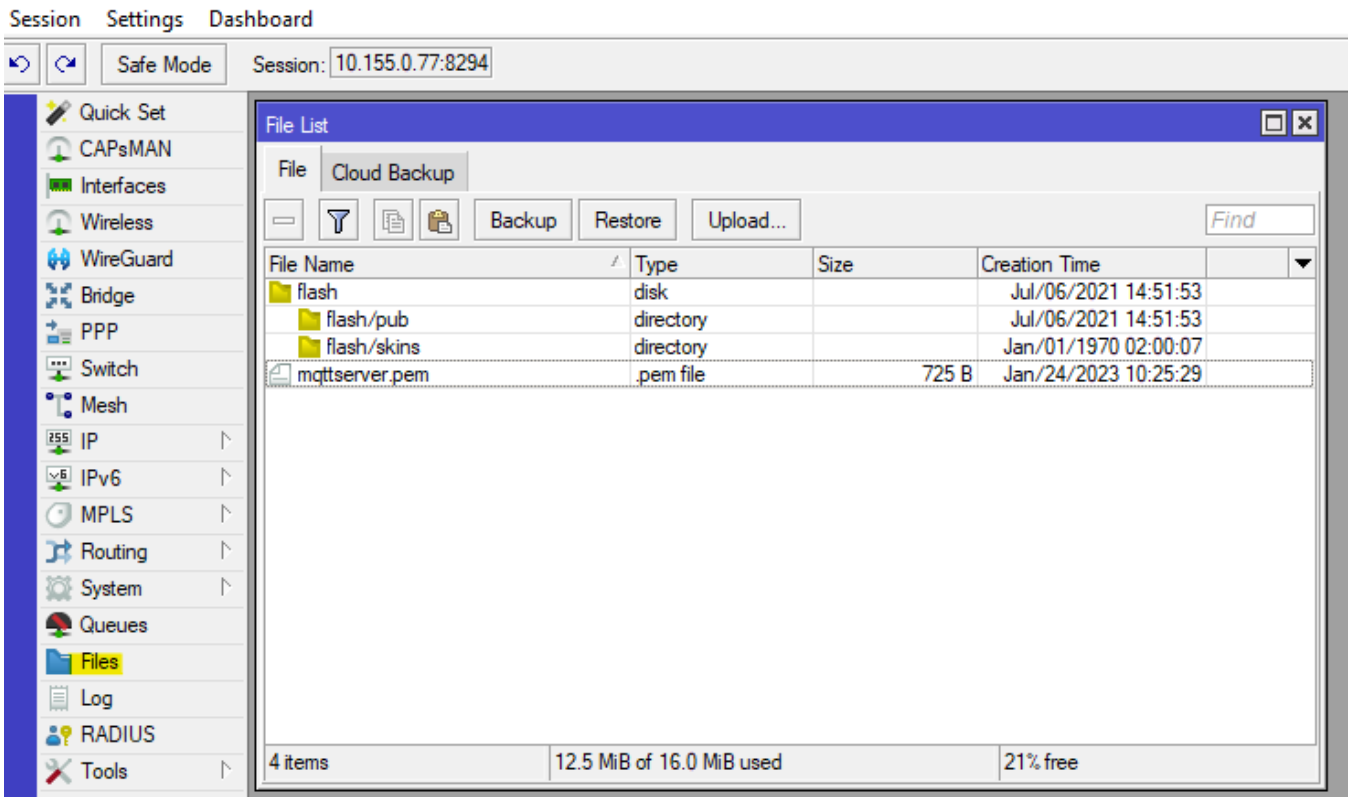
- Change "`address`" to the actual IP/domain address of your ThingsBoard server;
- Change the "`username`", "`password`" and "`client-id`" to the actual values that you've used in the ThingsBoard settings.

One-way SSL communication scenario

> Recommended scenario to use!

In this scenario, RouterOS needs to have a server certificate imported into its system.

Drag-and-drop server certificate, that was installed into the ThingsBoard, into the router's "File List" menu:



Import server certificate:

```
/certificate/import file-name=mqttserver.pem passphrase=""
```

When using **SSL one-way communication** and an **access token scenario**, add an MQTT broker as shown below:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=8883 username=access_token ssl=yes
```

- Change the "`address`" to the actual IP/domain address of your ThingsBoard server;
- Change the "`username`" to the access token that you've used in the ThingsBoard settings;
- Make sure to use "`port=8883`" (the MQTT SSL port that the server is listening to);
- Make sure to enable "`ssl=yes`".

When using **SSL one-way communication** and an **MQTT Basic scenario**, add an MQTT broker as shown below:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=8883 client-id=clientid password=password username=username
ssl=yes
```

- Change the "`address`" to the actual IP/domain address of your ThingsBoard server;
- Change the "`username`", "`password`" and "`client-id`" to the actual values that you've used in the ThingsBoard settings;
- Make sure to use "`port=8883`" (the MQTT SSL port that the server is listening to);
- Make sure to enable "`ssl=yes`".

X.509 (two-way SSL communication) scenario

Drag-and-drop the certificates into the router's "Files/File List" menu → *server certificate, client certificate, and its private key.*

Import certificates one by one:

```
/certificate/import file-name=mqttserver.pem passphrase=""
/certificate/import file-name=cert.pem passphrase=""
/certificate/import file-name=key.pem passphrase=""
```

Add an MQTT broker as shown below:

```
/iot/mqtt/brokers/add name=tb address=x.x.x.x port=8883 certificate=cert.pem_0 ssl=yes
```

- Change the "`address`" to the actual IP/domain address of your ThingsBoard server;
- Change the "`certificate`" selected to the actual client certificate name that you've imported;
- Make sure to use "`port=8883`" (the MQTT SSL port that the server is listening to);
- Make sure to enable "`ssl=yes`".

MQTT Publish

a) A quick MQTT publish test with a static value:

```
/iot/mqtt/publish broker="tb" topic="v1/devices/me/telemetry" message="{\"cpu\":\"7\"}"
```

b) In order to publish relevant data from the RouterOS to the Thingsboard, you can use the script shown below as a reference. The script collects the data from the RouterOS device (model name, serial number, RouterOS version, current CPU, used memory, free memory, and uptime) and publishes the message (the data) to the broker in the JSON format:

```
# Required packages: iot

################################## Configuration ##################################
# Name of an existing MQTT broker that should be used for publishing
:local broker "tb"

# MQTT topic where the message should be published
:local topic "v1/devices/me/telemetry"

##################################### System #####################################
:put ("[*] Gathering system info...")
:local cpuLoad [/system resource get cpu-load]
:local freeMemory [/system resource get free-memory]
:local usedMemory ([/system resource get total-memory] - $freeMemory)
:local rosVersion [/system package get value-name=version \
[/system package find where name ~ "^routeros"]]
:local model [/system routerboard get value-name=model]
:local serialNumber [/system routerboard get value-name=serial-number]
:local upTime [/system resource get uptime]

###################################### MQTT ######################################
:local message \
"{\"model\":\"$model\",\
\"sn\":\"$serialNumber\",\
\"ros\":\"$rosVersion\",\
\"cpu\":$cpuLoad,\
\"umem\":$usedMemory,\
\"fmem\":$freeMemory,\
\"uptime\":\"$upTime\"}"

:log info "$message";
:put ("[*] Total message size: $[:len $message] bytes")
:put ("[*] Sending message to MQTT broker...")
/iot mqtt publish broker=$broker topic=$topic message=$message
:put ("[*] Done")
```

2 script lines should be taken into account.

> *:local broker "tb"*

line, where you should specify the broker's name within the quotation marks "".

> *:local topic "v1/devices/me/telemetry"*

line, where you should specify the correct topic within the quotation marks "" (check Thingsboard's documentation for the exact topic that needs to be used).

The rest of the script configuration depends on the overall requirements.

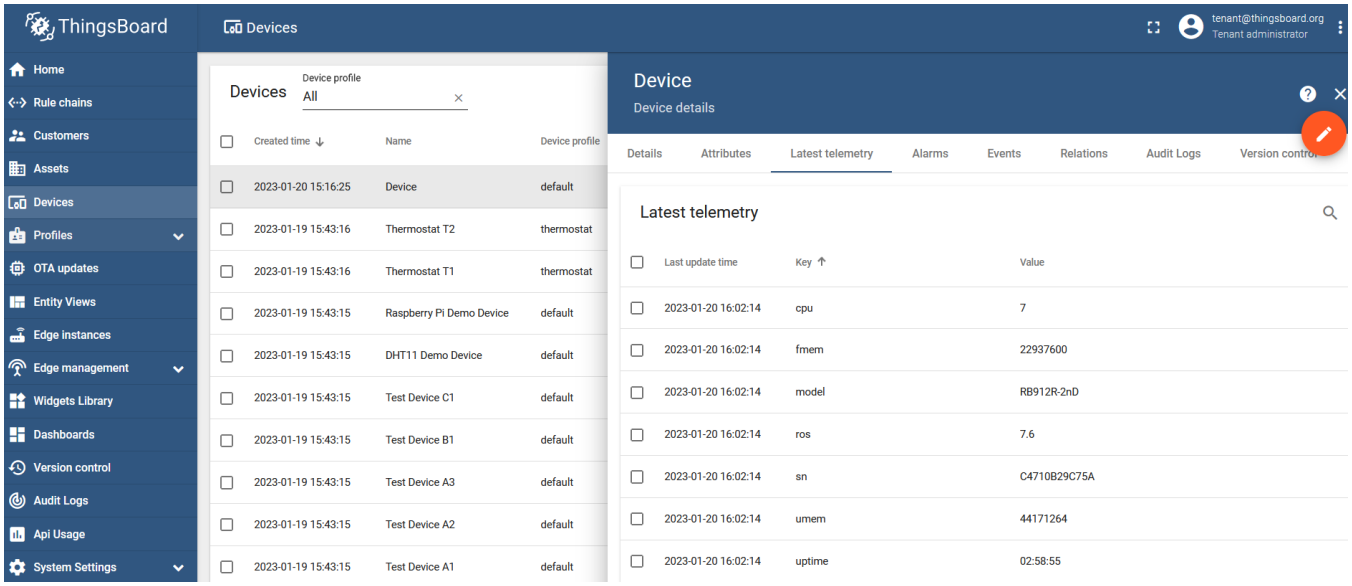
Copy and paste the above script into a notepad, and re-copy it again. Navigate to System>Scripts menu, add a new script there, and paste the script that is shown above. Name it, for example, script1.

To run the script, you can use the command line:

```
/system script run script1
```

Verification

You can check the received/published data for the device under the "Latest telemetry" section:



Hardware

In This Section:

Disks

Summary

Sub-menu: `/disk`

This menu will list all attached storage devices, presuming that they are supported and in working condition. This is especially useful for RouterBOARD devices with SD/CF/USB/SATA/NVMe slots and x86 systems with additional dedicated storage drives - as the built-in storage is quite small, an external drive comes in very handy when you want a big User Manager database, proxy cache or possibly SMB shares on your router.

You can add as many external or secondary drives as you want, and select any number of them for each of the mentioned feature usages. For example, User Manager could be used on 3 disks, one of them would be the active database, and the rest would be backups. You can then add a fourth disk, copy the active data to it - unmount - unplug it - and move to another server, to keep using the actual database. This means migration and backup are made easy!

Note: Starting from RouterOS 7.7 disks will carry names (slots) where they are physically connected

ROSE-storage - package adds additional enterprise data center functionality to RouterOS.
Current manual page does not include additional features included in **ROSE-storage** package.

Properties

Property	Description
eject-drive (*Integer*; Default:)	Safely unmounts (ejects) drive of your selection by using drive ID or NAME that is assigned to it. After issuing this command it can be removed from host device. If drive is sata/etc on x86, device must be previously also shut down or hot-plug for SATA interfaces.
format-drive ()	Command to initiate disk formatting process. Contains additional properties of its own. Such as "file-system" and "label". • *slot* - select disk (slot) that should be formatted • *file-system* ('fat32', 'ext4' or 'wipe') - Format disk with type FAT32 or EXT4 or securely wipe all data • *label* • mbr partition table - make mbr partition table
reset-counters	resets disk (slot) statistics

Flags

Property	Description
X - disabled	Disabled device

E - empty	Empty slot
B - BLOCK-DEVICE	The "B - BLOCK-DEVICE"- Flag means that this device works using blocks for input/output operations. In the context of RouterOS, its distinction is crucial, as it helps determine whether a device is functioning as a data carrier or simply providing information about the disk layout structure. This difference becomes important when considering the extender with the device behind it. If a device is marked with the letter "B", this indicates its ability to be used as storage or memory. In contrast, devices that do not have a "B" mark are designed primarily to understand the structure of the disk. This allows to quickly recognize the presence of a PCIe or SAS expander, as well as detect the presence of drives in the first expander. In addition, it allows you to estimate the speed of the connection to which each device is connected. However, the most notable benefit of the "B" flag is its ability to instantly indicate whether a device can be formatted or used for RAID purposes.
M - mounted	Mounted partition
F - formatting	The device is currently in the formatting process
p - partition	The device has a partition
f - raid-member-failed	These options are used with the ROSE package.
r - raid-member	
c - encrypted	
g - guid-partition-table	
t - nvme-tcp-export	
i - iscsi-export	
s - smb-export	
n - nfs-export	
O - tcg-opal-self-encryption-enabled	
o - tcg-opal-self-encryption-supported	

Settings

Property	Description
auto-smb-sharing (yes | no; Default: no)	Enables dynamic SMB shares when new disk/partition item is added in "/disk"

auto-smb-user (list of strings; Default:)	Default value for smb-sharing/smb-user setting, when new disk/partition item is added in "/disk"
auto-media-share (yes | no; Default: no)	Enables media dynamically when new disk/partition item is added in "/disk"
auto-media-interface (list of strings; Default:)	Interface that will be used in dynamic instance for ip/media when new disk/partition item is added in "/disk"

Notes

With "auto-smb-sharing=yes" and "/ip smb share enabled=auto" SMB server gets enabled when a storage device is physically plugged in

Examples

Formatting attached storage unit - Simple

1. Disk is attached, and already mounted automatically by the system.

```
[admin@MikroTik] > disk print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE, FS
#    SLOT  MODEL           SERIAL            INTERFACE                  SIZE           FREE  FS
0 BM usb1  USB Flash Disk  FBA0911260071572  USB 2.00 480Mbps  2 004 877 312  1 921 835 008  ext4
```

```
[admin@MikroTik] > /file print
 # NAME                        TYPE          SIZE CREATION-TIME
 0 skins                       directory          jan/01/1970 03:00:01
 1 pub                         directory          feb/04/1970 21:31:40
 2 usb1                        disk               mar/07/2022 14:05:16
```

2. Formatting the disk, in either of two supported file-systems (ext4 or fat32).

```
[admin@MikroTik] > /disk format-drive usb1 file-system=ext4 mbr-partition-table=no
  formatted: 100%
```

3. It's done! Drive is formatted and should be automatically mounted after formatting process is finished.

Formatting attached storage unit - Detailed

Let us presume that you have added a storage device to your device that is running RouterOS. System will try to automatically mount it and in such case if storage is formatted in a supported file-system and partition record, it will be found in "/files" menu moments after you plugged it in to the host device.

If not, here is what you have to do.

1. Do a quick print of disk menu, to make sure that router sees the attached storage.

```
[admin@MikroTik] > disk print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE, FS
#    SLOT  MODEL           SERIAL            INTERFACE                  SIZE           FREE  FS
0 BM usb1  USB Flash Disk  FBA0911260071572  USB 2.00 480Mbps  2 004 877 312  1 921 835 008  ext4
```

We can here see that system sees one storage drive and also that it is formatted with a known file-system type.

When running file menu print-out we also see that is mounted.

```
[admin@MikroTik] > file print
 # NAME     TYPE    SIZE CREATION-TIME
 0 usb1     disk         mar/07/2022 14:05:16
 1 skins    directory    jan/01/1970 03:00:01
 2 pub      directory    feb/04/1970 21:31:40
```

2. To formatting drive - we issue command with previously know id or name(slot) and with desired file-system (ext4 or fat32), we can also assign label to device as I did in this example and make mbr partition table

```
[admin@MikroTik] > /disk format-drive usb1 file-system=ext4 label=usb-flash mbr-partition-table=yes
  formatted: 100%
```

Note: In printout, you can see that there is a progress percentage counter in formatting process. For larger storage drives, it might take longer for this process to finish, so be patient.

If multiple GPT partitions are needed format drive without partition table and add them manually:

```
[admin@MikroTik] > /disk format-drive usb1 file-system=ext4 label=usb-flash mbr-partition-table=no
  formatted: 100%
```

```
[admin@MikroTik] > /disk add type=partition parent=usb1 partition-size=200M
[admin@MikroTik] > /disk add type=partition parent=usb1 partition-size=500M
[admin@MikroTik] > /disk add type=partition parent=usb1 slot=usb1-last-partition
```

Note: Slot (partition or disk name) is assumed automatically, but can be overwritten by using slot parameter.
If partition size is not used all available space will be used from last partition.
To offset partition start "partition-offset" parameter can be used.

Web-Proxy cache configuration example

Enter proxy cache path under IP -> Proxy menu and web proxy store is automatically created in files menu. If a non-existent directory path is used, an additional sub-directory is also created automatically.

```
[admin@MikroTik] >  /ip proxy set cache-path=usb1/cache-n-db/proxy/

...

[admin@MikroTik] >  /file print
 # NAME                                          TYPE                          SIZE CREATION-TIME
 0 skins                                         directory                          mar/02/2015 18:56:23
 1 sys-note.txt                                  .txt file                     23   jul/03/2015 11:40:48
 2 usb1                                         disk                              jul/03/2015 11:35:05
 3 usb1/lost+found                              directory                          jul/03/2015 11:34:56
 4 usb1/cache-n-db                              directory                          jul/03/2015 11:41:54
 4 usb1/cache-n-db/proxy                        web-proxy store                    jul/03/2015 11:42:09
```

Log on disk configuration example

When configuring logging on disk make sure that you create directories in which you want to store the log files manually, as non-existent directories will NOT be automatically created in this case.

```
[admin@MikroTik] >  /system logging action set disk disk-file-name=/disk1/log

...

[admin@MikroTik] >  /file print where name~"disk1/log"
 # NAME                                            TYPE                              SIZE CREATION-TIME
 0 disk1/log                                       directory                             jul/03/2015 12:44:09
 1 disk1/log/syslog.0.txt                          .txt file                         160 jul/03/2015 12:44:11
```

Note: Logging topics such as firewall, web-proxy and some other topics that tend to save a large amount or rapid printing of logs on system NAND disk might cause it to wear out faster, so using some attached storage or remote logging is recommended in this case or save data in RAM folder

Allocate RAM to folder

Starting from 7.7 it is possible to add folders linked to RAM. Folders will be emptied on reboot or power loss.
RAM will be filled up to tmpfs-max-size and if this variable in not provided - up to 1/2 from available RAM.

```
[admin@MikroTik] >  /disk add type=tmpfs tmpfs-max-size=100M slot=RAM
[admin@MikroTik] > file print
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME            TYPE       SIZE     CREATION-TIME
0  RAM             disk                dec/12/2022 11:01:48
```

Grounding

Introduction

Shielded cable installation infrastructure (towers and masts), as well as antennas and the router itself, must be properly grounded, and lightning arrestors must be installed on all external antenna cables (near the antennas or on the antennas themselves) to prevent equipment damage and human injury. Note that lightning arrestors will not have any effect if not grounded.

Use 1 AWG (7mm in diameter) wire with corrosion-resistant connectors for grounding. Be sure to check that the grounding infrastructure you use is indeed functional (as opposed to decorative-only grounding present on some sites). For smaller devices, you can use thinner wire.

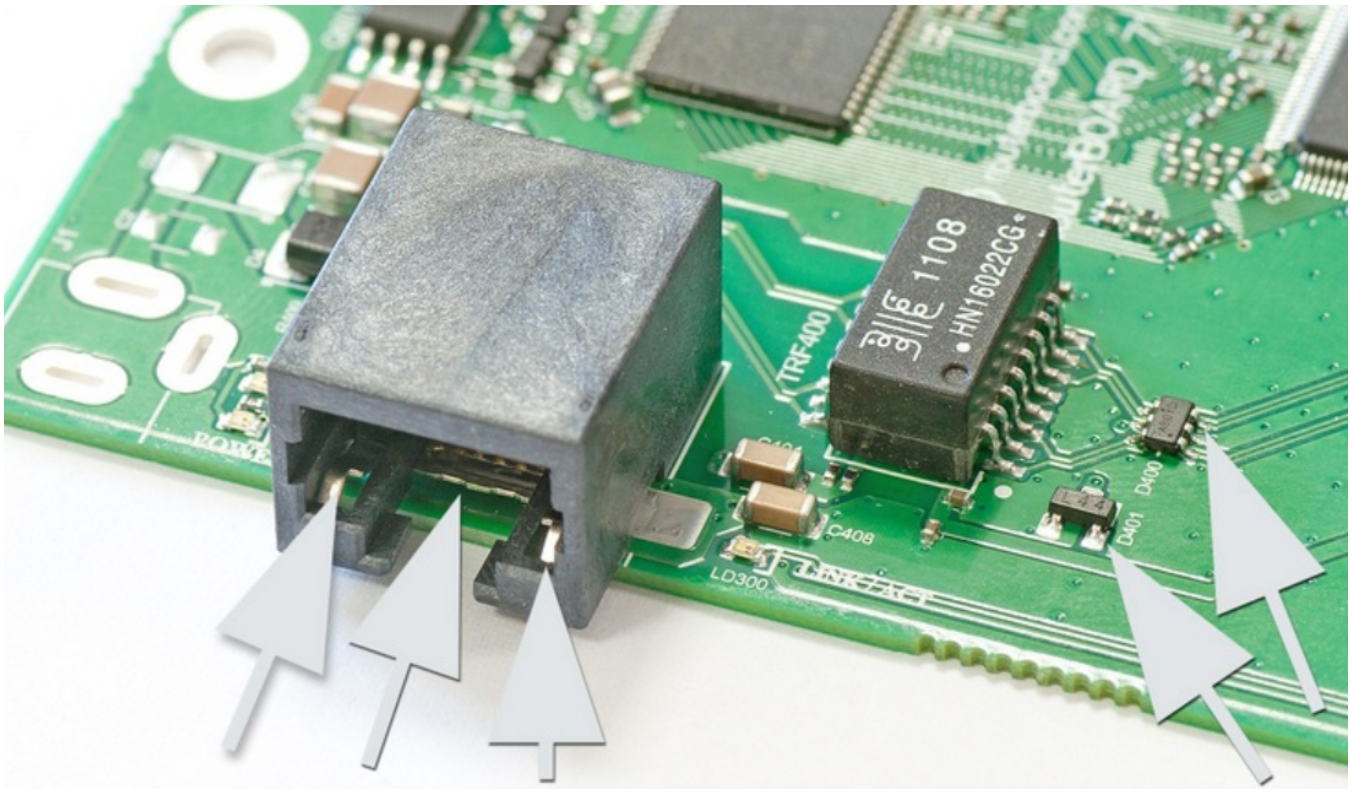
1. Only shielded and outdoor usage Ethernet cables should be used, the magnetic shield should be grounded via shielded RJ-45 connector or via an additional wire that is soldered to RJ45 or ground wire.
2. The grounding wire should be connected to RouterBOARD (to the mounting point where the board is fastened to the outdoor box), this wire is connected to the bottom of the tower, and the connection to the tower is according to the standards. The antenna grounding wire is connected near the RouterBOARD Outdoor case, this wire could be connected to the same RouterBOARD grounding wire.
3. Ethernet port lightning protectors are not recommended, as most of them are not intended to use for PoE (they are shortening PoE supply). If protectors are used, they could be placed at the outdoor case, where RouterBOARD and grounding pads are connected.

Example grounding wire attachment screw on an outdoor case:

ESD Protection on RouterBOARD devices

1. Three arrows mark the grounding inside the ethernet port, the shielded cable connects it's a shield to these two grounding pins via the metallic Ethernet connector.
2. The middle arrow points to the metal plate inside the port, which connects the grounding pins to the board. The board needs to be grounded at the mounting hole (put grounding wire on the screw when you mount the board inside a case). Any surges will go from the grounding pins to the grounding plate, to the board, and then to the grounding installation.
3. The two separate arrows show the ESD protection chips on the board - in case there was no shielded cable, to protect the CPU and other parts of the board.

The protection is not too effective if you only use shielded cable, and don't ground the board itself. You need to do both things to be successful. See below for possible methods, option 1 is recommended.



Grounding RouterBOARD installations

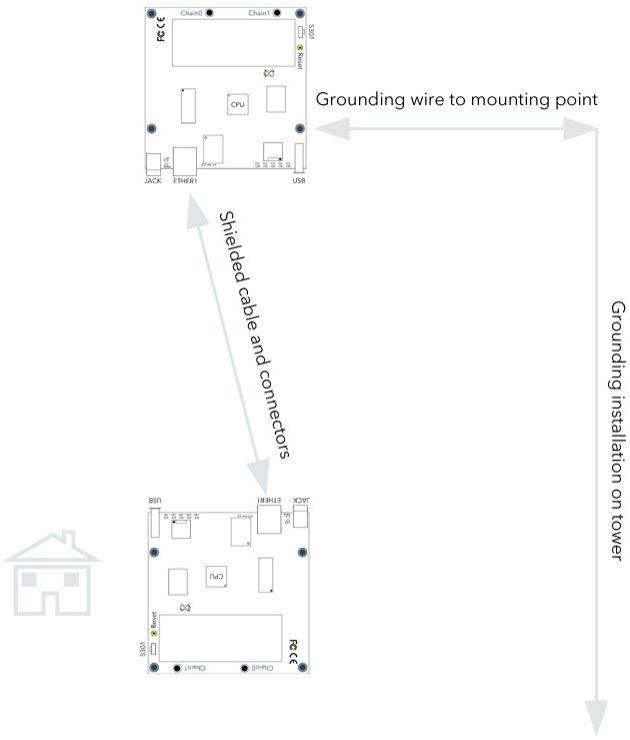
There are two methods, one of them more effective than the other.

PoE with shielded connectors:

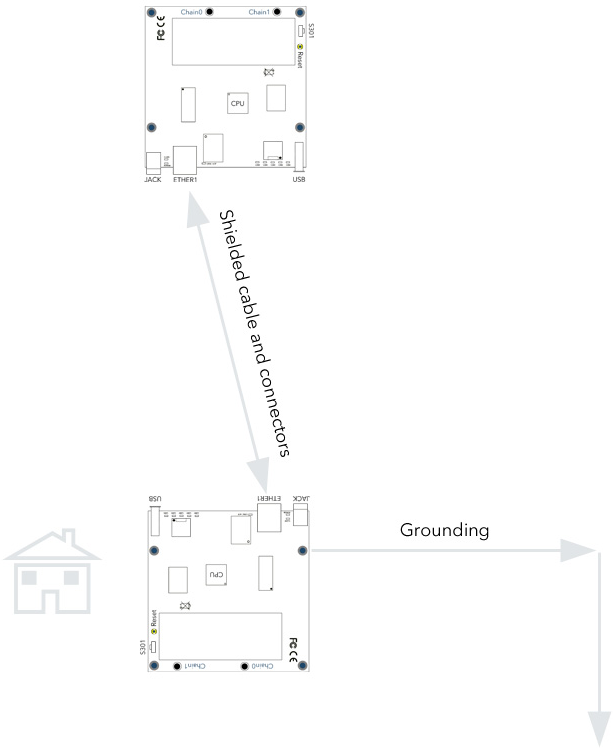
1. **Using a Shielded cable + Board is grounded**: If you connect grounding to the mounting point of the RB711 (or the mounting loop inside the SXT door), you don't necessarily need to ground the device at another end of the shielded cable. Just using a shielded cable is enough. Special PoE is also not needed. This is the best option to protect against all ESD damage.
2. **Using only shielded cable**: If you can't ground the RB711/SXT/another device itself, you can ground the device on the other end of your shielded cable (switch, router, etc). If you need to use PoE, the injector with a metal shielding around connectors will be required, because it allows the shielded cable to be used. This method is not recommended, better ground the board itself also (option 1).

Illustrations of the above methods

Method #1 (shielded cable + grounding of the device):



Method #2 (only shielded cable):



You should not use PSE with the positive terminal connected to PE if it powers a MikroTik device. It may cause a short circuit, harm you and your device.

Even if you don't ground the outdoor wireless device, and only use a shielded cable, you should still ground the device it's connected to (indoors). Ie. the switch, routerboard, or PC.

LCD Touchscreen

Summary

RouterBOARD 2011U and CCR series devices are equipped with a resistive touchscreen, for quick access to device stats and simple configuration options. Touchscreen requires pressure against the surface to register a touch, therefore light swipes and quick/short taps might not get registered (as opposed to a capacitive touchscreen commonly found on phones). If you find trouble operating the screen with your finger, you can also try a stylus, or opposite end of a pen.

Configuration

Sub-menu: `/lcd`

Property	Description
backlight-timeout (*time interval: 5m..2h* | *never*; Default: **30m**)	Time after which LCD touchscreen is turned off
color-scheme (*dark* | *light*; Default: depends on RouterBoard model)	Changes to color scheme with a dark or light background.
default-screen (*informative-slideshow|interfaces|log|main-menu|stat-slideshow|stats|stats-all*; Default: **main-menu**)	Default screen that is showed after startup.
enabled (*yes* | *no*; Default: **yes**)	Turns LCD touchscreen on/off. When off, it stops and resets statistics gathering and closes the LCD program.
read-only-mode (*yes* | *no*; Default: **yes**)	Enables or disables Read-Only mode. If Read-Only mode is enabled, then menus which can be used to change configuration are hidden.
time-interval (*min* | *hour* | *daily* | *weekly*; Default: **min**)	Time interval of displayed interface statistics in Stats screen
touch-screen (*enabled* | *disabled*, Default: **enabled**)	Enable/disable touch screen input.

Available functions:

- **backlight** - Turns on/off LCD touchscreen backlight, LCD program remains working;
- **recalibrate** - Starts LCD Touchscreen Calibration process;
- **show** - Set the screen which is displayed on the LCD;
- **take-screenshot** - Creates image of currently displayed LCD screen.

LCD Touchscreen Calibration

Before the LCD touchscreen can be used, it needs to be calibrated at least once. After the first successful calibration, data is stored on the router. If no calibration values are present, calibration process will start automatically.

During the calibration/recalibration you must touch 4 points drawn on the screen. Three of the points are used to calculate calibration variables and the 4th point is used to test whether the calibration was successful. If calibration is unsuccessful, calibration variables are not saved. At the end (after touching 4th point) a message is displayed with the calibration result.

Take LCD Screenshot

Take-screenshot function allows to create BMP image of currently displayed LCD screen and saves it in File List with specified name. Screenshots without file name are not saved, screenshots with an existing file name are overwritten.

Example:

```
[admin@MikroTik]  /lcd take-screenshot file-name=screen-1
Screenshot taken
[admin@MikroTik] >
```

LCD Interfaces

Sub-menu: `/lcd interface`

Interfaces menu provide configuration for interface display timing in Stat Slideshow. Up to 10 additional (non-physical) interfaces can be added to the LCD.

Property	Description
disabled (*yes* | *no*; Default: **no**)	Sets whether interface is shown in Stat Slideshow
max-speed (*integer* | *auto*; Default:)	Maximum interface speed that is used to determine bandwidth usage in All interface graphs and Interfaces screens. "auto" value can be set only for physical interfaces.
timeout (*time interval: 1s..1m*; Default: **10s**)	Time of displaying interface slide

Available functions:

- **display** - Display the interface in Stats screen.

All Interface Graph Screen

Sub-menu: `/lcd interface pages`

A Page is an screen that can contain up to 12 interface bar graphs. Sub-menu allows to configure which interfaces are shown in a page. Up to 5 pages can be added to all interface graph screen and up to 12 interfaces per page. To add an interface to a page, it first must be added under /lcd interfaces sub-menu.

Property	Description
interfaces (interface names; Default:)	Interfaces that are shown in the screen. Must have at least 1 interface.

LCD Informative Screens

Sub-menu: `/lcd screen`

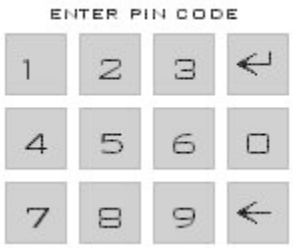
Screens menu provide configuration for slide display timing in Informative Slideshow.

Property	Description
disabled (*yes* | *no*; Default: **no**)	Defines whether item is ignored or used in Informative Slideshow

timeout (*time interval: 1s..1m*; Default: **10s**)	Time of displaying informative slide

LCD PIN Code

Sub-menu: `/lcd pin`



PIN code number allows to protect sensitive menus on the LCD screen. The PIN number will be asked if Read-Only mode is disabled and you add an IP address, reset or reboot the router. Default PIN is **1234**

Property	Description
pin-number (*number*; Default: **1234**)	PIN protection code
hide-pin-number (*yes* / *no*; Default: **no**)	Whether to show the typed digits on the LCD screen or hide them with asterisks

LCD screens/modes

Since v6.0, LCD has a menu structure. Menu screens consist of buttons that are used to navigate the menus. A scrollbar is shown on the right side of the screen if it does not fit on the actual display. The screen can be dragged up or down to access more options if they are available. At the top of each menu screen is a "Back" button that jumps to the previous screen.

Startup

If the router has default configuration - user named "admin" with no password, then a warning on LCD will appear. This screen shows IP's assigned to the interfaces which could be used to connect to the router. Otherwise the Main menu screen is displayed after booting up.

Interfaces

Interfaces menu displays all the Ethernet and Wireless interfaces. Bandwidth usage is shown similar to the All interface graph screen. From the Interfaces screen you can choose a specific interface to look at. The following options are available:

- Info (only for physical interfaces) - menu which shows information about the interface;
- Registration Table (only for wireless) - menu which shows all the registered clients for the wireless interface and their respective signal strengths;
- Addresses - menu which lists all the addresses assigned to the interface;
- Stats - menu which allows to jump to the selected interface in the "Stats" screen. You can directly choose to show Bandwidth or Packets.

Info | Registration Table | Addresses | Stats selection

Stats

Stats screen shows single interface graphs for RX and TX. Values are updated from right to left (newest to oldest). Info that is shown: RX/TX rate and packets.

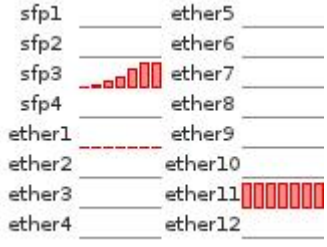
Interface name is shown at the top right, it is trimmed if it's too long (last characters are cut off). The top right corner shows the time interval for the values. Following time values are available:

- **Min (Minute)** - shows values for the last minute. Unit = second. Vertical line separates first 30 seconds. Total values: 30 + 24;
- **Hour** - shows values for the last hours. Unit = 5 minutes. Vertical lines separate 1 hour. Total values: 12 + 12 + 3;
- **Daily** - shows values for the last days. Unit = hour. Vertical lines separate 1 day. Total values: 12 + 12 + 3;
- **Weekly** - shows values for the last weeks. Unit = day. Vertical lines separate 1 week. Total values: 7 + 7 + 4;

Motions:

- Tap - tapping the finger against the touch screen without moving it too much.
  - If a tap lands into the top right corner of the screen (square box 1/4 of the screen height), info time interval is changed: Min -> Hour -> Daily -> Weekly -> Min...
  - Otherwise a tap cycles through graph info: rate -> packets -> rate...
- Swipe/Drag - while holding the finger down, move in any direction. The changes should be highlighted during the drag.
  - Up - Go to Main menu
  - Down - Select All Interface graph screen
  - Left - Next interface
  - Right - Previous interface

All Interface Graph Screen



All interface graph screen shows the RX/TX bandwidth usage about all interfaces. The max values are calculated like this - for Ethernet interfaces it's the negotiated rate or set speed. For wireless interfaces it's calculated from used band, channel-width and chain count using the theoretical values. The goal of this screen is to see how values are related to each other for a single interface.

Motions:

- Swipe/drag.
  - Up - Back (to Stats screen).
  - Left - Next page.
  - Right - Previous page.

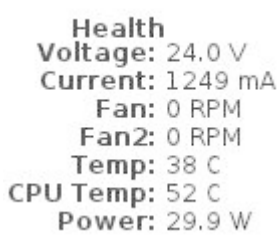
Stat Slideshow

Stat Slideshow screen is similar to the "Stats" screen, but the interfaces are switched after they timeout. Settings for slideshow are stored in RouterOS submenu /lcd interface

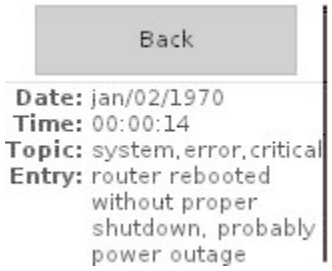
Informative Slideshow

Submenu /lcd screen Informative Slideshow screen cycles through screens with various system information:

- Aggregate traffic;
- Aggregate packets;
- Resources;
- System;
- Health;
- Date & time.

		
System	Resources	Health

Log



The Log screen shows 5 last log entries where log action=echo.

Reboot and Reset Configuration

These screens are only available when Read-Only mode is disabled. To access any of the screens, the Pin number must be entered. If the Pin authentication is successful, the user must confirm the desired action by pressing the "Yes" button, or cancel by pressing - "No".

LEDs

Summary

Sub-menu: `/system leds`

RouterOS allows configuring each LED's activity the way that the user wishes. It is possible to configure the LEDs to display wireless strength, blink the LEDs on interface traffic activity, and many other options.

For example, default led configuration on Groove

```
[admin@MikroTik] /system leds> print
Flags: X - disabled
# TYPE INTERFACE LEDS
0 wireless-signal-strength led1
led2
led3
led4
led5
1 interface-activity ether1 user-led
```

⚠ RB Groove uses five LEDs for wireless strength and one for ethernet activity monitoring.

Property Description

Property	Description
disabled (*yes* | *no*; Default: **no**)	Whether an item is disabled
interface (*string*; Default:)	Name of the interface which will be used for led status. Applicable only if **type** is interface specific.
modem-signal-treshold (*integer [-113..-51]*; Default:)	Applicable if a type is **modem-signal**
leds (*list of leds*; Default:)	List of led names used for a status report. For example, wireless signal strength will require more than one led.

type (*align-down | align-left | align-right | align-up | ap-cap | fan-fault | flash-access | interface-activity | interface-receive | interface-speed | interface-speed-1G | interface-speed-25G | interface-status | interface-transmit | modem-signal | modem-technology | off | on | poe-fault | poe-out | wireless-signal-strength | wireless-status*; Default:)	Type of the status: • align-down - light the led if the w60g device needs to be aligned downwards for the best signal quality • align-left - light the led if the w60g device needs to be aligned to the left • align-right - light the led if the w60g device needs to be aligned to the right • align-up - light the led if the w60g device needs to be aligned upwards • ap-cap - blink on CAP initializing with CAPsMAN, steady on once connected • fan-fault - light the led when any of the devices controlled fans stop working • flash-access - blink the led on flash access • interface-activity - blink the led on interface (traffic) activity • interface-receive - blink the led on interface received a traffic • interface-speed - light the led when interface works in 10Gbit rate • interface-speed-1G - light the led when interface works in 1Gbit rate • interface-speed-25G - light the led when interface works in 25Gbit rate • interface-speed-100G - light the led when interface works in 100Gbit rate • interface-status - light the led on interface status change • interface-transmit - blink the led on interface transmitted traffic • modem-signal - blink the led on 3G modem signal (either USB or miniPCIe) • modem-technology - turns on LEDs in order of modem technology generation: GSM; 3G; LTE; single led turns on only when LTE is active. • off - turn off the led • on - turn on the led • poe-fault - light the led when PoE out budget is close to the maximum supported limit • poe-out - light the led when interface PoE out turns on • wireless-signal-strength - light the leds displaying wireless signal (requires more than one led) • wireless-status - light the led on wireless status change.

LED Settings

Global settings are stored in LEDs Setting menu.

Sub-menu: `/system leds setting`

Property	Description

all-leds-off (*after-1h* | *after-1min* | *immediate* | *never*; Default: **never**)	Whether and when all LEDs of a router can be turned off

The listed devices support turning off their LEDs (LED dark mode), however, some LEDs still cannot be turned off due to the device design factors.

Indoor devices

RouterBoard	LED description
CRS305-1G-4S+	Turns off all LEDs except Ethernet LED and Power LED
CRS309-1G-8S+	Turns off all LEDs except Ethernet LEDs
RB5009UG+S+IN	Turns off all LEDs
L009UiGS-RM	Turns off PWR, USR and Ether1 LEDs
L009UiGS-2HaxD-IN	Turns off PWR, USR and Ether1 LEDs
RB760iGS (hEX S)	Turns off Power LED and SFP LED
RB924i-2nD-BT5&BG77; RB924iR-2nD-BT5&BG77 (KNOT series)	Turns off all LEDs
RB951Ui-2HnD	Turns off all LEDs except Power LED
RB951Ui-2nD (hAP); RB952Ui-5ac2nD (hAP ac lite); RB952Ui-5ac2nD-TC (hAP ac lite TC)	Turns off all LEDs except Power LED
RB962UiGS-5HacT2HnT (hAP ac)	Turns off all LEDs except Port5 PoE LED
RBcAP2n; RBcAP2nD (cAP)	Turns off all LEDs
RBcAPGi-5acD2nD (cAP ac); RBcAPGi-5acD2nD-XL (cAP XL ac)	Turns off all LEDs
RBD25G/RB25GR-5HPacQD2HPnD (Audience)	Turns off all LEDs except Ethernet LEDs
RBD52G-5HacD2HnD-TC (hAP ac^2)	Turns off all LEDs
RBD53iG-5HacD2HnD (hAP ac^3)	Turns off all LEDs
RBD53G-5HacD2HnD-TC (Chateau series)	Turns off all LEDs
RBwsAP5Hac2nD (wsAP ac lite)	Turns off all LEDs
C52iG-5HaxD2HaxD-TC (hAP ax^2)	Turns off all LEDs except Ethernet LEDs
C53UiG+5HPaxD2HPaxD (hAP ax^3)	Turns off all LEDs except Ethernet LEDs
S53UG+5HaxD2HaxD-TC (Chateau ax series)	Turns off all LEDs except Ethernet LEDs

Wireless Systems

RouterBoard	LED description
CubeG-5ac60ay (Cube 60Pro ac); CubeG-5ac60ay-SA (CubeSA 60Pro ac)	Turns off all LEDs
CubeG-5ac60ad (Cube 60G ac)	Turns off all LEDs
RB912R-2nD-LTm (ltAP mini / ltAP mini LTE kit)	Turns off all LEDs
RB912UAG-6HPnD (BaseBox 6)	Turns off all LEDs
RBD23UGS-5HPacD2HnD (NetMetal ac^2)	Turns off all LEDs
RBLDF-2nD (LDF 2); RBLDF-5nD (LDF 5); RBLHGR	Turns off all LEDs
RBLDFG-5acD (LDF 5 ac)	Turns off all LEDs except Ethernet LED

RBLHG2nD (LHG 2); RBLHG2nD-XL (LHG XL 2)	Turns off all LEDs
RBLHG5nD (LHG 5); RBLHG5HPnD (LHG HP5); RBLHG5HPnD-XL (LHG XL HP5)	Turns off all LEDs
RBLHGG-5acD (LHG 5 ac); RBLHGG-5acD-XL (LHG XL 5 ac); RBLHGG-5HPacD2HPnD (LHG XL 52 ac); RBSXTsqG-5acD (SXTsq 5 ac)	Turns off all LEDs except Ethernet LED
RBLHGG-60ad (Wireless Wire Dish)	Turns off all LEDs
LHGGM&EG18-EA (LHG LTE18 kit)	Turns off all LEDs
RBLtAP-2HnD (LtAP)	Turns off all LEDs except Ethernet LEDs
RBSXTsq-60ad (SXTsq Lite60); RBCube-60ad (Cube Lite60)	Turns off all LEDs
RBwAPG-60ad (Wireless Wire)	Turns off all LEDs
RBwAPGR-5HacD2HnD (wAP ac)	Turns off all LEDs except Ethernet LED

Examples

Basic example

LED control via CLI commands for scripting purposes:

```
#add led entry with specific type "on" or "off" to leds menu
/system leds add leds=led1 type=off
#to control led
/system leds set [find where leds="led1"] type=on
or
/system leds set [find where leds="led1"] type=off
```

Enable the User ACT LED to show current CAP status on an RB951

```
/system leds
add leds=user-led type=ap-cap
```

Modem Signal Strength example

The whole modem-signal strength range is [-113..-51] and the modem-signal-threshold increases the weakest signal limit to -91 so the signal range for LED indication is [-91..-51]. That range is divided into equal parts depending on number of LEDs configured for modem-signal LED trigger. The first LED turns on when signal is above -91 and the last LED turns on when signal reaches -51.

```
/system leds
add interface=lte1 leds=led1,led2,led3,led4,led5 modem-signal-treshold=-91 type=modem-signal
```

Modem Access Technology example

These LED trigger examples turn on LEDs in order of modem technology generation: GSM; 3G; LTE.

- 1 LED: led1 turns on when LTE is active;

```
/system leds add interface=lte1 leds=led1 modem-type=modem-technology
```

- 2 LEDs: led1 - 3G; led2 - LTE;

```
/system leds
add interface=lte1 leds=led1,led2 modem-type=modem-technology
```

- 3 LEDs: led1 - GSM; led2 - 3G; led3 - LTE

```
/system leds add interface=lte1 leds=led1,led2,led3 modem-type=modem-technology
```

MTU in RouterOS

Introduction

It is the sole responsibility of administrators to configure the Maximum Transmission Unit (MTU) such that intended services and applications can be successfully implemented in the network. In other words - administrators must make sure that MTUs are configured in a way that packet sizes do not exceed the capabilities of network equipment.

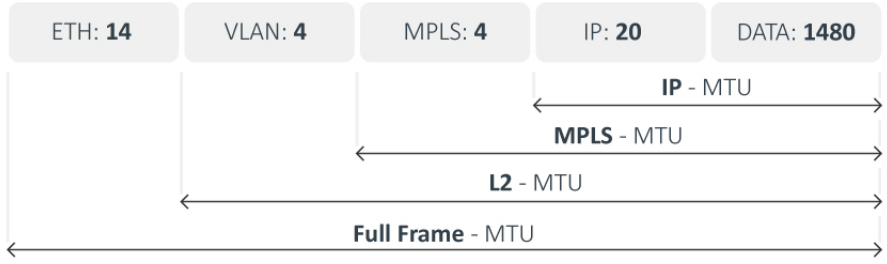
Originally MTU was introduced because of the high error rates and low speed of communications. Fragmentation of the data stream gives the ability to correct corruption errors only by resending corrupted fragments, not the whole stream. Also on low-speed connections such as modems, it can take too much time to send a big fragment, so in this case, communication is possible only with smaller fragments.

But in the present day we have much lower error rates and higher speed of communication, this opens a possibility to increase the value of MTU. By increasing the value of MTU we will result in less protocol overhead and reduce CPU utilization mostly due to interrupt reduction. This way some non-standard frames started to emerge:

- **Giant** or **Jumbo** frames - frames that are bigger than standard (IEEE) Ethernet MTU;
- **Baby Giant** or **Baby Jumbo** frames - frames that are just slightly bigger than standard (IEEE) Ethernet MTU;

It is common now for Ethernet interfaces to support physical MTU above standard, but this can not be taken for granted. Abilities of other network equipment must be taken into account as well - for example, if 2 routers with Ethernet interfaces supporting physical MTU 1526 are connected through an Ethernet switch, in order to successfully implement some application that will produce these big Ethernet frames, the switch must also support forwarding such frames.

Maximum Transmission Unit



Mikrotik RouterOS recognizes several types of MTU:

- IP/Layer-3/L3 MTU
- MPLS/Layer-2.5/L2.5 MTU
- MAC/Layer-2/L2 MTU
- Full frame MTU

Full frame MTU

Full frame MTU indicates the actual size of the frame that is sent by a particular interface. Frame Checksum is not included as it is removed by an ethernet driver as soon as it reaches its destination.

MAC/Layer-2/L2 MTU

L2MTU indicates the maximum size of the frame without the MAC header that can be sent by this interface.

In RouterOS L2MTU values can be seen in the "/interface" menu. L2MTU support is added for all Routerboard related Ethernet interfaces, VLANs, Bridge, VPLS, and wireless interfaces. Some of them support the configuration of the L2MTU value. All other Ethernet interfaces might indicate L2MTU only if the chipset is the same as Routerboard Ethernets.

This will allow users to check if the desired setup is possible. Users will be able to utilize additional bytes for VLAN and MPLS tags, or simply increase interface MTU to get rid of some unnecessary fragmentation.

This table shows *max-l2mtu* supported by Mikrotik RouterBoards (available in the "/interface print" menu as the value of the read-only "max-l2mtu" option):

Model name	MTU description
RB SXT series, RB LHG, RB LDF, PL6411-2nD, PL7411-2nD, RB711 series, wAP R-2nD, RB912R-2nD-LTm (LtAP mini), RB Metal series, RB SXT Lite series, RB Groove series, Cube Lite60, LHG Lite60	ether1:2028
RB SXT G series, RB DynaDish, wAP ac, RB QRT series, RB711G series, RB911G, RB912UAG	ether1:4076
RB OmniTik series, RB750, RB750UP, RB751U-2HnD, RB951-2n	ether1:4076; ether2-ether5:2028
RB OmniTik ac series, RB750GL, RB750Gr2	ether1-ether5:4074
RB mAP, RB mAP lite, RB cAP, RB wAP	ether1-ether2:2028
RB750r2, RB750P-PBr2, RB750UPr2, RB941-2nD, RB951Ui/RB952Ui series	ether1-ether5:2028
RB750Gr3	ether1-ether5:2026
RB751G-2HnD, RB951G-2HnD	ether1-ether5:4074
RB962UiGS, RB960PGS	ether1-ether5:4074; sfp1:4076
RB LHGG series	ether1:9214
LHG XL 52 ac	ether1:9214; sfp1:9214
RB1100Hx2, RB1100AHx2	ether1-ether10:9498; ether11:9500; ether12-ether13:9116
RB4011iGS+ series	ether1-ether10:9578; sfp-sfpplus1:9982
CCR1009 series	ether1-ether4:10224; ether5-ether8:10226; sfp1:10226; sfp-sfpplus1:10226
CCR1016 series	ether1-ether12:10226; sfp1-sfp12:10226; sfp-sfpplus1:10226
CCR1036 series	ether1-ether12:10226; sfp1-sfp4:10226; sfp-sfpplus1-sfp-sfpplus2:10226
CCR1072 series	ether1:9116; sfp-sfpplus1-sfp-sfpplus8:10226
CCR2004-1G-12S+2XS	ether1:9586; sfp-sfpplus1-sfp-sfpplus12:9578; sfp28-1 - sfp28-2:9578
CCR2004-16G-2S+	ether1-ether16:9582; sfp-sfpplus1-sfp-sfpplus2:9586
CCR2116-12G-4S+	ether1-ether12:9570; ether13:9586; sfp-sfpplus1-sfp-sfpplus4:9570
CCR2216-1G-12XS-2XQ	ether1:9586; sfp28-1 - sfp28-12:9570; qsfp28-1-1 - qsfp28-2-4:9570
CRS109-8G-1S	ether1-ether8:4064; sfp1:4064
CRS125-24G-1S	ether1-ether24:4064; sfp1:4064

CRS112-8G-4S, CRS112-8P-4S	ether1-ether8:9204; sfp9-sfp12:9204
CRS106-1C-5S	sfp1-sfp5:9204; combo1:9204
CRS210-8G-2S+	ether1-ether8:9204; sfp-sfpplus1:9204; sfpplus2:9204
CRS212-1G-10S-1S+	ether1:9204; sfp1-sfp10:9204; sfpplus1:9204
CRS226-24G-2S+	ether1-ether24:9204; sfp-sfpplus1:9204; sfpplus2:9204
CRS326-24G-2S+, CSS326-24G-2S+	ether1-ether24:10218; sfp-sfpplus1:10218; sfpplus2:10218
CRS317-1G-16S+	ether1:10218; sfp-sfpplus1-sfp-sfpplus16:10218
CRS328-24P-4S+	ether1-ether24:10218; sfp-sfpplus1-sfp-sfpplus4:10218
CRS328-4C-20S-4S+	combo1-combo4:10218; sfp1-sfp20:10218; sfp-sfpplus1-sfp-sfpplus4:10218
CRS305-1G-4S+	ether1:10218; sfp-sfpplus1-sfp-sfpplus4:10218
CRS309-1G-8S+	ether1:10218; sfp-sfpplus1-sfp-sfpplus8:10218
netFiber 9/IN (CRS310-1G-5S-4S+)	sfp1-sfp5:10218; sfp-sfpplus1-sfp-sfpplus4:10218
CRS310-8G+2S+IN	ether1-ether8:10218; sfp-sfpplus1-sfp-sfpplus2:10218
CRS312-4C+8XG	combo1-combo4:10218; ether1-ether8:10218; ether9:2028
netPower 15FR (CRS318-1Fi-15Fr-2S)	ether1-ether16:10218; sfp1-sfp2:10218
netPower 16P (CRS318-16P-2S+)	ether1-ether16:10218; sfp-sfpplus1-sfp-sfpplus2:10218
CRS326-4C+20G+2Q+	combo1-combo4:10218; ether1-ether20:10218; qsfpplus1-1-qsfpplus2-4:10218; ether21:2028
CRS326-24S+2Q+	sfp-sfpplus1-sfp-sfpplus24:10218; qsfpplus1-1-qsfpplus2-4:10218; ether1:2028
CRS354-48G-4S+2Q+, CRS354-48P-4S+2Q+	sfp-sfpplus1-sfp-sfpplus4:10218; qsfpplus1-1-qsfpplus2-4:10218; ether1-ether48:10218; ether49:2028
CRS504-4XQ-IN	ether1:2028; qsfp28-1-1 - qsfp28-4-4:10218
CRS510-8XS-2XQ-IN	ether1:2028; sfp28-1 - sfp28-8:10218; qsfp28-1-1 - qsfp28-2-4:10218
CRS518-16XS-2XQ	ether1:2028; sfp28-1 - sfp28-16:10218; qsfp28-1-1 - qsfp28-2-4:10218
CSS610-8G-2S+, CSS610-8P-2S+	ether1-ether8:10218; sfp-sfpplus1-sfp-sfpplus2:10218
D52G-5HacD2HnD (hAP ac²)	ether1-ether5:9124
C52iG-5HaxD2HaxD (hAP ax^2)	ether1-ether5:9214
C53UiG+5HPaxD2HPaxD (hAP ax^3)	ether1-ether5:9214
L41G-2axD (hAP ax lite)	ether1-ether4:2026
cAP ac	ether1-ether2:9124

GPEN21	ether1-ether2:10222; sfp1: 10222
wAP60G, LHG60G	ether1:9124
RB260GS series, CSS106-5G-1S, CSS106-1G-4P-1S	ether1-ether5:9198; sfp1:9198
RBFTC11	ether1:4046; sfp1:4046
RBM33G	ether1-ether3:2026
RBM11G	ether1:2026
RB760iGS	ether1-ether5:2026; sfp1:2026
RB411 series	ether1:1526
RB433 series, RB450, RB493 series	ether1:1526; ether2-ether3:1522
RB450Gx4	ether1-ether5:9214
RB411GL	ether1:1520
RB433GL, RB435G , RB450G, RB493G	ether1-ether3:1520
RB800	ether1-ether2:9500; ether3:9116
RB850Gx2	ether1-ether5:1580
RB921UAGS, RB922UAGS	ether1:4076; sfp1:4076
D23UGS-5HPacD2HnD (NetMetal ac²)	ether1:9214 ; sfp1:9214
RB953GS	ether1-ether2:4074; sfp1:4074; sfp2:4076
RB2011 series	ether1-ether5:4074; ether6-ether10:2028; sfp1: 4074
RB3011 series	ether1-ether5:8156; ether6-ether10:8156; sfp1: 8158
RB5009 series	ether1-ether8: 9796; sfp-sfpplus1: 9796
L009 series	ether1: 8158; ether2-ether8: 8154; sfp1: 8154
RB44Ge	ether1-ether4:9116

All wireless interfaces in RouterOS (including Nstreme2) support 2290 byte L2MTU.

L2MTU configuration changes evoke all interface reloads (link down/link up) due to necessary internal processes.
It is recommended to configure L2MTU with caution by keeping in mind that it can cause short interruption with connected devices.

MPLS/Layer-2.5/L2.5 MTU

Configured in the "/mpls interface" menu, specifies the maximal size of the packet, including MPLS labels, that is allowed to send out by the particular interface.

Make sure that MPLS MTU is smaller or equal to L2MTU. MPLS MTU affects packets depending on what action the MPLS router is performing. It is strongly recommended that MPLS MTU is configured to the same value on all routers forming the MPLS cloud because of the effects MPLS MTU has on MPLS switched packets. This requirement means that all interfaces participating in the MPLS cloud must be configured to the smallest MPLS MTU values among participating interfaces, therefore care must be taken to properly select the hardware to be used.

You can read more about MPLS MTU here.

MPLS Switching

If the packet with labels included is bigger than MPLS MTU, MPLS tries to guess the protocol that is carried inside the MPLS frame:

- If this is an IP packet, MPLS produces an ICMP Need Fragment error. This behavior mimics IP protocol behavior. Note that this ICMP error is not routed back to the originator of a packet but is switched towards the end of LSP so that the egress router can route it back.
- If this is not an IP packet, MPLS simply drops it, because it does not know how to interpret the contents of the packet. This feature is very important in situations where MPLS applications such as VPLS are used (where frames that are MPLS tagged are not IP packets, but e.g. encapsulated Ethernet frames as in the case of VPLS) - if somewhere along the LSP MPLS MTU will be less than packet size prepared by ingress router, frames will simply get dropped.

IP ingress

When a router first introduces a label (or labels) on an IP packet, and the resulting packet size including MPLS labels exceeds MPLS MTU, the router behaves as if interface MTU was exceeded - either fragment packet in fragments that do not exceed MPLS MTU when labels are attached (if IP Don't Fragment is not set) or generate ICMP Need Fragmentation error that is sent back to the originator.

VPLS ingress

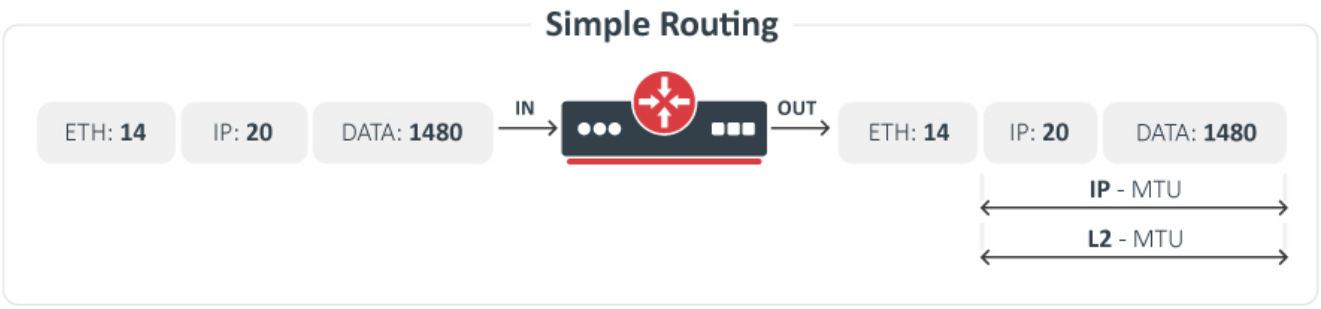
When the router encapsulates the Ethernet frame for forwarding over VPLS pseudowire, it checks if packet size with VPLS Control Word (4 bytes) and any necessary labels (usually 2 labels - 8 bytes), exceeds MPLS MTU of the outgoing interface. If it does, VPLS fragments packets so that it honors the MPLS MTU of the outgoing interface. A packet is defragmented at the egress point of the VPLS pseudowire.

Setup Examples

In these examples, we will take a look at frames entering and leaving the router via Ethernet interfaces.

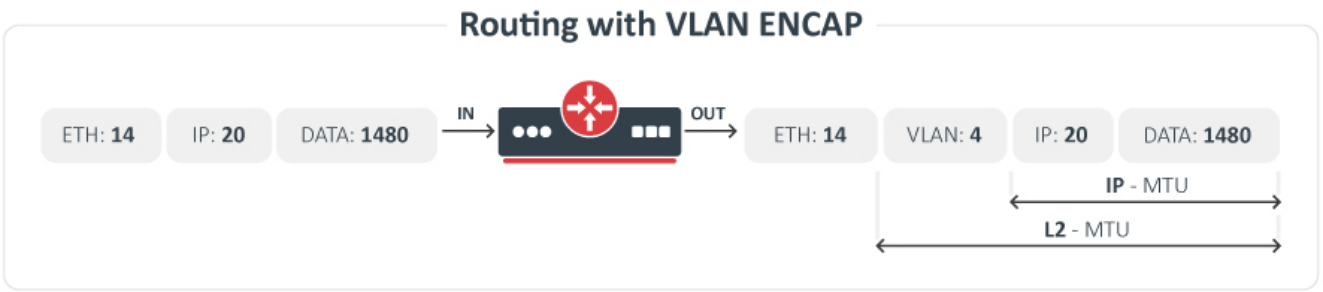
Simple Routing

The image shows the packet MTU size for simple routing, packets size is not modified.



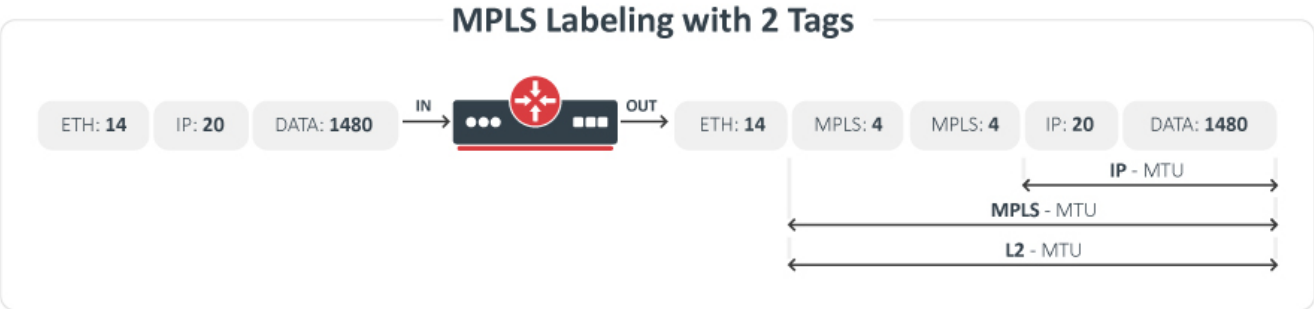
Routing with VLAN Encap

Each VLAN tag is 4 bytes long, the VLAN tag is added by a router. L2-MTU is increased by 4 bytes.



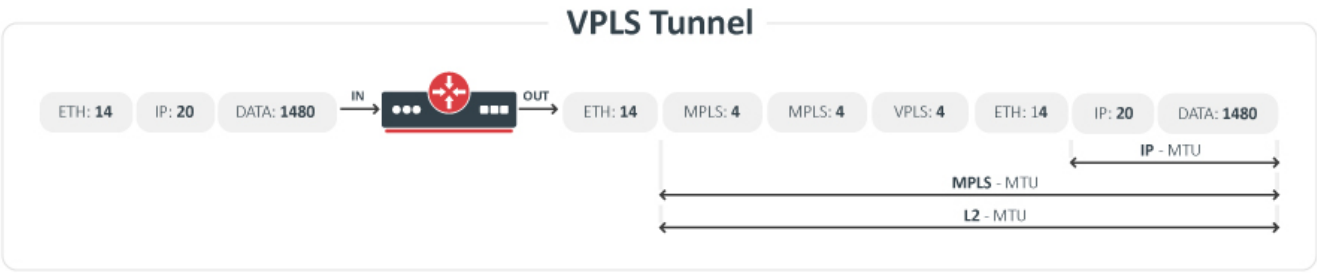
Simple MPLS with Tags

When MPLS is used as a plain replacement for IP routing, only one label is attached to every packet, therefore packet size increases by 4 bytes, we have the situation with two MPLS labels. In order to be able to forward standard size (1500 bytes) IP packets without fragmentation, MPLS MTU must be set to at least 1508 for two MPLS labels.



VPLS Tunnel

Two MPLS labels are present when a remote endpoint is not directly attached. One MPLS label is used to get to a remote endpoint, the second label is used to identify the VPLS tunnel.

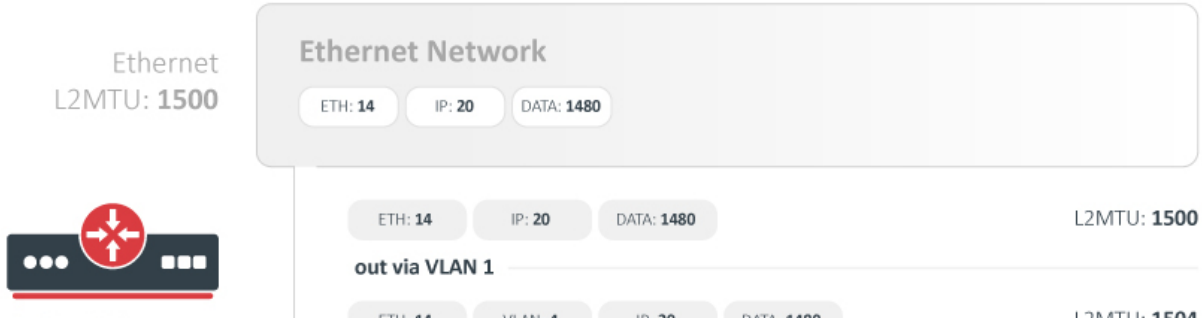


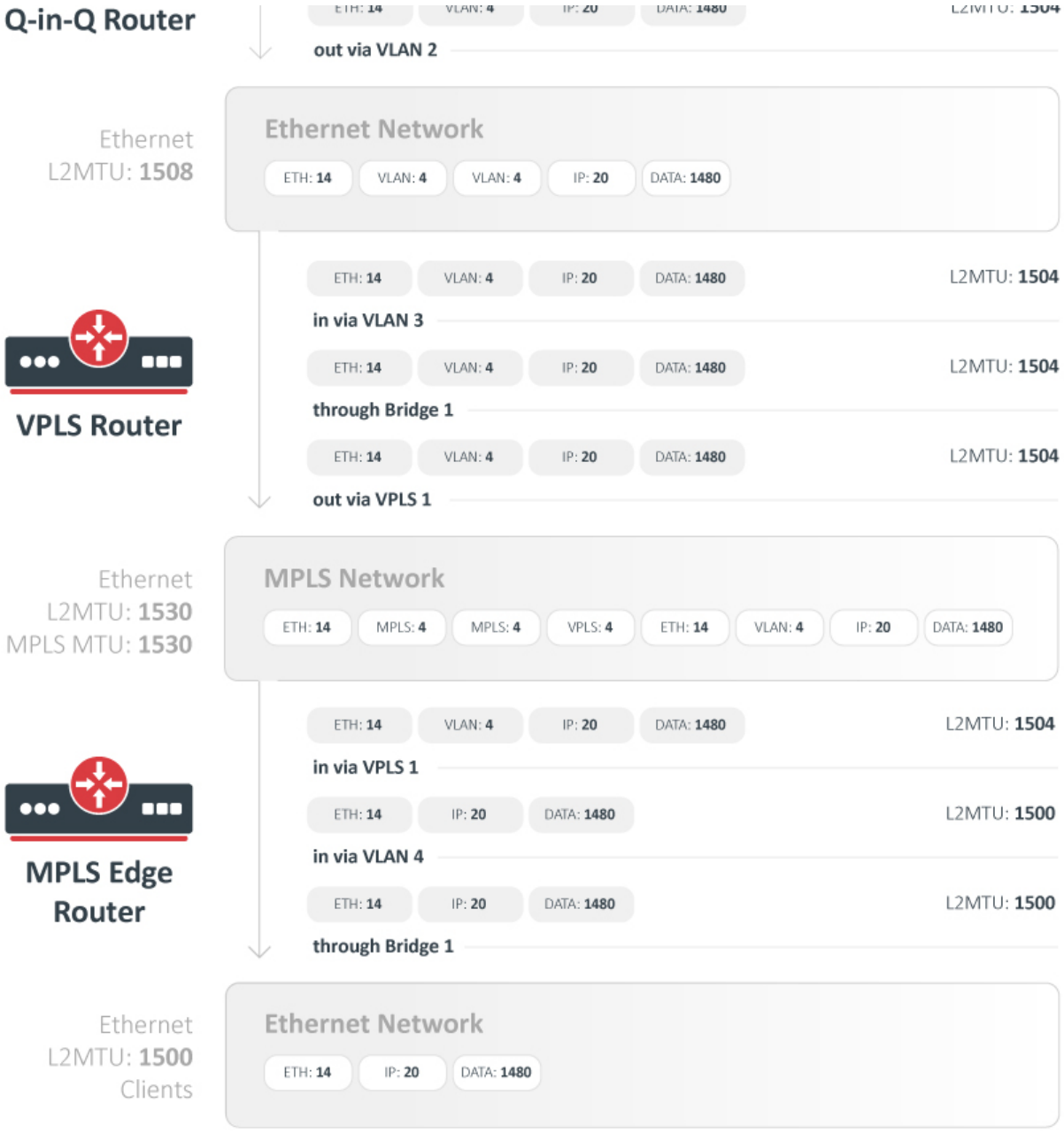
Advanced Setup Examples

In this example, we will take a closer look at the required L2MTU of all Ethernet-like interfaces including Bridge, VLAN, and VPLS interfaces.

In this setup we will have 3 routers:

- Q-in-Q router - this router will receive a standard 1500 byte Ethernet frame and will add two VLAN tags to the packet. Then packet will be sent out via an Ethernet network to the second router
- VPLS router - this router will remove the outer VLAN tag and will bridge the packet with the remaining VLAN tag with the VPLS tunnel. VPLS tunnel will take a packet through the MPLS network to the third router.
- MPLS Edge router - will remove VPLS and VLAN tags and bridge packet to the client Ethernet network.




Q-in-Q Router
out via VLAN 2
Ethernet
L2MTU: 1508
Ethernet Network
ETH: 14
VLAN: 4
VLAN: 4
IP: 20
DATA: 1480
ETH: 14
VLAN: 4
IP: 20
DATA: 1480
L2MTU: 1504
in via VLAN 3
ETH: 14
VLAN: 4
IP: 20
DATA: 1480
L2MTU: 1504
through Bridge 1
ETH: 14
VLAN: 4
IP: 20
DATA: 1480
L2MTU: 1504
VPLS Router
out via VPLS 1
Ethernet
L2MTU: 1530
MPLS MTU: 1530
MPLS Network
ETH: 14
MPLS: 4
MPLS: 4
VPLS: 4
ETH: 14
VLAN: 4
IP: 20
DATA: 1480
ETH: 14
VLAN: 4
IP: 20
DATA: 1480
L2MTU: 1504
in via VPLS 1
ETH: 14
IP: 20
DATA: 1480
L2MTU: 1500
in via VLAN 4
ETH: 14
IP: 20
DATA: 1480
L2MTU: 1500
MPLS Edge Router
through Bridge 1
Ethernet
L2MTU: 1500
Clients
Ethernet Network
ETH: 14
IP: 20
DATA: 1480

Peripherals

This article describes supported add-on peripherals for RouterBOARD hardware devices.

Cellular modems

RouterOS v6 and v7 supported cellular modems:

- MikroTik modems
- 3rd party modems supported by device class/type:
  - MBIM class USB interface
  - USB-CDC class USB interface
  - RNDIS type USB interface
- 3rd party modem with added support, see modem table below

Please note:

- not all modems are listed in the supported modem table, some may work because modem manufacturers re-use the same hardware IDs and vice versa
- customized, localized and locked units may have compatibility issues
- 3rd party modem may require a modem configuration adjustment before it can be used with RouterOS
- mini-PCIe modems with USB3.0 interface installed in mini-PCIe PCIe/USB2.0 enabled slot USB speed must be limited to USB2.0 speed or mini-PCIe shared PCIe/USB3.0 pins isolated. See the picture below table

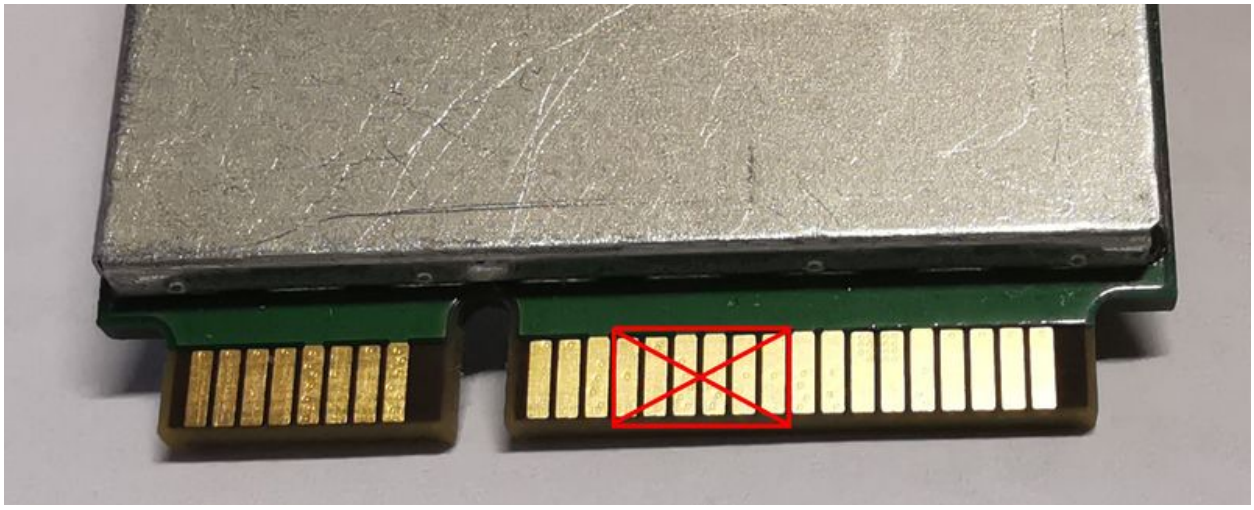
Model	vendor-id	device-id	Tested RouterOS version	Comments	Format
Alcatel IK40			v6.41RC11	Config-less LTE interface	USB
Alcatel IK41			v6.48	Config-less LTE interface	USB
Android usb tethering interface			v6.7	Config-less LTE interface	USB
AnyData ADU-E630WH			v6	(aka "USB Wireless HSDPA/UMTS 2.1GHz GSM/GPRS/EGPRS 900/17000MHz/CDMA 1x EVDO Rev.A")	USB
Anteniti 3372h-153			7.12		USB
BandRich C501			v5.25 and v6.0		USB
D-link DWM-157	0x2001	0x7d02	v6.xx	Works! Data Channel: 2, Info Channel: 3,Modem Init: AT+CFUN=1, vendor-id="0x2001" device-id="0x7d02" Some info from modem: > H/W Ver.: B1, F/W Ver.: 2.0.1eu, revision: +CGMR: MOLY.WR8.W1231.DC.WG.MP.V3, 2013/04/09 02:08 Different HW revisions might not work with RouterOS	USB
D-link DWM-222			v6.38	Multiple modem versions with the same marketing name exist, only H/W ver: A1 supported as config-less LTE interface	USB
Dell DW5821e			v7.4beta4 and higher	MBIM driver. Revision: T77W968.F1.0.0.5.2.VZ.013 044	M.2
Dell DW5821e-eSIM	0x413c	0x81e0	7.11 and higher	MBIM driver. FW: T77W968.F1.0.0.5.2.GC.013. "at-chat" support added	M.2
Dell Wireless 5530 HSPA			v6.1 and higher	Data channel 0, Info channel 0, init: AT+CFUN=1 (needs manualy change profile by command AT*ENAP=1,1)	MiniPCI-e

Ericsson F5521gw			v6.x and higher		MiniPCI-e
Fibocom FM150-AE	0x2cb7	0x0111	v7.1beta5	MBIM driver. Revision: 89603.1000.00.01.01.03	M.2
Fibocom FG621-EA	0x2cb7	0x0a07		MBIM LTE interface	mini-PCIe USB device
Fibocom NL-952-EAU			v7.1beta5	MBIM driver. Revision: 19600.7000.00.04.01.05	M.2
Marvell PXA1802 based modems	0x1286	0x4e31	v7.2.2		mini-PCIe
Huawei E153			v6.31< and higher		USB
Huawei E171			v6.xx	Works! ppp interface, vendorid=0x12d1 deviceid=0x140c	USB
Huawei e3131			v6.xx and higher	ppp interface	USB
Huawei E3372h, E5576h, E8372h	0x12d1	0x14db	v6.8	Config-less LTE interface. Models with suffixes -320 and -608 will not work with RouterOS v6, please use v7 instead. Various customized variants were observed which does not provide LTE interface, but can be used in serial/PPP mode.	USB
Huawei E3276-150			v6.xx	ppp interface	USB
Huawei E3351			v6.24 and higher		USB
Huawei E3531			v6.24 or 6.40 RC25	There are different versions of this modem E3531-6 works from version 6.40RC25 as ppp, mbim supported only from RouterOS V7	USB
Huawei e398			v6.xx and higher	ppp interface	USB
Huawei E5377			v6.36.1	MIFI unit. No serial support, but works with IP on LTE interface	USB
Huawei E5673s-609			v6.xx	LTE interface	USB
Huawei K5160			v6.37 v7.0beta6	v6 and v7 - config-less LTE interface v7 - by default will try to use the modem in MBIM mode	USB
Huawei K5161			v6.47	Config-less LTE interface	USB
Huawei ME909s-120			v6.28	Recommended modem firmware version 11.617.24.00.00 To reduce LTE interface IP subnet mask to /32 configure the modem with at-chat command: /interface lte at-chat [find] input="AT^CUSTFEATURE=3,1"	MiniPCI-e
Huawei ME909u-521			v6.11		MiniPCI-e
Huawei MU609			v6.11		MiniPCI-e
Huawei MU709s-2			v6.28		MiniPCI-e
Huawei MS2372h-517			v7.12beta3	serial/PPP interface	USB
Jaton MT421e			v6.40RC32	config-less LTE interface	MiniPCI-e
Netgear Unite Explore 815S			v6.41	MIFI unit. No serial support, but works with IP on LTE interface.	USB
Novatel USB730L			v6.41RC6	LTE interface	USB

Olivetti Olicard 500			v6.41RC11	ppp interface	USB
Quectel EC20 /EC21			v6.xx	ppp interface, there is page in wiki about Quectel: article	MiniPCI-e
Quectel BG77	0x2c7c	0x0700	v6.47	Serial/PPP interface, single AT/modem channel=2	OEM module
Quectel BG95-M3			v6.47	Serial/PPP interface, single AT/modem channel=2 Will not work in wAP R ac boards.	mini-PCIe
Quectel BG96			v6.45	Serial/PPP interface, 2x AT/modem channels	mini-PCIe
Quectel EC25-EU	0x2c7c	0x0125	v6.42	ppp/LTE interface, there is page in wiki about Quectel ppp mode: article RouterOS v6 CDC-ECM mode - LTE interface receive address in modems internal network. RouterOS v7 MBIM mode - LTE interface uses APN IP address.	MiniPCI-e
Quectel EG25-G	0x2c7c	0x0125	6.48.3	RouterOS v6 CDC-ECM mode - LTE interface receives address in modem internal network. RouterOS v7 MBIM mode - LTE interface uses APN IP address. In some boards may be required to disable SIM hot plug detection: `/interface lte set [find] modem-init="AT+QSIMDET=0,1"`	MiniPCI-e
Quectel EM12-G	0x2c7c	0x0512	v7.1beta5	MBIM LTE interface	m.2
Quectel EP06			v6.42	ppp/LTE interface, there is page in wiki about Quectel: article	MiniPCI-e
Quectel RM500Q-GL	0x2c7c	0x0800	v7.1beta6	MBIM LTE interface	m.2
Quectel RM500Q-AE	0x2c7c	0x0800	v7.1beta6	MBIM LTE interface	m.2
Quectel RM502Q-AE	0x2c7c	0x0800	v7.1beta5	MBIM LTE interface	m.2
Quectel RM510Q-GL	0x2c7c	0x0800	7.9	MBIM LTE interface	m.2
Quectel UC15			v6.xx	Works, ppp interface	MiniPCI-e
Quectel UC20			v6.xx	Works, ppp interface	MiniPCI-e
R11e-4G	0x2cd2	0x0003	v6.42	LTE interface. Supports multiple APN, LTE passthrough	MiniPCI-e
R11e-LTE6	0x2cd2	0x0004	v6.39.2	LTE interface. Supports multiple APN, LTEpassthrough	MiniPCI-e
R11e-LTE	0x2cd2	0x0001	v6.39.2	LTE interface. Supports multiple APN, LTEpassthrough	MiniPCI-e
Sierra Netgear AirCard 320U			6.41	Customer tested the modem with firmware 03.05.23	USB
Sierra wireless MC73xx			v6.xx(ppp) v7.xx (LTE)	Works! PPP interface. And starting with v7.xx it will support LTE interface. MC7304 tested with firmware SWI9X15C_05.05.67.00	MiniPCI-e
Sierra Wireless MC7430			v6.xx and higher	Data channel 2, Info channel 2, Modem init: AT+CGATT=0, Dial-command: AT+CGATT=1;D*99#, also needs USB 3.0 pins isolated	MiniPCI-e
Sierra Wireless MC74xx			v7.1	MBIM LTE interface	mini-PCIe
Sierra Wireless MC7455	0x1199	0x9071	v7.3beta37	MBIM LTE interface with extended support for USB compositions: • 1009 - diag,modem,mbim • 100D - diag,nmea,modem,mbim Regarding available modem USB compositions (device-id) please see modem manual, command AT!USBCOMP	mini-PCIe

Sierra Wireless MC7710 /MC7700 /MC7750			v5.25, v6.0 and 6.40 RC43	If modem uses firmware 3.5 it should be upgraded to 3.5.23.2 firmware release in order to work in RouterOS correctly again.	MiniPCI-e
SIMcom various models	0x1e0e	0x9006	v6.xx	PPP interface	Mini-PCIe USB device
SIMcom various models	0x1e0e	0x9000 0x9001 0x9003 0x901a 0x901b 0x9205	v6.xx	LTE interface Some modems may not work with RouterOS v7 on MMIPS-based RouterBoards (M11G, M33G, LtAP etc), please use RouterOS v6. Regarding available modem USB compositions (device-id) please see modem manual, commands AT+CUSBPIDSWITCH; AT+CUSBCFG	m.2/mini-PCIe USB device
SIMcom various models	0x1e0e	0x9003 0x9005	v7.xx	MBIM LTE interface Regarding available modem USB compositions (device-id) please see modem manual, commands AT+CUSBPIDSWITCH; AT+CUSBCFG	m.2/mini-PCIe USB device
SIMCom SIM8202G-M.2	0x1e0e	0x901e	v7.11	MBIM LTE interface Regarding available modem USB compositions (device-id) please see modem manual, command AT+CUSBCFG=usbid,1e0e,901e	m.2 USB device
SIMCom SIM8262	0x1e0e	0x901e	V7.14	MBIM LTE interface + AT-Chat support	
SXT LTE 3-7			v6	LTE interface. Old version of SXT LTE	m.2
Tele2.ru LTE-D402			v6.47	Config-less LTE interface	
Telecom NZ T-Stick ZTE MF-181			v6.0rc13	Data Channel=2, Info Channel=2, APN internet.telecom.co.nz, PHONE=*99#. Tested ok for both data and SMS on CCR1016-12G	USB
Telit FN980m			v7.5	AT#USBCFG=2	m.2
Telit LE910	0x1bc7	0x1201	v6.xx	ppp interface	MiniPCI-e
Telit LE910C1			v6.46	Non-configurable from RouterOS	MiniPCI-e
Telit LE910C4	0x1bc7	0x1204	v7.xx	Works in MBIM mode : LTE interface + AT-chat	MiniPCI-e
Telit LM940			v6.44	LTE interface In some cases needs 3.0 pins isolated	MiniPCI-e
Telit LM960			v6.46	LTE interface In some cases needs 3.0 pins isolated	MiniPCI-e
TPS (Turning Point Solution) GCT450			v6.48	Config-less LTE interface	MiniPCI-e
Vodafone (Huawei) K4203			v7.xx	Not supported in ROS v6, but as this modem supports MBIM drivers support will be possible in ROS v7.	USB
Vodafone K4201-Z			v6.8	Some settings are ignored. LTE interface.	USB
Vodafone K4305			v6.7	Some settings are ignored.	USB
Vodafone K5160			v6.37	Some settings are ignored.	USB

Yota LU150			v5.22 and v6.4	Some settings are ignored.	USB
Yota wifi modem			v6.7	Some settings are ignored.	USB
Yota WLTUBA-107			v6.0	Some settings are ignored.	USB
ZTE 821D			v6.x	Set Info channel = 1, Data channel = 3, Dial command=ATDT	USB
ZTE AC5730			v6.x		USB
ZTE ME3630-E			v6.40RC26	ppp and LTE interface	MiniPCI-e
ZTE MF110			v6.28 and higher	Set info channel = 2, data channel = 2, Dial command=ATM1L3DT	USB
ZTE MF823			v6.8	Some settings are ignored. For some devices it's needed to enter in FACTORY mode to change operating state.	USB
ZTE MF825A			v6.xx	Some settings are ignored.	USB
ZTE MF827			v6.8	Some settings are ignored.	USB
ZTE MF832S			v7.10	Config-less LTE interface may require to set some settings using at-chat or modem init string	USB
ZTE MF90			v6.44beta32 and higher	LTE interface	USB



mini-PCIe slot shared PCIe/USB3.0 pins

SFP modules

Brand	Model	Rate	Connector /Cable type	Wavelength	Tested with	Works / Doesn't
MikroTik	S-85DLC05D	1,25G	Dual LC, MM	850nm	*Check: SFP/SFP+ compatibility reference table	Natively supported
MikroTik	S-31DLC20D	1,25G	Dual LC, SM	1310nm	*Check: SFP/SFP+ compatibility reference table	Natively supported
MikroTik	S-35LC20D	1,25G	BiDi LC, SM	Tx:1310nm/Rx: 1550nm	*Check: SFP/SFP+ compatibility reference table	Natively supported
MikroTik	S-53LC20D	1,25G	BiDi LC, SM	Tx:1550nm/Rx: 1310nm	*Check: SFP/SFP+ compatibility reference table	Natively supported

MikroTik	S-RJ01	1000/100/10	RJ45, Cat5/Cat6	N/A	*Check: SFP/SFP+ compatibility reference table	Natively supported
Axiom	AXG91632	1000BASE-LX	Dual LC	1310nm	CRS125-24G-1S-RM	Works!
Finisar	FCLF-8521-3	10/100/1000	RJ45, Cat6	N/A	RB2011LS-IN	Works!
Finisar	FCLF-8521-3-MD	10/100/1000	RJ45, Cat6	N/A	RB2011LS-IN	Works!
Finisar	FTRJ8519P1BNL-B1	10/100/1000 1.25 Gb/s 1000Base-SX Ethernet	Dual LC, MM	850nm	RB2011LS-IN	Works!
Finisar	FTLF8519P2BNL	10/100/1000 1.25 Gb/s 1000Base-SX Ethernet	Dual LC, MM	850nm	RB2011LS-IN	Works!
Finisar	FTRJ1319P1BTL	1.25Gb/s 1000Base-LX Ethernet	Dual LC, SM	1310nm	CCR1009-8G-1S-1S+ and CCR1009-7G-1C-1S+	Works!
Unica	SFP-1.25G-T	1000M	RJ45, Cat6	N/A	RB2011LS-IN	Works!
Dell	FTLX8571D3BCL	1,25G	Dual LC, MM	850nm	RB2011LS-IN	Works!
Unica	GP-3124-L2CD-C	1,25G	Dual LC, MM	1310nm	RB2011LS-IN	Works!
Cisco	GLC-T	1.25G	RJ45, Cat6	N/A	RB2011LS-IN	Works!
Cisco	GLC-SX-MM	1000BASE-SX SFP transceiver module for MMF, 1.25G	Dual LC, MM	850nm	RB2011LS-IN	Works!
Cisco	SFP-GE-L	1000BASE-LX/LH SFP transceiver module for SMF, 1.25G	Dual LC, SM	1300nm	Various MT hardware	Works!
6COM	6C-SFP-T	10/100/1000	RJ45, Cat6	N/A	RB2011LS-IN	Works!
6COM	6C-WDM-0210BSD	1,25G	BiDi SC, SM	Tx:1550nm/Rx:1310nm	RB2011LS-IN	Works!
6COM	6C-WDM-0210ASD	1,25G	BiDi SC, SM	Tx:1310nm/Rx:1550nm	RB2011LS-IN	Works!
6COM	6C-SFP-0310D	1,25G	Dual LC, MM	1310nm	RB2011LS-IN	Works!
6COM	6C-SFP-0301D	1,25G	Dual LC, MM	850nm	RB2011LS-IN	Works!
Ingellen	INSP-T(10/100/1000)	10/100/1000	RJ45, Cat6	N/A	RB2011LS-IN	Works!
Ingellen	INSPL-53-BX	1,25G	BiDi LC, MM	1550/1310	RB2011LS-IN	Works!
Ingellen	INSPL-35-BX	1,25G	BiDi LC, MM	1310/1550	RB2011LS-IN	Works!
Ingellen	INSP-LX-SM	1,25G	Dual LC, SM	1310nm	RB2011LS-IN	Works!
Ingellen	INSP-SX-MM	1,25G	Dual LC, MM	850nm	RB2011LS-IN	Works!
AXCEN	AXGT-R1T4-05I1	10/100/1000	RJ45, Cat6	N/A	RB2011LS-IN	Works!
AXCEN	AXGD-37A4-0531	1,25G	BiDi LC, MM	Tx:1550nm/Rx:1310nm	RB2011LS-IN	Works!
AXCEN	AXGD-16A4-0531	1,25G	BiDi LC, MM	Tx:1310nm/Rx:1550nm	RB2011LS-IN	Works!
AXCEN	AXGD-1354-0531	1,25G	Dual LC, MM	1310nm	RB2011LS-IN	Works!
AXCEN	AXGD-5854-0511	1,25G	Dual LC, MM	850nm	RB2011LS-IN	Works!
TP-Link	TL-SM311LS	1,25G	Dual LC, SM	1310nm	RB2011LS-IN	Works!
TP-Link	TL-SM311LM	1,25G	Dual LC, MM	850nm	CCR1036 12G-4S	Works!
OPTIC	OPTIC-SFP-3524S-02-SC	1,25G	BiDi SC, SM	Tx:1310nm/Rx:1550nm	RB2011UAS-RM, RB260GS	Works!
OPTIC	OPTIC-SFP-5324S-02-SC	1,25G	BiDi SC, SM	Tx:1550nm/Rx:1310nm	RB2011UAS-RM, RB260GS	Works!
OPTIC	OPTIC-SFP-S1203-L3302-LC	1,25G	BiDi LC, SM	Tx:1310nm/Rx:1550nm	RB2011UAS-RM, RB260GS	Works!

OPTIC	OPTIC-SFP-S1205-L3302-LC	1,25G	BiDi LC, SM	Tx:1550nm/Rx:1310nm	RB2011UAS-RM, RB260GS	Works!
ROBOFiber	SFP-7120-55	1,25G	Dual LC, SM	1550nm	CCR1036-12G-4S, RB2011	Works!
ROBOFiber	SFP-7120-WA	1,25G	BiDi LC, MM	Tx:1490nm/Rx:1550nm	CCR, RB2011	Works!
ROBOFiber	SFP-7120-WB	1,25G	BiDi LC, MM	Tx:1550nm/Rx:1490nm	CCR, RB2011	Works!
Enguity	SFP-3647603KM.b1310 XT	1,25G	BiDi LC, SM	Tx:1310nm/Rx:1550nm	CCR, RB2011, RB260GS	Works!
Enguity	SFP-3647603KM.b1550 XT	1,25G	BiDi LC, SM	Tx:1550nm/Rx:1310nm	CCR, RB2011, RB260GS	Works!
Enguity	SFP-3647610KM.b1490 XT	1,25G	BiDi LC, SM	Tx:1490nm/Rx:1550nm	CCR, RB2011, RB260GS	Works!
Enguity	SFP-3647610KM.b1550 XT	1,25G	BiDi LC, SM	Tx:1550nm/Rx:1490nm	CCR, RB2011, RB260GS	Works!
AdvOptics MSA	GLC-SX-MM	1,25G	BiDi LC, MM	Tx:1310nm/Rx:1310nm	CCR, RB2011, RB260GS	Works!
AdvOptics MSA	GLC-ZX-SM	1,25G	BiDi LC, SM	Tx:1310nm/Rx:1310nm	CCR, RB2011, RB260GS	Works!
Proline	GLC-BX-D20-PRO	1,25G	BiDi LC, SM	Tx:1490nm/Rx:1310nm	CRS125	Works!
Proline	GLC-BX-D40-PRO	1,25G	BiDi LC, SM	Tx:1310nm/Rx:1490nm	CRS125	Works!
Foundry Networks	E1MG-BXU-AC	1,25G	BiDi LC, SM	Tx:1310nm/Rx:1490nm	RB3011UiAS, hAP ac	Works!
Avago	SFBR-5799APZ	1,25G	Dual LC, MM	850nm	CRS326, CRS112	Works in 1Gbps mode!
Eltex	NTU-SFP-100	1,25G	SC	N/A	RB4011iGS+	Works!

SFP+ modules

Brand	Model	Distance	Rate	Connector /Cable type	Wavelength	Tested with	Works / Doesn't
MikroTik	S+85DLC03D	300m	10G	Dual LC, MM	850nm	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+31DLC10D	10km	10G	Dual LC, SM	1310nm	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+23LC10D	10km	10G	BiDi LC, SM	Tx:1270nm/Rx:1330nm	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+32LC10D	10km	10G	BiDi LC, SM	Tx:1330nm/Rx:1270nm	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+DA0001	1m	10G	Twinax Copper	N/A	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+DA0003	3m	10G	Twinax Copper	N/A	All MikroTik products with SFP /SFP+ interfaces	Natively supported
MikroTik	S+RJ10	various, depending on link rate. Check brochure for more details	10G/5G/2.5G/1G/100M/10M	RJ45 - Cat5E /Cat6/Cat7	N/A	All MikroTik products with SFP+ interfaces	Natively supported
Atop	APSP55B30CDL40	40km	10G	Dual LC, SM	1550nm	CRS series, CCR series devices with SFP+ interfaces	Does NOT work!
Cisco	SFP-10G-LR	10km	10G	Dual LC, SM	1310nm	RB2011LS-IN	Works!
Dell (Finisar)	FTLX8571D3BCL	300m	10G	Dual LC, MM	850nm	Most of SFP/SFP+ MikroTik products	Works!
Juniper (Finisar)	FTLX8571D3BCL-J1	300m	10G	Dual LC, MM	850nm	Most of SFP/SFP+ MikroTik products	Works!

Intel (Finisar)	FTLX8571D3BCV-IT	300m	10G	Dual LC, MM	850nm	Most of SFP/SFP+ MikroTik products	Works!
OEM (Juniper?)	EX-SFP-10GE-SR-OEM	300m	10G	Dual LC, MM	850nm	Most of SFP/SFP+ MikroTik products	Works!
Fiberstore	SFP-10G31-40	40km	10G	Dual LC, SM	1310nm	CRS series, CCR series devices with SFP+ interfaces	Works!
Fiberstore	SFP-10G55-40	40km	10G	Dual LC, SM	1310nm	CRS series, CCR series devices with SFP+ interfaces	Works!
Fiberstore	SFP-10G32-40	40km	10G	BiDi LC, SM	Tx:1330nm/Rx:1270nm	CRS series, CCR series devices with SFP+ interfaces	Works!
Fiberstore	SFP-10G23-40	40km	10G	BiDi LC, SM	Tx:1270nm/Rx:1330nm	CRS series, CCR series devices with SFP+ interfaces	Works!
Optech	OPAK-TX1-00-C	30m	10G	RJ45 - Cat 6a /7 Cable	N/A	CCR, CCR, CSS series devices with SFP+ interfaces	Works, starting with v6.40rc20 RouterOS build.
ProLabs	SFP-10G-T-C	30m	10G	RJ45 - Cat 6a /7 Cable	N/A	CCR, CCR, CSS series devices with SFP+ interfaces	Works, starting with v6.40rc20 RouterOS build.

PoE-Out

Summary

Sub-menu: `/interface ethernet poe`

This page describes how PoE-Out (Power over Ethernet) feature can be used on MikroTik devices with at least one PoE-Out interface. MikroTik uses RJ45 mode B pinout for power distribution, where the PoE is passed trough pins 4,5 (+) and 7,8 (-). If a device supports powering other devices using PoE-out, then it is recommended to use **at least 18V** as the input voltage, except for devices that support multiple output voltages (e.g. CRS112-8P-4S-IN, CRS328-24P-4S+RM, CRS354-48P-4S+2Q+RM).

MikroTik supported PoE-Out standards

MikroTik devices can support some or all of the following PoE standards:

- **Passive PoE-Out up to 30 V** - PoE standard, which does not require negotiation between PSE (Power Sourcing Equipment) and PD (Powered Device). PoE-out uses the same voltage as supplied to the PSE (Power Sourcing Equipment). PoE-Out Standard for devices that supports input voltage up to 30 V. PD resistance should have ranged from 3kΩ to 26.5kΩ. (e.g. hEX PoE lite, RB3011UiAS-RM, RB2011iL-IN.)

- **Passive PoE-Out up to 57 V** - Works the same as low voltage (up to 30 V) PoE-Out, but is also capable to deliver high voltage over PoE ports. The output voltage depends on the power source connected to PSE. Can power up af/at compatible devices, which accepts power over 4,5 (+) and 7,8 (-), and does not require PoE negotiation. PD resistance should have ranged from 3kΩ to 26.5kΩ. (e.g. cAP ac, hAP ac, wsAP ac lite.)
- **IEEE Standards 802.3af/at** - Also called PoE Type 1/PoE+ Type 2, are PoE standards Defined by the IEEE. The aim of these standards is to reduce incompatibility between vendors. MikroTik PSE with af/at support is capable of powering both a Type 1 and a Type 2 PD. Valid PD should have PoE-In resistance from 23.75kΩ to 26.25kΩ. MikroTik devices that support af/at standard can also switch to Passive PoE-Out mode. (e.g. CRS112-8P-4S-IN, CRS328-24P-4S+RM, CRS354-48P-4S+2Q+RM.)

Each PoE-Out implementation supports overload and short-circuits detection.

> **Note:** Some MikroTik devices support all of the described standards (e.g. CRS112-8P-4S-IN, CRS328-24P-4S+RM, netPower 16P, CRS354-48P-4S+2Q+RM etc...)

How to choose your PoE PSE

This table can help you choose which PSE device is best suitable for your needs.

Device name	PoE-Out port count	Passive PoE	802.3af/at	802.3bt	Power input	Maximum output per port		Maximum power output, W
						Input 18-30V, mA	Input 30-57V, mA	
CSS610-8P-2S+IN	8	+	+	-	AC &DC 48-57 V	1000	625	140
CRS328-24P-4S+RM	24	+	+	-	AC	1000	450	450
CRS354-48P-4S+2Q+RM	48	+	+	-	AC	1000	570	700
CRS112-8P-4S-IN	8	+	+	-	DC 18-30V & DC 30-57V	1000	450	80
netPower 16P	16	+	+	-	DC 18-30V & DC 30-57V	1100	600	160
RB5009UPr+S+IN	8	+	+	-	DC 18-30V or DC 30-57V	640	420	130
hEX PoE	4	+	+	-	DC 18-30V or DC 30-57V	1000	450	102
PowerBox Pro	4	+	+	-	DC 18-30V or DC 30-57V	1000	450	102
OmniTIK 5 PoE ac	4	+	+	-	DC 18-30V or DC 30-57V	1000	450	102
hEX PoE lite	4	+	-	-	DC 18-30V	1000	-	60
PowerBox	4	+	-	-	DC 18-30V	1000		60
RB260GSP	4	+	-	-	DC 18-30V	1000		60
OmniTIK 5 PoE	4	+	-	-	DC 18-30V	1000		60

PoE-Out Configuration

PoE Configuration is supported on all MikroTik devices with PoE-Out interfaces, the configurations can be edited from the RouterOS and SwOS interfaces.

RouterOS

Usage

RouterOS provides an option to configure PoE-Out over Winbox, Webfig, and CLI, basic commands using the CLI are

Property	Description
print ()	Prints PoE-Out related settings.
export ()	export is displayed under `/interface ethernet` menu.
monitor (*string*| *interface*)	Shows poe-out-status of a specified port, or all ports with `/interface ethernet poe monitor [find]` command.

power-cycle (*duration:0..1m |*; Default: **5s**)	Disables PoE-Out power for a specified period of time.

Global Settings

Some MikroTik PoE-Out devices support the global PoE setting which can be configured under `/interface ethernet poe settings` menu. Global setting ether1-poe-in-long-cable feature disables strict input/output current monitoring (short detection) to allow the use of PoE-Out with long ethernet cables and/or avoiding improper short-circuit detection.

Property	Description
ether1-poe-in-long-cable (*yes | no*)	Setting it to "yes" will disable short detection on all poe-out ports. This is potentially dangerous settings and should be used with caution

Note: Global setting of ***ether1-poe-in-long-cable*** can also affect PoE-Out behavior on PSE which is powered using a DC connector

Port Settings

PoE-Out can be configured under the menu. Each port can be controlled independently.

Property	Description
name ()	Name of an interface
poe-out (*auto-on | forced-on | off*; Default: **auto-on**)	Specifies PoE-Out state • auto-on - the board will attempt to detect if power can be applied to the port. For powering there should be resistance in the range from 3kΩ to 26.5kΩ • forced-on - detection range is removed. As a result power over Ethernet will be always on • off - all detection and power is turned off for this port
poe-priority (*integer: 0..99 | any*; Default: **10**)	poe-priority specifies the importance of PoE-Out ports, in cases when a total PoE-Out limit is reached, interface with the lowest port priority will be powered off first. Highest priority is 0, the lowest priority is 99. If there are 2 or more ports with the same priority then port with the smallest port number will have a higher priority. For example, if ether2 and ether3 have the same priority and over-current is detected then PoE-Out on ether3 will be turned off. Every 6 seconds ports will be checked for a possibility to provide PoE-Out if it was turned off due to port priority.
poe-voltage (*auto | low | high*; Default: **auto**)	A feature that allows us to manually switch between two voltage outputs on PoE-Out ports. It will take effect only on PSE with switchable voltage modes (CRS112-8P-4S-IN, CRS328-24P-4S+RM, netPower 16P, CRS354-48P-4S+2Q+RM).
poe-lldp-enabled *(yes / no;* Default: **no**)	Link Layer Discovery Protocol (LLDP) is a layer-2 Ethernet protocol for managing devices. LLDP allows an exchange of information between a PSE and a PD. Starting from RouterOS version 7.15, the setting has been replaced with the Neighbor Discovery `lldp-poe-power` property.

Note: Enabling poe-lldp in RouterOS 7.8 can potentially solve issues encountered in VoIP setups.

Note: If poe-voltage=auto and poe-out is set to "forced-on", LOW voltage will be used by default. If the PD supports only high voltage, make sure you also set poe-voltage=high when forcing the PoE output.

Power-cycle settings

RouterOS provides a possibility to monitor PD using a ping, and power-cycle a PoE-Out port when the host does not respond. power-cycle-ping feature can be enabled under `/interface ethernet poe` menu.

Property	Description
power-cycle-ping-enabled (*yes* | *no*; Default: **no**)	Enables ping watchdog, power-cycles port if a host does not respond to ICMP or MAC-Telnet packets.
power-cycle-ping-address (*IPv4* | *IPv6* | *MAC*; Default:)	An address which will be monitored. Since RouterOS 6.46beta16, an active route towards PD is required in case an IP address is configured, so make sure PSE can reach the PD. In case the MAC address is specified, PSE will send MAC-Telnet ping requests only from a specified ethernet interface. When configuring a bridge vlan-filtering or some way of VLAN switching, it is recommended to use the IP address for monitoring your PD.
power-cycle-ping-timeout (*time:0..1h* |; Default: **5s**)	If the host does not respond for more than <timeout> period of time, then PoE-Out port is switched off for 5s.
power-cycle-interval (*time*| *any*; Default:)	Disables PoE-Out power for 5s between the specified intervals. Not related with the power-cycle-ping feature.

If power-cycle is enabled, `/interface ethernet poe monitor` will show the actual status of the host and time when power cycle will be performed [1]

SwOS

SwOS interface provides basic PoE-Out configuration and monitoring options, see more details in the SwOS PoE user manual.

PoE-Out Monitoring

RouterOS

MikroTik devices with PoE-Out controller (not injector) provides port monitoring option. `/interface ethernet poe monitor [find]`

Property	Description
name ()	Name of an interface
poe-out ()	Shows PoE-Out settings

poe-out-status ()	Shows current PoE-Out status on port • powered-on - Power is applied to the port, and PoE-Out is operating normally, • waiting-for-load - PSE attempts to detect if power can be applied to the port. For powering there should be resistance in the range from 3kΩ to 26.5kΩ; • short-circuit - Short-circuit is detected on PoE-Out port, power is switched off, the only detection with low voltage takes place. • overload - The PoE-Out current limit is exceeded, power is switched off on PoE-Out port. For port limits see each model specifications. • voltage-too-low - PD can not be powered with the voltage provided from PSE. • voltage_too_high - PSE controller cannot power PD with high voltage; • current-too-low - current-too-low means that PD draws too low current (<10mA) than normal PoE-Out device should, the reason for this can be: The delivered voltage at PD is too low for normal powering (for example Vmin =>30V, but provided 24V); PD uses a second power source which has a higher voltage than PSE, so all current is taken from the second DC source, not PSE PoE-Out port; • off - all detection and power is turned off for this port; • power_reset - PSE controller resetting the power, for example, when executing the power cycle command or when pings fail (power-cycle-ping); • controller_init - PSE controller initialization; • controller_upgrade - PSE controller is being upgraded; • controller_error - PSE controller does not respond.
poe-out-voltage ()	Displays PoE Voltage which is applied to the PD.
poe-out-current ()	Displays port current (mA) which is drawn by the PD.
poe-out-power ()	Displays PD power consumption

If `power-cycle-ping` feature is used, `/interface ethernet poe monitor [find]` will show additional fields:

power-cycle-host-alive: <YES/NO> (Shows if monitored host is reachable)
power-cycle-after:<TIME> (Shows time, after which the port will be power-cycled)

SNMP

It is possible to monitor PoE-Out values using SNMP protocol, this requires enabled SNMP on PSE. SNMP Wiki

SNMP OID tables:

- 1.3.6.1.4.1.14988.1.1.15.1.1.1 - interface-id
- 1.3.6.1.4.1.14988.1.1.15.1.1.2 - interface names
- 1.3.6.1.4.1.14988.1.1.15.1.1.4 - voltage in dV (decivolt)
- 1.3.6.1.4.1.14988.1.1.15.1.1.5 - current in mA
- 1.3.6.1.4.1.14988.1.1.15.1.1.6 - power usage in dW (deviwatt)

SNMP values can be requested also from the RouterOS, for example, `snmp-walk` will print current mA from all available PoE-Out ports:

/tool snmp-walk address=10.155.149.252 oid=1.3.6.1.4.1.14988.1.1.15.1.1.5

To get very specific OID value, use `snmp-get` tool (displays current mA on ether3 interface):

tool snmp-get address=10.155.149.252 oid=1.3.6.1.4.1.14988.1.1.15.1.1.5.3

PoE-Out notifications

PoE-Out LEDs

Models with dependant voltage output

PoE-Out LED behavior can differ between models, but most of them will indicate PoE-Out state on one additional LED. Devices with one voltage output will light:

- Red color LED - PoE-Out port state is **powered-on** (auto or forced-on mode).
- Blinking Red color LED - PoE-Out port state is **short-circuit**

Models with selectable voltage output

Models with multiple voltage options can indicate additional information:

- Green color triangle LED - PoE-Out port state is **powered-on** (auto or forced-on mode), PD uses low voltage.
- Red color triangle LED - PoE-Out port state is **powered-on** (auto or forced-on mode), PD uses high voltage (af/at or passive).
- Blinking Green color triangle LED - PoE-Out port state (low voltage) is **short-circuit** or **overload**
- Blinking Red color triangle LED - PoE-Out port state (high voltage) is **short-circuit** or **overload**

Model-specific LED behavior

- CRS112-8P-4S-IN - All PoE LEDs flashing: wrong voltage PSU plugged into one of the ports.
- netPower 16P - All PoE LEDs flashing: wrong voltage PSU plugged into one of the ports.
- CRS328-24P-4S+RM - indicates an exceeded overall max PoE output limit. Port PoE-Out priorities will work in 3 independent sections (8 ports each) and overload will happen in any section that breaches 150W consumption.

PoE-Out Logs

By default PoE-Out, event logging is enabled and uses "warning" and "info" topics to notify the user about PoE-Out state changes. Log entries will be added to each PoE-Out state change. Important logs will be added with a "warning" topic, informative logs will be added with the "info" topic. When PoE LLDP is enabled, LLDP status updates are available in the device logs, for example:

```
06:56:50 poe-out,debug ether4 LLDP TLV 25.0W request denied : hw-limit
```

Possible denial reasons:

- budget - requested power exceeds the total PSE budget.
- hw-limit - requested power is more than hardware supports (PSU affects this).
- low-voltage - LLDP request made to low-voltage port.
- off - port is shut down.
- class-limit - LLDP requires more than the class can provide.
- cmd-failed - RouterOS could not make a request to the controller.

To avoid unnecessary logging in cases when PD is not powered because of current-too-low, RouterOS will filter such events, and add one log per every 512 current-too-low events.

Logs can be disabled if necessary:

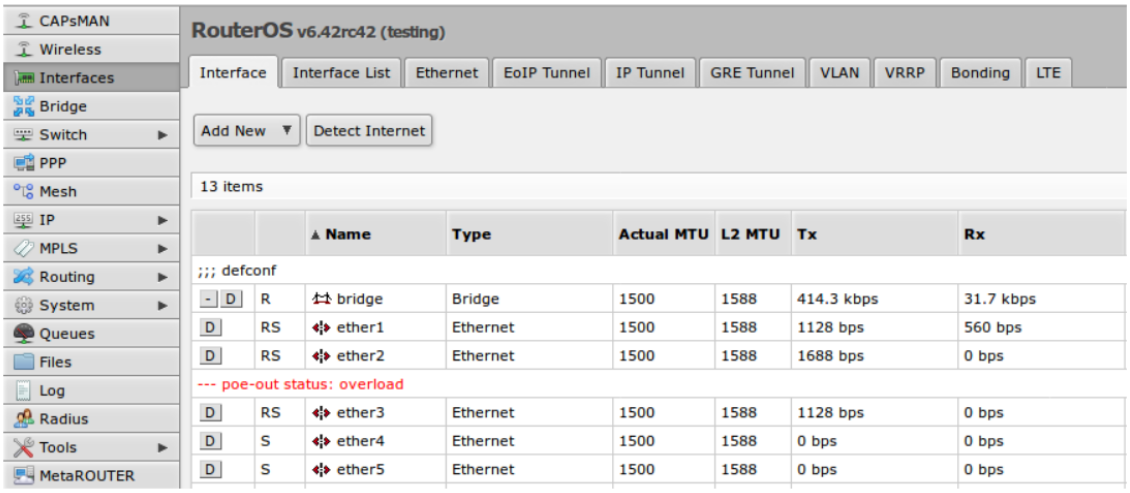
```
/system logging set [find topics~"info"] topics=info,!poe-out
/system logging set [find topics~"warning"] topics=warning,!poe-out
```

PoE-Out Warnings in GUI/CLI

To notify a user about important PoE-Out related problems, messages will be shown in Winbox / WebFig and CLI interface fields:

```
1 RS ;;; poe-out status: overload
ether1 ether 1500 1588 9204 64:D1:54:61:D5:E0
```

WebFig and Winbox will notify user under interfaces:



How it works

PoE-Out Modes

auto-on mode

If auto-on is selected on PoE-Out interface, then port operates in this strict order:

- PSE with low voltage checks for resistance on the connected port. If the detected resistance range is between (3kΩ to 26.5kΩ) power is turned on;
- When power is applied, the PSE continuously checks if the overload limit is not reached or short circuit detected
- If the cable is unplugged, the port returns in detection state and will remain off until suitable PD is detected

forced-on mode

If forced-on is selected then port operates in this strict order:

- PSE disables resistance check on the port, and apply power on pins 4,5 (+) and 7,8 (-), even if no cable is attached
- When power is applied, PSE still continuously checks if an overload or short circuit is not detected
- After the cable is unplugged, the power still remains enabled on the port.

off mode

If off mode is used, PoE-Out on the port will be turned off, no detection will take place, and the interface will behave like a simple Ethernet port.

PoE-Out limitations

It is important to check PoE-Out specification to find out hardware limitations because it can differ between models

PoE-Out port limitation

PoE-Out ports are limited with max amp values which are supported in particular voltage, usually max current will differ for low voltage devices (up to 30 V), and for high voltage devices (31 to 57 V).

PoE-Out total limitation

PSE has also a total PoE-Out current limitation which can't be exceeded, even if the individual port limit allows it.

PoE Out polarity

All MikroTik PSE uses the same PoE-Out pin polarity Mode B4,5 (+) and 7,8 (-), however other vendors can use opposite or Mode A pinout on PD. Reverse polarity would require using a crossover cable but Mode A PD would require Mode B to Mode A converter.

> **Note:** Passive PD with high input inrush current can result in overcurrent protection on PSE, make sure that PD specification supports powering from PSE (not only from the passive power injector)

Safety

PSE has the following safety features:

PoE-Out compatibility detection

The auto-on mode is considered safe, it will check if the resistance on the port is within allowed range and only then enable PoE out on the interface. The range is 3kΩ to 26.5kΩ

Overload protection

When a PoE-Out port is powered-on, it is constantly checked for overload. If the overload is detected, PoE-Out is turned off on the port to avoid damage to the PD or PSE.

In seconds the PoE Out feature will be turned on again to see if the environment has changed and PD can be supplied with power again. That is important for configurations that are not connected to mains (solar installations, equipment running on batteries due to mains failure) so that when voltage drops - overload will be detected and connected devices turned off. After a while when the voltage level returns to usual operating value - connected equipment can be powered up again.

Short circuit detection

When power is enabled on PoE-Out port, PSE continuously checks for a short circuit. If it is detected to ensure that there is no additional damage to PD and PSE, the power is turned off on all ports. PSE will continue to check PoE-Out port until the environment returns to normal.

> **Warning:** Make sure that non-standard incompatible PD which does not have the resistance range 3kΩ to 26.5kΩ are not attached, so the PSE would not try to apply power on them

Model-specific features

PSE with independent 8-port sections (CRS112-8P-4S-IN, CRS328-24P-4S+RM, netPower 16P, CRS354-48P-4S+2Q+RM) allows PoE-Out to work independently from the RouterOS, this means that you can reboot/upgrade your RouterOS and the PD will not be rebooted.

> Note: CRS328-24P-4S+, netPower 16P Poe-Out priorities work independently on each 8 port section!

PoE Out examples

RouterOS allows us to define priorities on PoE-Out ports, so if your installation is going overpower budget, the PSE will disable less important PD with the lowest priority.

The priority of *0* is the highest priority, *99* - lowest

Setting up priority

Example of how to set priorities from CLI:

```
/interface ethernet poe set ether2 poe-priority=10
/interface ethernet poe set ether3 poe-priority=13
/interface ethernet poe set ether4 poe-priority=11
/interface ethernet poe set ether5 poe-priority=14
```

What will happen when power budget will go over total PoE-Out limit - first if the overload is detected, ether5 will be turned off (lowest priority), then recheck is done and if the still total limit overload is detected next port in priority will be turned off, in this example, ether3 will be turned off. Both of these ports will be reached every few seconds to check if it is possible to turn PoE-Out on for these ports. Power up will happen in reverse order as the power was cut.

Same priority

if all, or some ports will have the same poe-priority, then port with the lowest port number will have higher priority

```
/interface ethernet poe set ether2 poe-priority=10
/interface ethernet poe set ether3 poe-priority=10
/interface ethernet poe set ether4 poe-priority=10
/interface ethernet poe set ether5 poe-priority=10
```

In this example, if the total PoE-Out limit is reached ether5 will be turned off first, then ether4 then ether3 as all of these ports have same poe priority.

Monitoring PoE-Out

PoE-Out ports can be monitored using a command `/interface ethernet poe monitor <interface>`

```
[admin@MikroTik] > interface ethernet poe monitor [find]
name: ether2 ether3 ether4 ether5
poe-out-voltage: 23.2V 23.2V 23.2V
poe-out-current: 224mA 116mA 64mA
poe-out-power: 5.1W 2.6W 1.4W
```

Power-cycle ping

Monitor connected PD with power-cycle-ping feature:

```
/interface ethernet poe set ether1 power-cycle-ping-enabled=yes power-cycle-ping-address=192.168.88.10 power-cycle-ping-timeout=30s
```

In this example, PD attached to ether1 will be continuously monitored using a power-cycle-ping feature, which will send ICMP ping requests and wait for a reply. If PD with IP address 192.168.88.10 will not respond for more than 30s, the PoE-Out port will be switched off for 5s.

Troubleshooting

In cases where a PD does not power-up or reboots unexpectedly when powered from your PSE, it's suggested to the first check:

- **PD supported input voltage** - PSE output voltage must be in the range supported by the PD. Otherwise, the PD is incompatible with the PSE, and will not be able to power-up. Check the PD datasheet.
- **PD supported input PoE-in standard** - Some PDs do not support af/at standard even if it has PoE-in support up to 57 V, check PD datasheet.
- **PD is rebooted from PSE**
  - Check if PD does not exceed PoE-Out port limit and Total-PoE-Out port limit of the PSE, check PSE datasheet.

    - Check if the Voltage limit does not drop bellow supported (Can be caused by voltage drop on the wires).
    - Check if you are using a proper power supply, the output power of PSU should be calculated from:
      ```
      (MAX power consumption of PSE) + (MAX power consumption of all PD) + 10%)
      ```
    - Check if you are using good quality ethernet cables, it's important especially in cases if PoE is used.
- **Check RouterOS version** - it's possible, that some PoE related features will be updated with RouterOS, make sure that you are running the latest RouterOS version.
- **PD Does not power up**
    - There can be cases where a PD does not power up, even though it supports passive PoE, and does not consume more power than the specified PSE port limit. This can be caused by inrush current triggering overcurrent protection on the PSE. Make sure that PD specification supports powering from PSE (not only from passive power injector)
    - Polarity - Devices with opposite or different pinouts can be unable to powerup from all PSE. Check the PD datasheet.
    - Incompatible resistance - PD resistance should have ranged from 3kΩ to 26.5kΩ (For Passive-PoE) and from 23.75kΩ to 26.25kΩ on af /at.

Legacy

PoE-Out Controller upgrade

PoE-Out devices which are running RouterOS 5.x can also hold old PoE-Out controller firmware, upgrade to RouterOS 6.x will automatically update the PoE-Out firmware. Changes between 1.x and 2.x PoE-Out controller firmware will result in higher Max-port limits (0.5A to 1A) in case if it's supported by the hardware, also will provide some additional data which can be monitored, and allow to use PoE-Out priorities.

All MikroTik devices which come with RouterOS 6.x already support the latest PoE-Out firmware.

Ports

Summary

There are many ways how to use ports on the routers. Most obvious one is to use serial port for initial RouterOS configuration after installation (by default serial0 is used by serial-terminal).

Serial and USB ports can also be used to:

- connect 3G modems;
- connect to another device through a serial cable
- access device connected to serial cable remotely.

General

Sub-menu: `/port`

Menu lists all available serial and USB ports on the router and allows to configure port parameters, like baud-rate, flow-control, etc.

Below you can see default port configuration on RB493.

```
[admin@RB493G] /port> print
Flags: I - inactive
# NAME CHANNELS USED-BY BAUD-RATE
0 serial0 1 serial-terminal 115200
```

⚠ List of the ports are maintained automatically by the RouterOS.

Properties

Property	Description
baud-rate (*integer* / *auto*; Default: **auto**)	Baud rate (speed) used by the port. If set to **auto**, then RouterOS tries to detect baud rate automatically.
data-bits (*7* / *8*; Default:)	The number of data bits in each character. • **7** - true ASCII • **8** - any data (matches the size of a byte)
dtr (*on* / *off*; Default:)	Whether to enable RS-232 DTR signal circuit used by flow control.
flow-control (*hardware* / *none* / *xon-xoff*; Default:)	method of flow control to pause and resume the transmission of data.
name (*string*; Default:)	Name of the port.
parity (*even* / *none* / *odd*; Default:)	Error detection method. If enabled, extra bit is sent to detect the communication errors. In most cases parity is set to **none** and errors are handled by the communication protocol.

rts (*on* / *off*; Default:)	Whether to enable RS-232 RTS signal circuit used by flow control.
stop-bits (*1* / *2*; Default:)	Stop bits sent after each character. Electronic devices usually uses 1 stop bit.

Read-only properties

Property	Description
channels (*integer*)	Number of channels supported by the port.
inactive (*yes* / *no*)	
line-state ()	
used-by (*string*)	Shows what is using current port. For example, by default **Serial0** is used by serial-console.

Firmware

Sub-menu: `/port firmware`

This submenu allows to specify **directory** where drivers for 3g modems can be uploaded and used.

Remote Access

Sub-menu: `/port remote-access`

If you want to access serial device that can only talk to COM ports and is located somewhere else behind router, then you can use remote-access.

As defined in RFC 2217 RouterOS can transfer data from/to a serial device over TCP connection.

Enabling remote access on RouterOS is very easy:

```
/port remote-access add port=serial0 protocol=rfc2217 tcp-port=9999
```

⚠ By default serial0 is used by serial-terminal. Without releasing the port, it cannot be used by remote-access or other services

Properties

Property	Description
allowed-addresses (*IP address range*; Default: **0.0.0.0/0**)	Range of IP addresses allowed to access port remotely.
channel (*integer [0..4294967295]*; Default: **0**)	Port channel that will be used. If port has only one channel then channel number should always be **0**.
disabled (*yes* / *no*; Default: **no**)	
local-address (*IP address*; Default:)	IP address used as source address.
log-file (*string*; Default: **""**)	Name of the file, where communication will be logged. By default logging is disabled.
port (*string*; Default:)	Name of the port from Port list.
protocol (*raw* / *rfc2217*; Default: **rfc2217**)	RFC 2217 defines a protocol to transfer data from/to a serial device over TCP. If set to **raw**, then data is sent to serial as is.
tcp-port (*integer [1..65535]*; Default: **0**)	TCP port on which to listen for incoming connections.

Read-only properties

Property	Description
active (*yes* | *no*)	Whether remote access is active and ready to accept connection.
busy (*yes* | *no*)	Whether port is currently busy.
inactive (*yes* | *no*)	
logging-active (*yes* | *no*)	Whether logging to file is currently running
remote-address (*IP address*)	IP address of remote location that is currently connected.

Product Naming

Introduction

MikroTik product naming can be confusing at first glance, but all the product codes have a logical explanation and follow a code.

RouterBOARD products naming details

RouterBOARD (short version RB)

<board name> <board features>-<built-in wireless> <wireless card features>-<connector type>-<enclosure type>

Board Name

Currently, there can be three types of board names:

- **3-symbol name**
  - 1st symbol stands for series (this can either be a number or a letter)
  - 2nd digit for indicating the number of potential wired interfaces (Ethernet, SFP, SFP+)
  - 3rd digit for indicating the number of potential wireless interfaces (built-in and mPCI and mPCIe slots)
- **Word** - currently used names are: **OmniTIK, Groove, SXT, SEXTANT, Metal, LHG, DynaDish, cAP, wAP, LDF, DISC, mANTBox, QRT, DynaDish, cAP, hAP, hEX**. If the board has fundamental changes in hardware (such as completely different CPU) revision version will be added at the end
- **Exceptional naming** - 600, 800, 1000, 1100, 1200, 2011, 3011, 4011 boards are standalone representatives of the series or have more than 9 wired interfaces, so the name was simplified to full hundreds or development year.

Board Features

Board features follow immediately after board name section (no spaces or dashes), except when board name is a word, then board features are separated by space.

Currently used features (listed in the order they are used):

- **U** - USB
- **P** - power injection with a controller
- **i** - single port power injector without a controller
- **A** - more memory and (or) higher license level
- **H** - more powerful CPU
- **G** - Gigabit (may include "U","A","H", if not used with "L")
- **L** - Lite edition
- **S** - SFP port (legacy usage - SwitchOS devices)
- **e** - PCIe interface extension card
- **x<N>** - where N is number of CPU cores (x2, x16, x36, etc)
- **R** - MiniPCI or mini PCIe slot

Built-in wireless details

If the board has built-in wireless, then all its features are represented in the following format:

`<band><power_per_chain><protocol><number_of_chains>`

- **band**
  - **5** - 5Ghz
  - **2** - 2.4Ghz
  - **52** - dual-band 5Ghz and 2.4Ghz
- **power per chain**
  - (not used) - "Normal" - <23dBm at 6Mbps 802.11a; <24dBm at 6Mbps 802.11g
  - **H** - "High" - 23-24dBm at 6Mbps 802.11a; 24-27dBm at 6Mbps 802.11g
  - **HP** - "High Power" - 25-26dBm 6Mbps 802.11a; 28-29dBm at 6Mbps 802.11g
  - **SHP** - "Super High Power" - 27+dBm at 6Mbps 802.11a; 30+dBm at 6Mbps 802.11g
- **protocol**
  - (not used) - for cards with only 802.11a/b/g support
  - **n** - for cards with 802.11n support
  - **ac** - for cards with 802.11ac support
- **number_of_chains**
  - (not used) - single chain
  - **D** - dual chain
  - **T** - triple chain
- **connector type**
  - (not used) - only one connector option on the model
  - **MMCX** - MMCX connector type
  - **u.FL** - u.FL connector type

Enclosure type

- (not used) - the main type of enclosure for a product
- **BU** - board unit (no enclosure) - for a situation when a board-only option is required, but the main product already comes in the case
- **RM** - rack-mount enclosure
- **IN** - indoor enclosure
- **EM** - extended memory
- **LM** - lite memory
- **BE** - black edition case
- **TC** - Tower (vertical) case
- **PC** - PassiveCooling enclosure (for CCR)
- **TC** - Tower (vertical) Case enclosure (for hEX, hAP and other home routers.)
- **OUT** - outdoor enclosure

More Specific types OUT enclosures are:

- **SA** - sector antenna enclosure (for SXT)
- **HG** - high gain antenna enclosure (for SXT)
- **BB** - Basebox enclosure (for RB911)
- **NB** - NetBox enclosure (for RB911)
- **NM** - NetMetal enclosure (for RB911)
- **QRT** - QRT enclosure (for RB911)
- **SX** - Sextant enclosure (for RB911,RB711)
- **PB** - PowerBOX enclosure (for RB750P, RB950P)

Example

Let's decode RB912UAG-5HPnD naming

- RB (RouterBOARD)
- 912 - 9th series board with 1 wired (ethernet) interface and two wireless interfaces (built-in and mini PCIe)
- UAG - has a USB port, more memory, and gigabit ethernet port
- 5HPnD - has built-in 5GHz high power dual chain wireless card with 802.11n support.

CloudCoreRouter naming details

CloudCoreRouter (short version CCR) naming consists of:

```
<4 digit number>-<list of ports>-<enclosure type>
```

- **4 digit number**
  - 1st digit stands for series
  - 2nd (reserved)
  - 3rd-4th digit indicates the number of total CPU cores on the device
- **list of ports**
  - -<n>**G** number of 1G Ethernet ports
  - -<n>**P** number of 1G Ethernet ports with PoE-out
  - -<n>**C** number of combo 1G Ethernet/SFP ports
  - -<n>**S** number of 1G SFP ports
  - -<n>**G+** number of 2.5G Ethernet ports
  - -<n>**P+** number of 2.5G Ethernet ports with PoE-out
  - -<n>**C+** number of combo 10G Ethernet/SFP+ ports
  - -<n>**S+** number of 10G SFP+ ports
  - -<n>**XG** number of 5G/10G Ethernet ports
  - -<n>**XP** number of 5G/10G Ethernet ports with PoE-out
  - -<n>**XC** number of combo 10G/25G SFP+ ports
  - -<n>**XS** number of 25G SFP+ ports
  - -<n>**Q+** number of 40G QSFP+ ports
  - -<n>**XQ** number of 100G QSFP+ ports
- **enclosure type** - same as for RouterBOARD products.

CloudRouterSwitch and CloudSmartSwitch naming details

CloudRouterSwitch (short version CRS, RouterOS device) CloudSmartSwitch (short version CSS, SwOS device) naming consists of:

```
<3 digit number>-<list of ports>-<built-in wireless card>-<enclosure type>
```

- **3 digit number**
  - 1st digit stands for series
  - 2nd-3rd digit - total number of wired interfaces (Ethernet, SFP, SFP+)
- **list of ports**
  - -<n>**F** number of 100M Ethernet ports
  - -<n>**Fi** number of 100M Ethernet ports with PoE-out injector
  - -<n>**Fp** number of 100M Ethernet ports with controlled PoE-out
  - -<n>**Fr** number of 100M Ethernet ports with Reverse PoE (PoE-in)
  - -<n>**G** number of 1G Ethernet ports
  - -<n>**P** number of 1G Ethernet ports with PoE-out
  - -<n>**C** number of combo 1G Ethernet/SFP ports
  - -<n>**S** number of 1G SFP ports
  - -<n>**G+** number of 2.5G Ethernet ports
  - -<n>**P+** number of 2.5G Ethernet ports with PoE-out
  - -<n>**C+** number of combo 10G Ethernet/SFP ports
  - -<n>**S+** number of 10G SFP+ ports
  - -<n>**XG** number of 5G/10G Ethernet ports
  - -<n>**XP** number of 5G/10G Ethernet ports with PoE-out
  - -<n>**XC** number of combo 10G/25G SFP+ ports
  - -<n>**XS** number of 25G SFP+ ports
  - -<n>**Q+** number of 40G QSFP+ ports
  - -<n>**XQ** number of 100G QSFP+ ports
- **built-in wireless card** - same as for RouterBOARD products.
- **enclosure type** - same as for RouterBOARD products.

RouterBOARD

Introduction

On RouterBOARD devices, the following menu exists which gives you some basic information about your device:

```
[admin@demo.mt.lv] /system routerboard> print
       routerboard: yes
             model: 1200
     serial-number: 3B5E02741BF0
     firmware-type: amcc460
  factory-firmware: 2.38
  current-firmware: 7.1beta3
  upgrade-firmware: 7.2
```

Properties

Read-only properties:

Property	Description
current-firmware (*string*)	The version of the RouterBOOT loader is currently in use. Not to be confused with RouterOS operating system version.
factory-firmware (*string*)	The version of the RouterBOOT loader the particular device was manufactured with.
firmware-type (*string*)	Firmware type that is used on the particular device.
model (*string*)	Describes the model name.
routerboard (*yes* | *no*)	Whether this is a MikroTik RouterBOARD device.
serial-number (*string*)	The serial number of this particular device.
upgrade-firmware (*string*)	RouterOS upgrades also include new RouterBOOT version files, but they have to be applied manually. This line shows if a new RouterBOOT file has been found in the device. The file can either be included via a recent RouterOS upgrade or an FWF file that has been manually uploaded to the router. In either case, the newest found version will be shown here.

Upgrading RouterBOOT

RouterBOOT upgrades usually include minor improvements to overall RouterBOARD operation. It is recommended to keep this version upgraded. If you see that the **upgrade-firmware** value is bigger than **current firmware**, you simply need to perform the **upgrade** command, accept it with **y** and then reboot with **/system reboot**

```
 [admin@mikrotik] /system routerboard> upgrade
Do you really want to upgrade firmware? [y/n]
y
echo: system,info,critical Firmware upgraded successfully, please reboot for changes to take effect!
```

After rebooting, the **current-firmware** value should become identical with **upgrade-firmware**

Settings

Sub-menu level: `/system routerboard settings`

```
[admin@MikroTik] /system/routerboard/settings> print
             auto-upgrade: no
                baud-rate: 115200
               boot-delay: 2s
           enter-setup-on: any-key
              boot-device: nand-if-fail-then-ethernet
        preboot-etherboot: disabled
            cpu-frequency: 1200MHz
         memory-frequency: 1066DDR
            boot-protocol: bootp
      enable-jumper-reset: yes
      force-backup-booter: no
              silent-boot: yes
     protected-routerboot: disabled
     reformat-hold-button: 20s
  reformat-hold-button-max: 10m
```

Property	Description
auto-upgrade (*yes* | *no*; Default: **no**)	Whether to upgrade firmware automatically after RouterOS upgrade. The latest firmware will be applied after an additional reboot
baud-rate (*integer*; Default: **115200**)	Choose the onboard RS232 speed in bits per second (if installed)
boot-delay (*time*; Default: 2**s**)	How much time does to wait for a keystroke while booting

boot-device (*nand-if-fail-then-ethernet ...*; Default: **nand-if-fail-then-ethernet**)	Choose the way RouterBOOT loads the operating system: • ethernet - boot the device in **Etherboot** mode; • flash-boot - Flashfig mode on startup is enabled. This setting will revert to NAND after a successful configuration change OR once any user logs into the board. • flash-boot-once-then-nand - Flashfig mode on startup is enabled for a single boot only, resets to nand-if-fail-then-ethernet after that. • nand-if-fail-then-ethernet - boot RouterOS from the NAND, if RouterOS is not booting - it goes to Etherboot automatically. This is the default mode for devices straight out of the box; • nand-only - boot RouterOS from the NAND; • try-ethernet-once-then-nand - boot device in Etherboot mode once and if no server is available - it will boot directly from the NAND or of the storage type the device is using. **Note** Etherboot mode is a special state for a MikroTik device that allows you to reinstall your device using Netinstall. There are several ways to put your device into Etherboot mode depending on the device you are using: 1. Press the Reset button and power on your device (wait until "USR" led is blinking then stable "On" and when the "USR" led is "Off" - release the Reset button) - the device is booting in **bootp** mode to reinstall RouterOS using Netinstall. 2. Using the serial console, when the device is booting up, keep pressing **CTRL+E** on your keyboard until the device shows that it is **trying bootp protocol** 3. Using the serial console, press any key while the device is booting and choose "o" then "1" and "x"
boot-os (*router-os |swos*; Default: **router-os**)	Changes the booting operating system for CRS3xx series switches
boot-protocol (*bootp |dhcp ...*; Default: **bootp**)	Boot protocol to use: • bootp - the default option for booting RouterOS • dhpc - used for OpenWRT and possibly other OS
cpu-frequency (*depends on model*; Default: **depends on model**)	This option allows for changing the CPU frequency of the device. Values depend on the model, to see available options, hit the [?] button in RouterOS version 6 or the [F1] button in RouterOS version 7 on the keyboard at this prompt
cpu-mode (*power-save | regular*; Default: **power-save**)	Whether to enter CPU suspend mode in HTL instruction. Most OSs use HLT instruction during the CPU idle cycle. When the CPU is in suspend mode, it consumes less power, but in low-temperature conditions, it is recommended to choose a regular mode, so that the overall system temperature would be higher
enable-jumper-reset (*yes | no*; Default: **yes**)	Disable this to avoid accidental setting reset via the onboard jumper
enter-setup-on (*any-key | delete-key*; Default: **delete-key**)	Which key will cause the BIOS to enter configuration mode during boot delay. Useful when the serial console prints out symbols during the boot process and goes into RouterBOOT menu by itself. Note that in some serial terminal programs, it is impossible to use the Delete key to enter the setup - in this case, it might be possible to do this with the Backspace key
force-backup-booter (*yes | no*; Default: **no**)	If to use the backup RouterBOOT. This is only useful if the main loader has become corrupted somehow and cannot be fixed. So that you don't have to boot the device with a pushed reset button (which loads the backup loader), you can use this setting to load it every time • yes - backup loader will be used always • no - main booter will be used
init-delay (*timeout interval 0s..9s*; Default:)	**Used for mPCIe modems with RB9xx series devices only** In case your modem is not being recognized after a soft reboot, then you might need to add a delay before the USB port is initialized

memory-frequency (*depends on model*; Default: **depends on model**)	This option allows changing the memory frequency of the device. Values depend on the model, to see available options, hit the [?] button in RouterOS version 6 or the [F1] button in RouterOS version 7 button on the keyboard at this prompt
memory-data-rate (*depends on the model*; Default: **depends on model**)	This option allows changing the memory data rate of the device. Values depend on the model, to see available options, hit the [?] button in RouterOS version 6 or the [F1] button in RouterOS version 7 button on the keyboard at this prompt
preboot-etherboot (*time out interval 1s..30s*; Default: **disabled**)	Enables **preboot etherboot**, which runs before the regular boot device. It works the same as **boot-device=etherboot**, but has an additional timeout value. If an IP address is not received from the Netinstall server before the timeout expires, the regular booting process will start. **Note** The preboot-etherboot configuration is stored in the BIOS, downgrading RouterOS to older versions will not disable it. This feature can be disabled from the RouterOS menu or by resetting the BIOS. Since etherboot accepts IP addresses from any BOOTP/DHCP server, use preboot-etherboot-server to start etherboot only when address is received from specified Netinstall server.
preboot-etherboot-server (*IP address, any*: Default: **any**)	Sets **preboot-etherboot** to accept IP address only from the specified Netinstall server IP address. By enabling this feature, unintentional etherboot from other BOOTP/DHCP servers can be prevented.
regulatory-domain-ce (*yes* / *no*; Default: **no**)	Enables extra-low TX power for high antenna gain devices (requires a reboot)
silent-boot (*yes* / *no*; Default: **no**)	This option disables output on the serial console and beeping sounds during booting, to avoid the text output interrupting a connected device. Useful if you have some temperature monitor or modem connected to the serial port • yes - no output on the serial console and no booting beeps (does not disable the RouterOS: beep command) • no - regular info and options menu on a serial console

If the CPU or memory is overclocked and that is the reason why the router is not performing as suspected, then this is not considered a warranty case and you should return both frequencies to a nominal value.

Preboot etherboot

preboot-etherboot is a feature that instructs RouterOS device to search for a Netinstall server on every boot for specified amount of time, before starting the regular booting process (e.g., RouterOS). This feature is particularly useful for remote reinstall, as it allows the device to attempt etherboot on every startup.

preboot-etherboot-server specifies that the **preboot-etherboot** booter should look only for a Netinstall server with a specific IP address. Since by default, etherboot accepts address from any BOOTP/DHCP server, with this feature, you can specify to accept the address only from a particular Netinstall server, meaning that the device can be reinstalled without removing it from the network.

With both features enabled, any device can be reinstalled remotely without accessing RouterOS or holding the reset button. One example of starting a remote reinstall process would be to simply power cycle RouterOS device from a PoE switch or another power controller. After installation, disable your Netinstall server and simply power cycle the installed device.

To enable **preboot-etherboot** and **preboot-etherboot-server**:

1) Install RouterOS version 7.9+
2) Upgrade RouterBOOT with "/system routerboard upgrade"
3) Reboot

The feature can be set with command:

```
system/routerboard/settings/set preboot-etherboot=9s preboot-etherboot-server=10.155.115.199
```

On every next reboot/power cycle, this device will try to receive IP address from Netinstall server with IP 10.155.115.199, for 9 seconds. If there will be no such server, regular boot process will start.

⚠ In case if the preboot-etherboot is set to boot from any IP (preboot-etherboot-server is not specified), the device will enter etherboot mode from any Netinstall/DHCP server-provided IP address and will wait for the reinstall process. Also, note that RouterOS reinstall does not affect BIOS settings, and preboot-etherboot would still be enabled and would try to enter etherboot. To avoid this scenario without accessing RouterOS, simply disable any attached Netinstall/DHCP server.

Protected bootloader

The feature allows the protection of RouterOS configuration and files from a physical attacker by disabling etherboot. It is called "Protected RouterBOOT". This feature can be enabled and disabled only from within RouterOS after login, i.e., there is no RouterBOOT setting to enable/disable this feature. These extra options appear only under certain conditions. When this setting is enabled - both the reset button and the reset pin-hole are disabled. RouterBOOT menu is also disabled. The only ability to change boot mode or enable the RouterBOOT settings menu is through RouterOS. If you do not know the RouterOS password - only a complete format is possible.

A special package is provided to upgrade the backup RouterBOOT (**DANGEROUS**). Newer devices will have this new backup loader already installed at the factory. If your RouterOS is **v7**, your **factory-firmware** version is lower than **7.6** and your device displays the message → **The "protected routerboot" feature requires a backup-routerboot upgrade** ← when trying to enable the feature, do the following:
a) upgrade or downgrade the device specifically to the **7.6** release (from our download page or archive).
b) upgrade your current RouterBOOT version with "/system routerboard upgrade" then reboot the device, so that the RouterBOOT version (**current-firmware** version when checking "/system routerboard print") is the same as the firmware version ("/system resource print") installed, which should be 7.6.
c) drag and drop the v7 universal package for all architectures into the device's file system then reboot the device again. This will make your **factory-firmware** version 7.6, where you are allowed to enable the feature. After this step, you can upgrade the device to a newer release.

If your RouterOS version is **v6** and you get the same prompt, follow the same steps mentioned above, but only update/downgrade/compare your device version to specifically **6.49.7** instead and use v6 universal package for all architectures.

ⓘ Starting from the v7 version, when enabling or making any changes to the protected-routerboard feature the reset or mode button must be pressed for confirmation.

For example, when setting protected-routeboard enabled, you will be given 60 seconds to confirm by pressing the reset button, otherwise, this setting won't be enabled.

```
[admin@450] > system/routerboard/settings/set protected-routerboot=enabled
[admin@450] > system/routerboard/settings/print
                        ;;; press button within 60 seconds to confirm
                            protected routerboot enable
              auto-upgrade: no
                 baud-rate: 115200
                boot-delay: 2s
             enter-setup-on: any-key
                boot-device: nand-if-fail-then-ethernet
              cpu-frequency: auto
              boot-protocol: bootp
       enable-jumper-reset: yes
       force-backup-booter: no
                silent-boot: yes
     protected-routerboot: enabled
     reformat-hold-button: 20s
  reformat-hold-button-max: 10m
```

Property	Description

protected-routerboot (*enabled | disabled*; Default: **disabled**)	This setting disables any access to the RouterBOOT configuration settings over a console cable and disables the operation of the reset button to change the boot mode (**Netinstall will be disabled**). Access to RouterOS will only be possible with a known RouterOS admin password. Unsetting of this option is only possible from RouterOS. If you forget the RouterOS password, the only option is to perform a complete reformat of both NAND and RAM with the following method, but you **have** to know the reset button hold time in seconds. • enabled - secure mode, only RouterOS can be accessed with a RouterOS admin password. Any user input from the serial port is ignored. Etherboot is not available, RouterBOOT setting change is not possible. • disabled - regular operation, RouterBOOT settings available with serial console and reset button can be used to launch Netinstall
reformat-hold-button (*5s .. 300s*; Default: **20s**)	As an emergency recovery option, it is possible to reset everything by pressing the button at power-on for longer than reformat-hold-button time, but less than reformat-hold-button-max (new in RouterBOOT 3.38.3). When you use the button for a complete reset, the following actions are taken: `EXTREMELY DANGEROUS. Use this only if you have lost all access to the device.` 1. RouterOS, all of its files and configuration is completely and irreversibly erased by nand re-format; 2. All RouterBOOT settings are reset to defaults; 3. Board is rebooted; 4. As boot from NAND fails, it goes to etherboot automatically; 5. Netinstall is required to reinstall RouterOS. **Please note!** Reformat on some RouterBOARDS can take more than 5 minutes. After formatting the board will be ready for Netinstall.
reformat-hold-button-max (*15s .. 600s*; Default: **10m**)	Increase the security even further by setting the max hold time, this means that you must release the reset button within a specified time interval. If you set the "reformat-hold-button" to 60s and "reformat-hold-button-max" to 65s, it will mean that you must hold the button 60 to 65 seconds, not less and not more, making guesses impossible. Introduced in RouterBOOT 3.38.3

⚠ RouterBOARD that has the protected RouterBOOT setting enabled will blink the LED every second, to make counting easier. The LED will turn off for one second, and turn on for the next second.

Mode and Reset buttons

Reset button additional functionality is supported by all MikroTik devices running RouterOS

Some RouterBOARD devices have a mode button that allows you to run any script when the button it pushed.

The list of supported devices:

- RBcAP-2nD (cAP)
- RBcAPGi-5acD2nD (cAP ac)
- RBwsAP5Hac2nD (wsAP ac lite)
- RB750Gr3 (hEX)
- RB760iGS (hEX S)
- RB912R-2nD (LtAP mini, LtAP mini LTE/4G kit)
- RBD52G-5HacD2HnD (hAP ac^2)
- RBLHGR (LHG LTE/4G kit)
- RBSXTR (SXT LTE/4G kit)
- CRS328-4C-20S-4S+RM
- CRS328-24P-4S+RM
- CCR1016-12G r2
- CCR1016-12S-1S+ r2
- CCR1036-12G-4S r2
- CCR1036-8G-2S+ r2
- RBD53G-5HacD2HnD (Chateau)
- RBD53GR-5HacD2HnD (hAP ac^3)

Property	Description

enabled (*no* / *yes*; Default: **no**)	Disable or enable the operation of the button
hold-time (*time interval Min..Max*; Default:)	HoldTime:= Button functionality can be called if a button is pressed for a certain period of time: Min..Max: Min -- 0s..1m (time interval), Max -- 0s..1m (time interval) (available only starting from RouterOS 6.47beta60)
on-event (*string*; Default:)	Name of the script that will be run upon pressing the button. The script must be defined and named in the " /system scripts" menu

Example for mode button:

```
/system script add name=test-mode-button source={:log info message=("mode button pressed");}
/system routerboard mode-button set on-event=test-mode-button enabled=yes
```

Upon pressing the button, the message *"mode button pressed"* will be logged in the system log.

Starting from RouterOS 6.47 reset-button functionality and a hold-time option have been added

Examples for RouterOS version newer than 6.47:

```
/system script add name=test-mode-button source={:log info message=("mode button pressed");}
/system routerboard mode-button set on-event=test-mode-button hold-time=3..5 enabled=yes
```

The reset button works in the same way, but the menu is moved under the : `/system routerboard reset-button`:

```
/system script add name=test-reset-button source={:log info message=("reset button pressed");}
/system routerboard reset-button set on-event=test-reset-button hold-time=0..10 enabled=yes
```

LED dark mode control with the mode button:

```
/system script add name=dark-mode source={
   :if ([system leds settings get all-leds-off] = "never") do={
      /system leds settings set all-leds-off=immediate
   } else={
        /system leds settings set all-leds-off=never
   }
 }
/system routerboard mode-button set enabled=yes on-event=dark-mode
```

The C53, S53 and cAP ax series RouterBoards have configurable WPS button. It also works in the same way as reset button and mode button and executes a script.

```
/system script add name=test-wps-button source={:log info message=("wps button pressed");}
/system routerboard wps-button set on-event=test-wps-button hold-time=0..10 enabled=yes
```

WPS accept control with the WPS button:

```
/system script add name=wps-accept source={
    :foreach iface in=[/interface/wifiwave2 find where configuration.mode="ap"] do={
        /interface/wifiwave2 wps-push-button $iface
    }
}
/system routerboard wps-button set enabled=yes on-event=wps-accept
```

USB Features

Summary

Sub-menu: `/system routerboard usb`
Package: `system (v6, v7)`

This article describes RouterBoard device supported USB features.

USB Power Reset

USB power reset turns off USB port power for a specified time. It is useful when 3G/LTE modem needs to be restarted but there is no physical access to it.

Available properties:

- duration (*time*; Default: "3s") - Time interval of how long power is turned off.

For example, turn off USB port power for 10 seconds:

```
/system routerboard usb power-reset duration=10s
```

RouterBoards with multiple USB buses also require a bus specified in order to do a USB power reset.
Available properties:

- duration (*time*; Default: "3s") - Time interval of how long power is turned off.
- bus (*integer*; Default: 1) - USB bus where power reset is applied.

⚠ **RB953GS**: The miniPCIe slot closer to Ethernet ports on the RB953GS board is the one which is shared with USB port and has configurable USB port type, its USB bus number is 1. Depending on USB port type the power reset is done on USB port or miniPCIe slot. The other miniPCIe slot closer to SFP slots has independent bus - 2.

⚠ **RB922UAG**: The USB port has an independent bus - 1. The miniPCIe slot has an independent bus - 2.

⚠ **CCR1072-1G-8S+**: The micro USB port has independent bus - 0. The USB Type-A port has independent bus - 1.

USB Port Type

RB912UAG and RB953GS have partially shared USB port and miniPCIe slot. Due to hardware restrictions it is possible to use only one at the time for 3G/LTE modem.

```
[admin@MikroTik] > /system routerboard usb set type=
USB-type-A  mini-PCIe
```

Available properties:

- type (*USB-type-A* | *mini-PCIe*; Default: "USB-type-A") - Type of enabled port.

RB953GS: The miniPCIe slot closer to Ethernet ports on the RB953GS board is the one which is shared with USB port and has configurable USB port type, its USB bus number is 1. Depending on USB port type the power reset is done on USB port or miniPCIe slot. The other miniPCIe slot closer to SFP slots has independent bus - 2.

USB Port Mode

RB2011 series, CRS1xx series, and mAP have micro USB port which operates in host mode when USB device is attached through USB OTG cable. Some vendor cables require forced host mode to recognize connected device.

Available properties:

- usb-mode (*automatic* | *force-host*; Default: "automatic") - Defines USB port mode.

Warning

On RB2011 and CRS1xx series boards USB devices may not work first time they are plugged in. In such cases power cycle (not reboot) is required.

RouterBoard USB Port Limitations

Some RouterBoard devices have notable USB port limitations.

* RB mAP 2n - does not support USB Power Reset
* RB750UP - does not support USB Power Reset
* RB751U-2HnD - does not support USB Power Reset
* RB751G-2HnD - does not support USB Power Reset
* RB411U - does not support USB Power Reset
* RB411UAHR - does not support USB Power Reset and USB Powering (USB power injector is required)
* RB433UAH - does not support USB Power Reset
* RB435G - does not support USB Power Reset
* RB493G - does not support USB Power Reset and USB Powering (USB power injector is required)

RouterOS does not support memory slot from USB modems.

Diagnostics, monitoring and troubleshooting

In This Section:

Bandwidth Test

Summary

```
Sub-menu: /tool
Packages required: system
```

The Bandwidth Tester can be used to measure the throughput to another MikroTik router (either wired or wireless) and thereby help to discover network "bottlenecks".

The TCP test uses the standard TCP protocol with acknowledgments and follows the TCP algorithm on how many packets to send according to latency, dropped packets, and other features in the TCP algorithm. Please review the TCP protocol for details on its internal speed settings and how to analyze its behavior. Statistics for throughput are calculated using the entire size of the TCP data stream. As acknowledgments are an internal working of TCP, their size and usage of the link are not included in the throughput statistics. Therefore this statistic is not as reliable as the UDP statistic when estimating throughput.

The UDP tester sends 110% or more packets than currently reported as received on the other side of the link. To see the maximum throughput of a link, the packet size should be set for the maximum MTU allowed by the links which is usually 1500 bytes. There is no acknowledgment required by UDP; this implementation means that the closest approximation of the throughput can be seen.

- Up to RouterOS version 6.44beta39 Bandwidth Test used only single CPU core and reached its limits when core was 100% loaded.
- Bandwidth Test uses all available bandwidth (by default) and may impact network usability.

- Bandwidth Test uses a lot of resources. If you want to test real throughput of a router, you should run bandwidth test through the tested router not from or to it. To do this you need at least 3 routers connected in chain: the Bandwidth Server, the router being tested and the Bandwidth Client.
- If you use UDP protocol then Bandwidth Test counts IP header+UDP header+UDP data. In case if you use TCP then Bandwidth Test counts only TCP data (TCP header and IP header are not included).

Bandwidth Test Server

```
Sub-menu: /tool bandwidth-server
```

Property	Description
allocate-udp-ports-from (*integer 1000..64000*; Default: **2000**)	Beginning of UDP port range
authenticate (*yes* | *no*; Default: **yes**)	Communicate only with authenticated clients
enabled (*yes* | *no*; Default: **yes**)	Defines whether bandwidth server is enabled or not
max-sessions (*integer 1..1000*; Default: **100**)	Maximal simultaneous test count

Example

Bandwidth Server:

```
[admin@MikroTik] /tool bandwidth-server> print
                  enabled: yes
             authenticate: yes
  allocate-udp-ports-from: 2000
             max-sessions: 100
[admin@MikroTik] /tool bandwidth-server>
```

Active sessions:

```
[admin@MikroTik] /tool bandwidth-server session> print
  # CLIENT          PROTOCOL DIRECTION USER
  0 35.35.35.1      udp      send      admin
  1 25.25.25.1      udp      send      admin
  2 36.36.36.1      udp      send      admin
[admin@MikroTik] /tool bandwidth-server session>
```

To enable **bandwidth-test** server without client authentication:

```
[admin@MikroTik] /tool bandwidth-server> set enabled=yes authenticate=no
[admin@MikroTik] /tool bandwidth-server> print
                  enabled: yes
             authenticate: no
  allocate-udp-ports-from: 2000
             max-sessions: 100
[admin@MikroTik] /tool bandwidth-server>
```

Bandwidth Test Client

```
Sub-menu: /tool bandwidth-test
```

Property	Description
address (*IP address* | *IPv6 prefix[%interface]*; Default:)	IP address of host
direction (*both* | *receive* | *transmit*; Default: **receive**)	Direction of data flow
duration (*time*; Default:)	Duration of the test
interval (*time: 20ms..5s*; Default: **1s**)	Delay between reports (in seconds)
local-tx-speed (*integer 0..18446744073709551615*; Default:)	Transfer test maximum speed (bits per second)
local-udp-tx-size (*integer: 28..64000*)	Local transmit packet size in bytes
password (*string*; Default: **""**)	Password for the remote user
protocol (*udp* | *tcp*; Default: **udp**)	Protocol to use
random-data (*yes* | *no*; Default: **no**)	If random-data is set to yes, the payload of the bandwidth test packets will have incompressible random data stream so that links that use data compression will not distort the results (this is CPU intensive and random-data should be set to no for low speed CPUs)

remote-tx-speed (*integer 0.. 18446744073709551615*; Default:)	Receive test maximum speed (bits per second)
remote-udp-tx-size (*integer : 28..64000*)	Remote transmit packet size in bytes
connection-count (*integer 1..255*; Default:)	Number of TCP connections to use
user (*string*; Default: **""**)	Remote user

Example

To run 15-second long bandwidth-test to the **10.0.0.32** host sending and receiving **1000**-byte UDP packets and using username **admin** to connect:

```
[admin@MikroTik] /tool> bandwidth-test 10.0.0.32 duration=15s \
\... direction=both local-udp-tx-size=1000 protocol=udp \
\... remote-udp-tx-size=1000 user=admin
                status: done testing
              duration: 15s
            tx-current: 272.8Mbps
  tx-10-second-average: 200.3Mbps
      tx-total-average: 139.5Mbps
            rx-current: 169.6Mbps
  rx-10-second-average: 164.8Mbps
      rx-total-average: 117.0Mbps
          lost-packets: 373
           random-data: no
             direction: both
               tx-size: 1000
               rx-size: 1000
[admin@MikroTik] /tool>
```

Link-local IPv6 example:

```
[admin@MikroTik] > /tool bandwidth-test fe80::34:23ff:fe6a:570c%local
                status: running
              duration: 5s
            rx-current: 23.9Mbps
  rx-10-second-average: 15.1Mbps
      rx-total-average: 15.1Mbps
          lost-packets: 0
           random-data: no
             direction: receive
               rx-size: 1500
```

Detect Internet

Introduction

Detect Internet is a tool that categorizes monitored interfaces into the following states - **Internet**, **WAN**, **LAN**, **unknown**, and **no-link**.

State

This submenu displays status of all monitored interfaces defined by the *detect-interface-list* parameter:

```
interface/detect-internet/state/print
```

LAN

All layer 2 interfaces initially have this state.

WAN

Any L3 tunnel and LTE interfaces will initially have this state. Layer 2 interfaces can obtain this state if:

- an interface has an active route to 8.8.8.8 in main routing table.
- an interface can obtain (dynamic DHCP client is created) or has obtained an address from DHCP (does not apply if DHCP server is also running Detect Internet on the DHCP server interface).

⚠ WAN interface can fall back to LAN state only when link status changes. LAN interfaces get locked to LAN after 1h and then change only when link status changes.

⚠ Note that Detect Internet can install DHCP clients, default routes, DNS servers and affect other facilities.
Use with precaution, and after enabling the service, check how it interferes with your other configuration.

Internet

WAN interfaces that can reach cloud.mikrotik.com using UDP protocol port 30000 can obtain this state. Reachability is checked every minute. If a cloud is not reached for 3 minutes, the state falls back to **WAN**.

Configuration

```
/interface detect-internet
```

Property	Description
detect-interface-list (*interface list*; Default: **none**)	All interfaces in the list will be monitored by Detect Internet
internet-interface-list (*interface list*; Default: **none**)	Interfaces with state Internet will be dynamically added to this list
lan-interface-list (*interface list*; Default: **none**)	Interfaces with state Lan will be dynamically added to this list
wan-interface-list (*interface list*; Default: **none**)	Interfaces with state Wan will be dynamically added to this list

```
[admin@MikroTik] > interface/detect-internet/print
detect-interface-list: none
lan-interface-list: none
wan-interface-list: none
internet-interface-list: none
[admin@MikroTik] > interface/detect-internet/set internet-interface-list=all wan-interface-list=all lan-
interface-list=all detect-interface-list=all
[admin@MikroTik] > interface/detect-internet/state/print
Columns: NAME, STATE, STATE-CHANGE-TIME, CLOUD-RTT
# NAME STATE STATE-CHANGE-TIME CLO
0 ether1 internet dec/22/2020 13:46:18 5ms
```

Dynamic DNS

Introduction

```
/tool dns-update
```

Standards: `RFC 2136, RFC 3007`

Dynamic DNS Update Tool gives a way to keep the domain name pointing to a dynamic IP address. It works by sending a domain name system update requests to the name server, which has a zone to be updated. Secure DNS updates are also supported.

The DNS update tool supports only one algorithm - **hmac-md5**. It's the only proposed algorithm for signing DNS messages.

DNS update tool works only with the BIND server, it will not work with DynDNS, EveryDNS, or any other similar service.

Properties

Property	Description
address (*IP*; Default:)	Defines the IP address associated with the domain name.
dns-server (*IP*; Default:)	DNS server to send updates to.
key (*string*; Default:)	Authorization key to access the server.
key-name (*string*; Default:)	Authorization key name (like a username) to access the server.
name (*string*; Default:)	Name to attach with the IP address.
ttl (*integer*; Default:)	Time to live for the item (in seconds).
zone (*string*; Default:)	DNS zone where to update the domain name in.

The system clock time on your router can't differ from the DNS server's time by more than 5 minutes. Otherwise, the DNS server will ignore this request.

Example

To tell 23.34.45.56 DNS server to (re)associate mydomain name in the myzone.com zone with 68.42.14.4 IP address specifying that the name of the key is dns-update-key and the actual key updates:

```
[admin@MikroTik] tool> dns-update dns-server=23.34.45.56 name=mydomain \
\... zone=myzone.com address=68.42.14.4 key-name=dns-update-key key=update
```

Graphing

Summary

Graphing is a tool to monitor various RouterOS parameters over time and put collected data in graphs.
Watch our video about this feature.

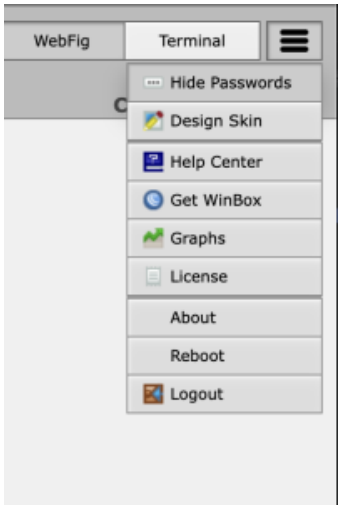
> We suggest not storing graphs on disk for devices with small built-in memory.

The Graphing tool can display graphics for:

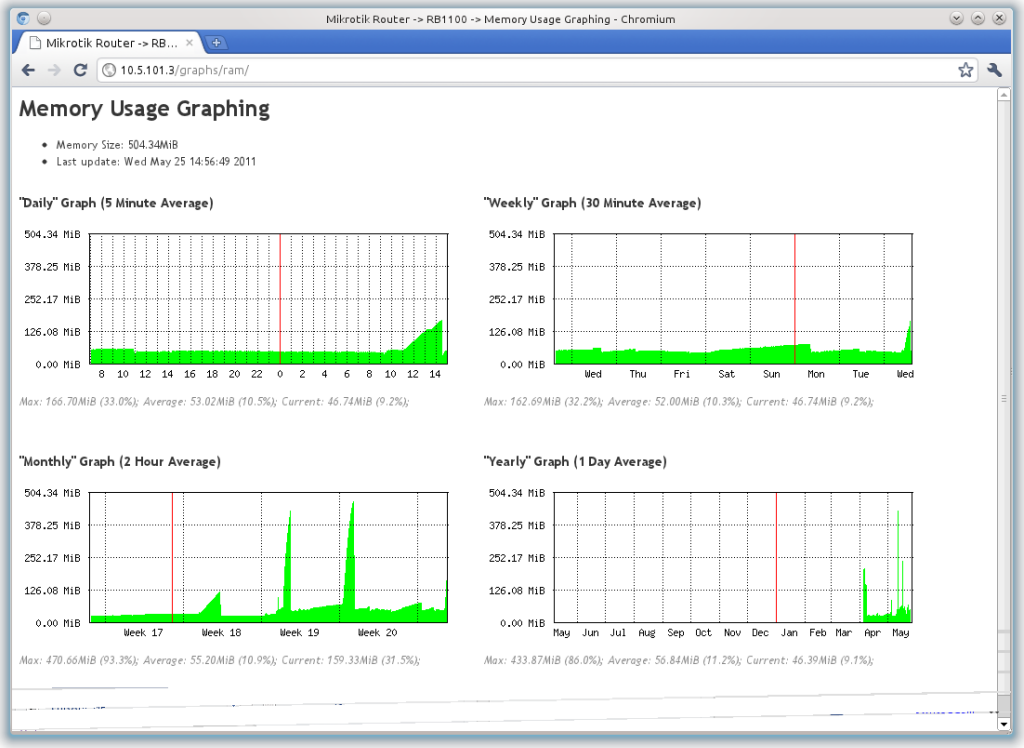
- Resource usage (CPU, Memory, and Disk usage)
- Traffic which is passed through an interface
- Traffic which is passed through a simple queue

Graphing consists of two parts - the first part collects information and the other part displays data on a Web page. To access the graphics, type http://[Router_IP_address]/graphs/ and choose a graphic to display in your Web browser.

Alternatively, look for the menu ≡ (triple bar sign) in the top right corner of the WebFig interface, allowing you to find "graphs":



Example of memory graphs:



Configuration

General

The configuration is done under "/tool graphing" menu, by default, graphing is disabled. You can configure graphing for interfaces, resources, and simple queues in their respective submenus.

Sub-menu: `/tool graphing`

Property	Description
store-every (*24hours* | *5min* | *hour*; Default: **5min**)	How often to write collected data to system drive.
page-refresh (*integer* | *never*; Default: **300**)	How often graph page is refreshed

Interface graphing

Sub-menu allows configuration for which interface graphing will collect bandwidth usage data.

Sub-menu: `/tool graphing interface`

Property	Description
allow-address (*IP/IPv6 prefix*; Default: **0.0.0.0/0**)	IP address range from which is allowed to access graphing information
comment (*string*; Default:)	Description of current entry
disabled (*yes* | *no*; Default: **no**)	Defines whether item is used
interface (*all* | *interface name*; Default: **all**)	Defines which interface will be monitored. **all** means that all interfaces on router will be monitored.

store-on-disk (*yes* / *no*; Default: **yes**)	Defines whether to store collected information on system drive.

Queue graphing

Sub-menu allows configuration for which simple queue graphing will collect bandwidth usage data.

Sub-menu: `/tool graphing queue`

Property	Description
allow-address (*IP/IPv6 prefix*; Default: **0.0.0.0/0**)	IP address range from which is allowed to access graphing information
allow-target (*yes* / *no*; Default: **yes**)	Whether to allow access to graphs from queue's target-address
comment (*string*; Default:)	Description of current entry
disabled (*yes* / *no*; Default: **no**)	Defines whether item is used
simple-queue (*all* / *queue name*; Default: **all**)	Defines which queues will be monitored. **all** means that all queues on router will be monitored.
store-on-disk (*yes* / *no*; Default: **yes**)	Defines whether to store collected information on system drive.

⚠ If simple queue has target-address set to 0.0.0.0/0 everyone will be able to access queue graphs even if allow address is set to specific address. This happens because by default queue graphs are accessible also from the target address.

Resource graphing

Sub-menu allows to enable graphing of system resources. Graphing collects data of:

- CPU usage
- Memory usage
- Disk usage

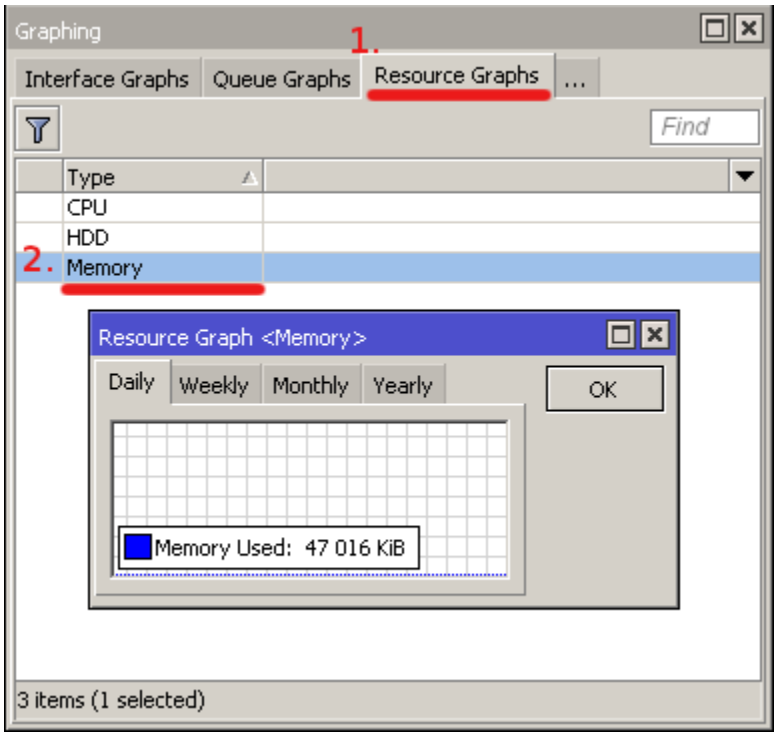
Sub-menu: `/tool graphing resource`

Property	Description
allow-address (*IP/IPv6 prefix*; Default: **0.0.0.0/0**)	IP address range from which is allowed to access graphing information
comment (*string*; Default:)	Description of current entry
disabled (*yes* / *no*; Default: **no**)	Defines whether item is used
store-on-disk (*yes* / *no*; Default: **yes**)	Defines whether to store collected information on system drive

Graphing in WinBox

Winbox allows viewing the same collected information as on the web page. Open **Tools->Graphing** window. Double click on the entry of which you want to see graphs.

The image below shows WinBox graphs of memory usage:


Graphing
1.
Interface Graphs
Queue Graphs
Resource Graphs
...
Find
Type
CPU
HDD
2.
Memory
Resource Graph <Memory>
Daily
Weekly
Monthly
Yearly
OK
Memory Used: 47 016 KiB
3 items (1 selected)

Health

Summary

Hardware that supports monitoring will display different information about hardware status, like temperature, voltage, current, fan-speed, etc.

Example on CCR1072-1G-8S+ device:

```
[admin@MikroTik] > system/health/print
Columns: NAME, VALUE, TYPE
 #  NAME                VALUE  TYPE
 0  power-consumption   50.8   W
 1  cpu-temperature     43     C
 2  fan1-speed          5654   RPM
 3  fan2-speed          5825   RPM
 4  fan3-speed          5800   RPM
 5  fan4-speed          5750   RPM
 6  board-temperature1  29     C
 7  board-temperature2  28     C
 8  psu1-voltage        0      V
 9  psu2-voltage        12.1   V
10  psu1-current        0      A
11  psu2-current        4.2    A
```

Warning: For feature availability on RouterBOARD products check mikrotik.com

Intensive health monitoring on the CCR2004-16G-2S+PC device (from the console, winbox or SNMP) causes significant CPU load.

Voltage

Routers that support voltage monitoring will display supplied voltage value. In CLI/Winbox it will display volts. In scripts/API/SNMP this will be dV or value showed in CLI/Winbox

Note: Routers that have PEXT and PoE power input are calibrated using PEXT, as a result, value showed over PoE can be lower than input voltage due to additional ethernet protection chains.

```
[admin@MikroTik] > system/health/print
Columns: NAME, VALUE, TYPE
#  NAME         VALUE  TYPE
0  voltage      23.8   V
1  temperature  39     C
```


Note: If old revision CRS112, CRS210 and CRS109 devices are powered with PoE - Health will show correct voltage only up to 26.7V. If higher voltage will be used - Health will show constant 16V.

Temperature

Routers that support temperature monitoring will display temperature reading. In CLI/Winbox it will display degrees Celsius. Using scripts/API/SNMP this value will be shown in CLI/Winbox multiplied by 10. There are various temperature sensors depending on the device. These sensors may refer to: cpu-temperature, pcb-temperature, sfp-temperature. Device tested ambient temperature range you can find in specification description at mikrotik.com. Tested ambient temperature range is temperature in which device can be physically located. It is **not** the same as temperature which reports system health monitor!

Fan control and behavior

```
/system health set
```

Using this menu users will be able to control fan behaviour on TILE architecture devices.

Note: Improved FAN stability starting from version 6.45.5.

There are three parameters that may affect fan behaviour: PoE-out consumption, SFP temperature and CPU temperature. As soon as one of the parameters exceeds the optimal value the, fans are started.

PoE-out consumption

If a device has PoE-out, then the fan RPM will change as described below:

PoE-out load	RPM % of max FAN speed
0%..24%	FAN speed 0%
25%..46%	FAN speed 25%
47%..70%	FAN speed 50%
71%..92%	FAN speed 75%
93%..	FAN speed 100%

For devices with **PWM** fans, the speed will linearly increase or decrease from 9..88% (note: below 100W the fan RPM=0)

Limited manual fan-control option

Note: Starting from RouterOS version 7.9 limited manual fan-control options have been added for CRS3xx, CRS5xx and CCR2xxx devices.

Note: Starting from RouterOS version 7.14 limited manual fan-control is available for the CCR1036-8G-2S+-r2, CCR1036-12G-4S-r2 and CCR1016-12S-1S+-r2 devices.

Fan behavior can be manipulated using the settings section of system health:

```
/system health settings set
```

Available properties are described below:

Property	Description
fan-full-speed-temp (*integer [-273..65];* Default: **65**)	Sets the temperature value upon which the fan speed will be increased to the maximum possible rpm. Reads temperature from CPU, PHY, SWITCH and SFP and adjusts fan speed based on the component with the highest temperature.
fan-target-temp (*integer [-273..65];* Default: **58**)	Sets the target temperature for the hottest component. Based on this setting adjusts fan behavior to hold temperatures in target range.
fan-min-speed-percent (*integer [0..100];* Default: **depends on FAN controller**)	Sets the minimum percentage of fan speed thus not allowing fans to have a lower rpm than this value. ***NOTE:** the default value may vary based on FAN controller chip and/or specific model requirements. From RouterOS verson 7.14 default value is set to **12,** all previous versions have **0.**

fan-control-interval (*integer [5..30];* Default: **30**)	Sets the actual temperature data read interval to get temperature values from CPU, PHY, SWITCH and SFP. ***NOTE:** THIS SETTING DIRECTLY AFFECTS CPU USAGE

Brief description of the fan-control

If at least one of the internal measured (CPU, SFP, Switch, Board etc.) temperatures exceed **fan-target-temp**, the fans will start to spin. The higher the temperature, the faster the fans will spin. For devices with PWM fans, as the internal measured temperatures exceed **fan-target-temp**, the fans will linearly increase their RPM to try to keep the temperature at **fan-target-temp** if possible and will get to their Max RPM when the temperature is equal or exceeds **fan-full-speed-temp**. For devices with DC fans, as the internal measured temperatures exceed **fan-target-temp**, the fans will start spinning but at a higher minimum RPM by default. This may result in cooling the device to the point where the fans turn-off completely if **fan-min-speed-percent** is set to **0%**, while with the default value of **12%** fans will never go to a full stop therefore reducing the noise and On/Off peaks that may occur. The temperature then may slowly increase to **fan-target-temp** and the fans will turn on again. Currently, there is one exception. The S+RJ10 modules have a temperature threshold of 65C before they trigger the fans. Since it's a higher temperature threshold, the fans will start spinning at a higher initial speed to cool the device. All the above mentioned functionality is directly related to the **fan-control-interval** parameter value as it will determine how often FAN controller monitors all sensor data and triggers changes in fan-control.

> ⚠ **Note:** PWM and DC fans react to fan-control differently. While PWM fans will increase/decrease their RPM in a linear way the DC fans have only few possible speed ratings at which they may operate.

Note: All readings are approximate and may not be 100% precise. Their purpose is to ~inform users about possible/upcoming failures.

Interface stats and monitor-traffic

Summary

Every RouterOS interface contains various counters, for example, the number of received and transmitted packets, Fast Path bytes, and link downs.

Stats

Use the `stats` or `stats-detail` commands to print the interface counters. The values starting with "fp" indicates the Fast Path counters. The value "tx-queue-drop" indicates the number of dropped packets by the interface queue.

```
[admin@MikroTik] > /interface print stats
Flags: R - RUNNING
Columns: NAME, RX-BYTE, TX-BYTE, RX-PACKET, TX-PACKET, TX-QUEUE-DROP
#   NAME        RX-BYTE      TX-BYTE  RX-PACKET  TX-PACKET  TX-QUEUE-DROP
0 R ether1  205 149 015  147 887 338    158 132    150 015              2
1 R ether2   32 400 148  335 690 764         19    216 509              0
2 R ether3  944 043 040   32 392 350    617 271         17              0
3 R ether4    7 038 417   32 398 973          9          4              0
4 R ether5    7 036 903   32 502 437          5          9              0
5   sfp1              0            0          0          0              0
[admin@MikroTik] > /interface print stats-detail
Flags: D - dynamic; X - disabled, R - running; S - slave; P - passthrough
 0  R   name="ether1" last-link-down-time=jul/19/2022 12:37:06 last-link-up-time=jul/19/2022 12:37:09 link-
downs=2
        rx-byte=205 164 277 tx-byte=147 977 500 rx-packet=158 254 tx-packet=150 156 tx-queue-drop=2
        fp-rx-byte=199 271 067 fp-tx-byte=0 fp-rx-packet=1 473 603 fp-tx-packet=0

 1  R   name="ether2" last-link-down-time=jul/19/2022 12:46:06 last-link-up-time=jul/19/2022 12:46:07 link-
downs=10
        rx-byte=32 400 148 tx-byte=335 690 764 rx-packet=19 tx-packet=216 509 tx-queue-drop=0
        fp-rx-byte=33 718 434 fp-tx-byte=67 fp-rx-packet=60 037 fp-tx-packet=1

 2  R   name="ether3" last-link-down-time=jul/19/2022 12:46:06 last-link-up-time=jul/19/2022 12:46:08 link-
downs=11
        rx-byte=944 043 040 tx-byte=32 392 350 rx-packet=617 271 tx-packet=17 tx-queue-drop=0 fp-rx-byte=6 860
921
        fp-tx-byte=0 fp-rx-packet=46 671 fp-tx-packet=0

 3  R   name="ether4" last-link-down-time=jul/19/2022 12:46:06 last-link-up-time=jul/19/2022 12:46:07 link-
downs=10
        rx-byte=7 038 417 tx-byte=32 398 973 rx-packet=9 tx-packet=4 tx-queue-drop=0 fp-rx-byte=6 852 283
        fp-tx-byte=0 fp-rx-packet=46 586 fp-tx-packet=0

 4  R   name="ether5" last-link-down-time=jul/19/2022 12:46:06 last-link-up-time=jul/19/2022 12:46:08 link-
downs=10
        rx-byte=7 036 903 tx-byte=32 502 437 rx-packet=5 tx-packet=9 tx-queue-drop=0 fp-rx-byte=6 850 637
        fp-tx-byte=178 fp-rx-packet=46 568 fp-tx-packet=2

 5      name="sfp1" link-downs=0 rx-byte=0 tx-byte=0 rx-packet=0 tx-packet=0 tx-queue-drop=0 fp-rx-byte=0
        fp-tx-byte=0 fp-rx-packet=0 fp-tx-packet=0
```

Monitor traffic

The traffic passing through any interface can be monitored using the `monitor-traffic` command.

```
[admin@MikroTik] > /interface monitor-traffic [find]
                         name:     ether1 ether2 ether3 ether4 ether5 sfp1
         rx-packets-per-second:         19      0      0      0      0    0
            rx-bits-per-second:   27.8kbps   0bps   0bps   0bps   0bps 0bps
      fp-rx-packets-per-second:         29      0      0      0      0    0
         fp-rx-bits-per-second:   26.8kbps   0bps   0bps   0bps   0bps 0bps
         tx-packets-per-second:         21      0      0      0      0    0
            tx-bits-per-second:  149.4kbps   0bps   0bps   0bps   0bps 0bps
      fp-tx-packets-per-second:          0      0      0      0      0    0
         fp-tx-bits-per-second:       0bps   0bps   0bps   0bps   0bps 0bps
     tx-queue-drops-per-second:          0      0      0      0      0    0
```

Additional Ethernet statistics are available in the "/interface ethernet" menu.

IP Scan

Summary

IP Scan tool allows a user to scan networks based on some network prefix or by setting an interface to listen to. Either way, the tool collects certain data from the network:

- address - IP address of network device;
- mac-address - MAC address of network device;
- time - response time of seen network device when found;
- DNS - DNS name of a network device;
- SNMP - SNMP name of the device;
- NET-BIOS - NET-BIOS name of device if advertised by the device;

When using IP scan tool user must choose what they want to scan for:

- certain IPv4 prefix - the tool will attempt to scan all the IP addresses or addresses set;
- the interface of the router - the tool will attempt to listen to packets that are "passing by" and attempt to compile results when something is found;

> There is a possibility to set both but then results may be inconclusive!

Quick Example

In the following example, we will scan the devices on 10.155.126.0/24 network:

```
[admin@MikroTik] > /tool ip-scan address-range=10.155.126.1-10.155.126.255
Columns: ADDRESS, MAC-ADDRESS, TIMe, SNMP
  ADDRESS         MAC-ADDRESS        TIM  SNMP
  10.155.126.1    E4:8D:8C:1C:D3:18  2ms  CCR1036-8G-2S+
  10.155.126.251                     2ms
  10.155.126.151  E4:8D:8C:49:49:DB  1ms
  10.155.126.153  6C:3B:6B:48:0E:8B  1ms  750Gr3
  10.155.126.249  CC:2D:E0:8D:01:88  0ms  CRS328-24P-4S+
  10.155.126.250  B8:69:F4:B3:1B:D2  0ms
  10.155.126.252  6C:3B:6B:ED:83:69  0ms
  10.155.126.253  6C:3B:6B:ED:81:83  0ms
```

Log

Summary

RouterOS is capable of logging various system events and status information. Logs can be saved in routers memory (RAM), disk, file, sent by email or even sent to remote syslog server (RFC 3164).

Log messages

Sub-menu level:
`/log`

All messages stored in routers local memory can be printed from `/log` menu. Each entry contains time and date when event occurred, topics that this message belongs to and message itself.

```
[admin@MikroTik] /log> print
jan/02/1970 02:00:09 system,info router rebooted
sep/15 09:54:33 system,info,account user admin logged in from 10.1.101.212 via winbox
sep/15 12:33:18 system,info item added by admin
sep/15 12:34:26 system,info mangle rule added by admin
sep/15 12:34:29 system,info mangle rule moved by admin
sep/15 12:35:34 system,info mangle rule changed by admin
sep/15 12:42:14 system,info,account user admin logged in from 10.1.101.212 via telnet
sep/15 12:42:55 system,info,account user admin logged out from 10.1.101.212 via telnet
01:01:58 firewall,info input: in:ether1 out:(none), src-mac 00:21:29:6d:82:07, proto UDP,
10.1.101.1:520->10.1.101.255:520, len 452
```

If logs are printed at the same date when log entry was added, then only time will be shown. In example above you can see that second message was added on sep/15 current year (year is not added) and the last message was added today so only the time is displayed.

Print command accepts several parameters that allows to detect new log entries, print only necessary messages and so on.

For example following command will print all log messages where one of the topics is info and will detect new log entries until Ctrl+C is pressed.

```
[admin@MikroTik] /log > print follow where topics~".info"
12:52:24 script,info hello from script
-- Ctrl-C to quit.
```

If print is in follow mode you can hit 'space' on keyboard to insert separator:

```
[admin@MikroTik] /log > print follow where topics~".info"
12:52:24 script,info hello from script

= = = = = = = = = = = = = = = = = = = = = = = = = = = = =

-- Ctrl-C to quit.
```

Logging configuration

Sub-menu level: `/system logging`

Property	Description
action (*name*; Default: **memory**)	specifies one of the system default actions or user specified action listed in actions menu
prefix (*string*; Default:)	prefix added at the beginning of log messages
topics (*account, bfd, caps, ddns, dns, error, gsm, info, iscsi, l2tp, manager, ntp, packet, pppoe, radvd, rip, script, smb, sstp, system, timer, vrrp, web-proxy, async, bgp, certificate, debug, dot1x, dude, event, hotspot, interface, isdn, ldp, mme, ospf, pim, pptp, raw, route, sertcp, snmp, state, telephony, upnp, warning, wireless, backup, calc, critical, dhcp, e-mail, firewall, igmp-proxy, ipsec, kvm, lte, mpls, ovpn, ppp, radius, read, rsvp, simulator, ssh, store, tftp, ups, watchdog, write*; Default: **info**)	log all messages that falls into specified topic or list of topics. **'!'** character can be used before topic to exclude messages falling under this topic. For example, we want to log NTP debug info without too much details: `/system logging add topics=ntp,debug,!packet`

Actions

Sub-menu level: `/system logging action`

Property	Description
bsd-syslog (*yes*|*no*; Default:)	whether to use bsd-syslog as defined in RFC 3164
disk-file-count (*integer [1..65535]*; Default: **2**)	specifies number of files used to store log messages, applicable only if action=disk
disk-file-name (*string*; Default: **log**)	name of the file used to store log messages, applicable only if action=disk
disk-lines-per-file (*integer [1..65535]*; Default: **100**)	specifies maximum size of file in lines, applicable only if action=disk
disk-stop-on-full (*yes*|*no*; Default: **no**)	whether to stop to save log messages to disk after the specified disk-lines-per-file and disk-file-count number is reached, applicable only if action=disk
email-start-tls (*yes* | *no*; Default: **no**)	Whether to use tls when sending email, applicable only if action=email
email-to (*string*; Default:)	email address where logs are sent, applicable only if action=email
memory-lines (*integer [1..65535]*; Default: **1000**)	number of records in local memory buffer, applicable only if action=memory
memory-stop-on-full (*yes*|*no*; Default: **no**)	whether to stop to save log messages in local buffer after the specified memory-lines number is reached
name (*string*; Default:)	name of an action
remember (*yes*|*no*; Default:)	whether to keep log messages, which have not yet been displayed in console, applicable if action=echo
remote (*IP/IPv6 Address[:Port]*; Default: **0.0.0.0:514**)	remote logging server's IP/IPv6 address and UDP port, applicable if action=remote

src-address (*IP address*; Default: **0.0.0.0**)	source address used when sending packets to remote server
syslog-facility (*auth, authpriv, cron, daemon, ftp, kern, local0, local1, local2, local3, local4, local5, local6, local7, lpr, mail, news, ntp, syslog, user, uucp*; Default: **daemon**)	
syslog-severity (*alert, auto, critical, debug, emergency, error, info, notice, warning*; Default: **auto**)	Severity level indicator defined in RFC 3164: • Emergency: system is unusable • Alert: action must be taken immediately • Critical: critical conditions • Error: error conditions • Warning: warning conditions • Notice: normal but significant condition • Informational: informational messages • Debug: debug-level messages
target (*disk, echo, email, memory, remote*; Default: **memory**)	storage facility or target of log messages • disk - logs are saved to the hard drive • echo - logs are displayed on the console screen • email - logs are sent by email • memory - logs are stored in local memory buffer • remote - logs are sent to remote host

Topics

Each log entry have topic which describes the origin of log message. There can be more than one topic assigned to log message. For example, OSPF debug logs have four different topics: route, ospf, debug and raw.

```
11:11:43 route,ospf,debug SEND: Hello Packet 10.255.255.1 -> 224.0.0.5 on lo0
11:11:43 route,ospf,debug,raw PACKET:
11:11:43 route,ospf,debug,raw 02 01 00 2C 0A FF FF 03 00 00 00 00 E7 9B 00 00
11:11:43 route,ospf,debug,raw 00 00 00 00 00 00 00 00 FF FF FF FF 00 0A 02 01
11:11:43 route,ospf,debug,raw 00 00 00 28 0A FF FF 01 00 00 00 00
```

List of Facility independent topics

Topic	Description
critical	Log entries marked as critical, these log entries are printed to console each time you log in.
debug	Debug log entries
error	Error messages
info	Informative log entry
packet	Log entry that shows contents from received/sent packet
raw	Log entry that shows raw contents of received/sent packet
warning	Warning message.

Topics used by various RouterOS facilities

Topic	Description
account	Log messages generated by accounting facility.
async	Log messages generated by asynchronous devices

backup	Log messages generated by backup creation facility.
bfd	Log messages generated by BFD protocol
bgp	Log messages generated by BGP protocol
calc	Routing calculation log messages.
caps	CAPsMAN wireless device management
certificate	Security certificate
dns	Name server lookup related information
ddns	Log messages generated by Dynamic DNS tool
dude	Messages related to the Dude server package The Dude tool
dhcp	DHCP client, server and relay log messages
e-mail	Messages generated by e-mail tool.
event	Log message generated at routing event. For example, new route have been installed in routing table.
firewall	Firewall log messages generated when **action=log** is set in firewall rule
gsm	Log messages generated by GSM devices
hotspot	Hotspot related log entries
igmp-proxy	IGMP Proxy related log entries
ipsec	IPSec log entries
iscsi	
isdn	
interface	
kvm	Messages related to the KVM virtual machine functionality
l2tp	Log entries generated by L2TP client and server
lte	Messages related to the LTE/4G modem configuration
ldp	LDP protocol related messages
manager	User Manager log messages.
mme	MME routing protocol messages
mpls	MPLS messages
ntp	sNTP client generated log entries
ospf	OSPF routing protocol messages
ovpn	OpenVPN tunnel messages
pim	Multicast PIM-SM related messages
ppp	ppp facility messages
pppoe	PPPoE server/client related messages
pptp	PPTP server/client related messages
radius	Log entries generated by RADIUS Client
radvd	IPv6 radv daemon log messages.

read	SMS tool messages
rip	RIP routing protocol messages
route	Routing facility log entries
rsvp	Resource Reservation Protocol generated messages.
script	Log entries generated from scripts
sertcp	Log messages related to facility responsible for "/port remote-access"
simulator	
state	DHCP Client and routing state messages.
store	Log entries generated by Store facility
smb	Messages related to the SMB file sharing system
snmp	Messages related to Simple network management protocol (SNMP) configuration
system	Generic system messages
telephony	*Obsolete! Previously used by the IP telephony package*
tftp	TFTP server generated messages

timer	Log messages that are related to timers used in RouterOS. For example bgp keepalive logs
	`12:41:40 route,bgp,debug,timer KeepaliveTimer expired`
	`12:41:40 route,bgp,debug,timer RemoteAddress=2001:470:1f09:131::1`

ups	Messages generated by UPS monitoring tool
vrrp	Messages generated VRRP
watchdog	Watchdog generated log entries
web-proxy	Log messages generated by web proxy
wireless	Wireless log entries.
write	SMS tool messages.

Examples

Logging to file

To log everything to file, add new log action:

```
/system logging action add name=file target=disk disk-file-name=log
```

and then make everything log using this new action:

```
/system logging add action=file
```

You can log only errors there by issuing command:

```
/system logging add topics=error action=file
```

This will log into files **log.0.txt** and **log.1.txt**.

You can specify maximum size of file in lines by specifying *disk-lines-per-file*. **<file>.0.txt** is active file were new logs are going to be appended and once it size will reach maximum it will become **<file>.1.txt**, and new empty **<file>.0.txt** will be created.

You can log into USB flashes or into *MicroSD/CF* (on Routerboards) by specifying it's directory name before file name. For example, if you have accessible usb flash as **usb1** directory under */files*, you should issue following command:

```
/system logging action add name=usb target=disk disk-file-name=usb1/log
```


Logging entries from files will be stored back in the memory after reboot.

Netwatch

Summary

Netwatch monitors the state of hosts on the network. Monitoring can be done with the following probe types:
1) ICMP - pings to a specified IP address - hosts, with an option to adjust threshold values
2) Simple - uses ping, without use of advanced metrics
3) TCP conn, to test the TCP connection
4) HTTP GET/HTTPS GET, request against a server you are monitoring

For each entry in the Netwatch table, you can specify an IP address, ping interval, and console scripts. The main advantage of Netwatch is its ability to issue arbitrary console commands on host state changes.

Since 7.4, Netwatch functionality has been expanded, prior versions only support simple ICMP probes. While upgrading to the new version, old Netwatch entries will be unchanged, reporting probe type "simple" - preserving the same functionality.

Default Netwatch values are always used - **even if they were not defined** by the user. Make sure to check the "status" page of the probe to see if the default thresholds are appropriate for your use case. Default threshold values can be found under the "probe options" section on this page.

Properties

Sub-menu: `/tool/netwatch`

Property	Description
host (Default:"")	The IP address of the server to be probed. Formats: - *ipv4* - *ipv4@vrf* - *ipv6* - *ipv6@vrf* - *ipv6-linklocal%interface*
type (icmp/ *tcp-conn|http-get|simple;* Default: **simple**)	Type of the probe: - icmp - (ping-style) series of ICMP request-response with statistics - tcp-conn - test TCP connection (3-way handshake) to a server specified by IP and port - http-get - do an HTTP Get request and test for a range of correct replies - simple - simplified ICMP probe, with fewer options than "ICMP" type, used for backward compatibility with the older Netwatch version
interval (Default: **10s**)	The time interval between probe tests

timeout (Default: **3s**)	Max time limit to wait for a response
src-address (Default:**""**)	Source IP address which the Netwatch will try to use in order to reach the host. If address is not present, then the host will be considered as "down".
start-delay (Default: **3s**)	Time to wait before starting probe (on add, enable, or system start)
startup-delay (Default: **5m**)	Time to wait until starting Netwatch probe after system startup
up-script (Default:"")	Script to execute on the event of probe state change 'fail' --> 'OK'
down-script (Default:"")	Script to execute on the event of probe state change 'OK' --> 'fail'
test-script (Default:"")	Script to execute at the end of every probe test

Netwatch executes scripts as *sys user, so any defined global variable in the Netwatch script will not be readable by for an example a scheduler or other users

> Netwatch is limited to *read,write,test,reboot* script policies. If the owner of the script does not have enough permissions to execute a certain command in the script, then the script will not be executed. If the script has greater policies than *read,write,test,reboot* - then the script will not be executed as well, make sure your scripts do not exceed the mentioned policies.

It is possible to disable permission checking for RouterOS scripts under */system/scripts* menu. This is useful when Netwatch does not have enough permissions to execute a script, though this decreases overall security. It is recommended to assign proper permissions to a script instead.

Type-specific options

All config options specific to one probe type (e.g. icmp's packet-interval) are ignored for other probe types (tcp-conn, http-get).

ICMP probe options

Property	Description
packet-interval (Default: **50ms**)	The time between ICMP-request packet send
packet-count (Default: **10**)	Total count of ICMP packets to send out within a single test
packet-size (Default: **54** (IPv4) or **54** (IPv6))	The total size of the IP ICMP packet
thr-rtt-max (Default**: 1s**)	Fail threshold for rtt-max (a value above thr-max is a probe fail)
thr-rtt-avg (Default: **100ms**)	Fail threshold for rtt-avg
thr-rtt-stdev (Default: **250ms**)	Fail threshold for rtt-stdev
thr-rtt-jitter (Default: **1s**)	Fail threshold for rtt-jitter
thr-loss-percent (Default: **85.0%**)	Fail threshold for loss-percent
thr-loss-count (Default: **4294967295**(max))	Fail threshold for loss-count

TCP-CONNECT/HTTP-GET probe options

Property	Description
port (Default: **80**)	TCP port (for both tcp-conn and http-get probes)

TCP-CONNECT pass-fail criteria

Property	Description

thr-tcp-conn-time (Default: **0 0:05...00:30**)	Fail threshold for tcp-connect-time, the configuration uses microseconds, if the time unit is not specified (s/m/h), log and status pages display the same value in milliseconds.

HTTP-GET probe pass/fail criteria

Property	Description
thr-http-time (Default: **10s**)	Fail threshold for http-resp-time
http-code-min (Default: **100**)	OK/fail criteria for HTTP response code.
http-code-max (Default: **299**)	Response in the range [`http-code-min` , `http-code-max`] is a probe pass/OK; outside - a probe fail. See mozilla-http-status or rfc7231

Probe statistics/variables

You can view statistics and use these variables in scripting, keep in mind that variables containing "-" must be written like this, for example, "done-tests" would be $"done-tests"

Generic:

Property	Description
name	user added name for Netwatch entry
comment	user added comment
host	host that was probed
type	probe type
interval	interval
timeout	timeout
since	The last time the status change happened
status	current status of probe
done-tests	total count of probe tests already done so far
failed-tests	count of failed probe tests

ICMP:

Property	Description
sent-count	ICMP packets sent out
response-count	Matching/valid ICMP packet responses received
loss-count	number of lost packets
loss-percent	number of lost packets in percent
rtt-avg	mean value of rtt (packet roundtrip time)
rtt-min	min rtt
rtt-max	max rtt

rtt-jitter	jitter (= max - min) of rtt
rtt-stdev	standard deviation of rtt

TCP:

Property	Description
tcp-connect-time	time taken to establish a TCP connection

HTTP:

Property	Description
http-status-code	HTTP response status code (200 OK, 404 Not Found, etc.). See mozilla-http-status or RFC7231

HTTPS:

Property	Description
http-status-code	HTTP response status code (200 OK, 404 Not Found, etc.). See mozilla-http-status or RFC7231

Logs

On each probe's OK/fail state change:

- probe identification info and OK->fail or fail->OK is printed to info level
- detailed probe stats and config is printed to debug level

Status

Command */tool/netwatch/print* will show the current status of Netwatch and **read-only** properties:

- since - Indicates when a state of the host changed last time;
- status - Shows the current status of the host;
- host - address being monitored

Quick Example

Here we will use a simple ICMP check to host with IP 8.8.8.8:

```
[admin@MikroTik] > /tool/netwatch add host=8.8.8.8 interval=30s up-script=":log info \"Ping to 8.8.8.8
successful\""
```

Afterward, in the logging section we can see Netwatch executed script:

```
[admin@MikroTik] > log print where message~"8.8.8.8"
08:03:26 script,info Ping to 8.8.8.8 successful
```

Packet Sniffer

Introduction

A packet sniffer is a tool that can capture and analyze packets that are going to, leaving, or going through the router. Packet sniffing is very useful when you diagnose networks or protect against security attacks over networks.

Unicast traffic between Wireless clients with client-to-client forwarding enabled will not be visible to the sniffer tool.

Packets that are processed with hardware offloading enabled bridge will not be visible (flooded packets like unknown unicast, broadcast, and multicast traffic might be visible).

Sniffer catches packets before they go into the firewall (direction=rx) or after they leave it (direction=tx).

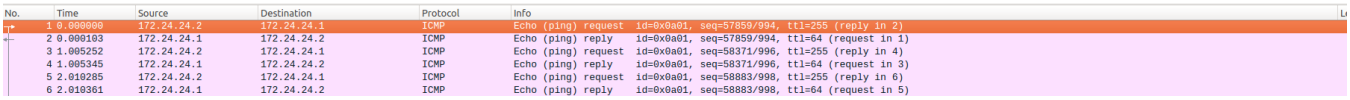
Packet Sniffer configuration

RouterOS embedded sniffer allows you to capture packets based on various protocols.

In the following example, we will configure the sniffer to match packets going through the ether1 interface:

```
[admin@MikroTik] > /tool/sniffer/start interface=ether1
[admin@MikroTik] > /tool/sniffer/stop
[admin@MikroTik] > /tool/sniffer/save file-name=/flash/test.pcap
MikroTik] > file print where name~"test"
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME        TYPE  SIZE  CREATION-TIME
9  flash/test.pcap  file  3696  dec/04/2019 10:48:16
```

You can download captured packets from a file section. Then you can use a packet analyzer such as Wireshark to analyze a file:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.24.24.2	172.24.24.1	ICMP	Echo (ping) request id=0x0a01, seq=57859/994, ttl=255 (reply in 2)
2	0.000103	172.24.24.1	172.24.24.2	ICMP	Echo (ping) reply id=0x0a01, seq=57859/994, ttl=64 (request in 1)
3	1.005252	172.24.24.2	172.24.24.1	ICMP	Echo (ping) request id=0x0a01, seq=58371/996, ttl=255 (reply in 4)
4	1.005345	172.24.24.1	172.24.24.2	ICMP	Echo (ping) reply id=0x0a01, seq=58371/996, ttl=64 (request in 3)
5	2.010285	172.24.24.2	172.24.24.1	ICMP	Echo (ping) request id=0x0a01, seq=58883/998, ttl=255 (reply in 6)
6	2.010361	172.24.24.1	172.24.24.2	ICMP	Echo (ping) reply id=0x0a01, seq=58883/998, ttl=64 (request in 5)

Please note that sniffed packets will be available for 10 minutes, if you need them permenately, set a "file-name" to save the directly or issue a "save" command as described previously.

Sub-menu: `/tool sniffer`

Property	Description
file-limit (*integer 10..4294967295[KiB]*; Default: **10 00KiB**)	File size limit. Sniffer will stop when a limit is reached.
file-name (*string*; Default:)	Name of the file where sniffed packets will be saved.

filter-cpu (*integer*; Default:)	CPU core used as a filter.
filter-ip-address (*ip/mask[,ip/mask] (max 16 items)*; Default:)	Up to 16 IP addresses used as a filter.
filter-dst-ip-address (*ip/mask[,ip/mask] (max 16 items)*; Default:)	Up to 16 IP destination addresses used as a filter.
filter-src-ip-address (*ip/mask[,ip/mask] (max 16 items)*; Default:)	Up to 16 IP source addresses used as a filter.
filter-ipv6-address (*ipv6/mask[,ipv6/mask] (max 16 items)*; Default:)	Up to 16 IPv6 addresses used as a filter.
filter-dst-ipv6-address (*ipv6/mask[,ipv6/mask] (max 16 items)*; Default:)	Up to 16 IPv6 destination addresses used as a filter.
filter-src-ipv6-address (*ipv6/mask[,ipv6/mask] (max 16 items)*; Default:)	Up to 16 IPv6 source addresses used as a filter.
filter-mac-address (*mac/mask[,mac/mask] (max 16 items)*; Default:)	Up to 16 MAC addresses and MAC address masks used as a filter.
filter-dst-mac-address (*mac/mask[,mac/mask] (max 16 items)*; Default:)	Up to 16 MAC destination addresses and MAC address masks used as a filter.
filter-src-mac-address (*mac/mask[,mac/mask] (max 16 items)*; Default:)	Up to 16 MAC source addresses and MAC address masks used as a filter.
filter-port (*[!]port[,port] (max 16 items)*; Default:)	Up to 16 comma-separated ports used as a filter. A list of predefined port names is also available, like ssh and telnet.
filter-dst-port (*[!]port[,port] (max 16 items)*; Default:)	Up to 16 comma-separated destination ports used as a filter. A list of predefined port names is also available, like ssh and telnet.
filter-src-port (*[!]port[,port] (max 16 items)*; Default:)	Up to 16 comma-separated source ports used as a filter. A list of predefined port names is also available, like ssh and telnet.

filter-ip-protocol (*[!]protocol[,protocol] (max 16 items)*; Default:)	Up to 16 comma-separated IP/IPv6 protocols used as a filter. IP protocols (instead of protocol names, protocol numbers can be used): • **ipsec-ah** - IPsec AH protocol • **ipsec-esp** - IPsec ESP protocol • **ddp** - datagram delivery protocol • **egp** - exterior gateway protocol • **ggp** - gateway-gateway protocol • **gre** - general routing encapsulation • **hmp** - host monitoring protocol • **idpr-cmtp** - idpr control message transport • **icmp** - internet control message protocol • **icmpv6** - internet control message protocol v6 • **igmp** - internet group management protocol • **ipencap** - ip encapsulated in ip • **ipip** - ip encapsulation • **encap** - ip encapsulation • **iso-tp4** - iso transport protocol class 4 • **ospf** - open shortest path first • **pup** - parc universal packet protocol • **pim** - protocol independent multicast • **rspf** - radio shortest path first • **rdp** - reliable datagram protocol • **st** - st datagram mode • **tcp** - transmission control protocol • **udp** - user datagram protocol • **vmtp** - versatile message transport • **vrrp** - virtual router redundancy protocol • **xns-idp** - xerox xns idp • **xtp** - xpress transfer protocol
filter-mac-protocol (*[!]protocol[,protocol] (max 16 items)*; Default:)	Up to 16 comma separated entries used as a filter. Mac protocols (instead of protocol names, protocol number can be used): • **802.2** - 802.2 Frames (0x0004) • **arp** - Address Resolution Protocol (0x0806) • **homeplug-av** - HomePlug AV MME (0x88E1) • **ip** - Internet Protocol version 4 (0x0800) • **ipv6** - Internet Protocol Version 6 (0x86DD) • **ipx** - Internetwork Packet Exchange (0x8137) • **lldp** - Link Layer Discovery Protocol (0x88CC) • **loop-protect** - Loop Protect Protocol (0x9003) • **mpls-multicast** - MPLS multicast (0x8848) • **mpls-unicast** - MPLS unicast (0x8847) • **packing-compr** - Encapsulated packets with compressed IP packing (0x9001) • **packing-simple** - Encapsulated packets with simple IP packing (0x9000) • **pppoe** - PPPoE Session Stage (0x8864) • **pppoe-discovery** - PPPoE Discovery Stage (0x8863) • **rarp** - Reverse Address Resolution Protocol (0x8035) • **service-vlan** - Provider Bridging (IEEE 802.1ad) & Shortest Path Bridging IEEE 802.1aq (0x88A8) • **vlan** - VLAN-tagged frame (IEEE 802.1Q) and Shortest Path Bridging IEEE 802.1aq with NNI compatibility (0x8100)
filter-stream (*yes* | *no*; Default: **yes**)	Sniffed packets that are devised for the sniffer server are ignored.
filter-size (*integer[-integer]:0..65535*; Default:)	Filters packets of specified size or size range in bytes.
filter-direction (*any* | *rx* | *tx*; Default:)	Specifies which direction filtering will be applied.
filter-interface (*all* | *name*; Default: **all**)	Interface name on which sniffer will be running. **all** indicates that the sniffer will sniff packets on all interfaces.
filter-operator-between-entries (*and* | *or*; Default: **or**)	Changes the logic for filters with multiple entries.

filter-vlan (*integer[,integer]:0..4095*; Default:)	Up to 16 VLAN IDs used as a filter.
memory-limit (*integer 10..4294967295[KiB]*; Default: **100KiB**)	Memory amount used to store sniffed data.
memory-scroll (*yes* | *no*; Default: **yes**)	Whether to rewrite older sniffed data when the memory limit is reached.
only-headers (*yes* | *no*; Default: **no**)	Save in the memory only the packet's headers, not the whole packet.
streaming-enabled (*yes* | *no*; Default: **no**)	Defines whether to send sniffed packets to the streaming server.
streaming-server (*IP*; Default: **0.0.0.0**)	Tazmen Sniffer Protocol (TZSP) stream receiver.

The `file-size` limit should not be configured more than available free memory!

Packet Sniffer Quick Mode

The quick mode will display results as they are filtered out with a limited-size buffer for packets. There are several attributes that can be set up for filtering. If no attributes are set current configuration will be used.

```
[admin@MikroTik] > /tool/sniffer/quick ip-protocol=icmp
Columns: INTERFace, TIME, NUm, DIr, SRC-MAC, DST-MAC, SRC-ADDRESS, DST-ADDRESS, PROTOCOl, SIze, Cpu, FP
INTERF  TIME    NU  DI  SRC-MAC            DST-MAC            SRC-ADDRESS     DST-ADDRESS     PROTOCO  SI  C  FP
ether7  35.472  79  <-  6C:3B:6B:ED:83:69  6C:3B:6B:ED:81:83  10.155.126.252  10.155.126.253  ip:icmp  70  7  no
ether7  35.472  80  ->  6C:3B:6B:ED:81:83  6C:3B:6B:ED:83:69  10.155.126.253  10.155.126.252  ip:icmp  70  7  no
ether1  35.595  81  <-  6C:3B:6B:ED:83:63  6C:3B:6B:ED:81:7D  172.24.24.2     172.24.24.1     ip:icmp  70  4  no
ether1  35.595  82  ->  6C:3B:6B:ED:81:7D  6C:3B:6B:ED:83:63  172.24.24.1     172.24.24.2     ip:icmp  70  4  no
ether7  36.457  83  <-  6C:3B:6B:ED:83:69  6C:3B:6B:ED:81:83  10.155.126.252  10.155.126.253  ip:icmp  70  7  no
ether7  36.457  84  ->  6C:3B:6B:ED:81:83  6C:3B:6B:ED:83:69  10.155.126.253  10.155.126.252  ip:icmp  70  7  no
ether1  36.6    85  <-  6C:3B:6B:ED:83:63  6C:3B:6B:ED:81:7D  172.24.24.2     172.24.24.1     ip:icmp  70  4  no
ether1  36.6    86  ->  6C:3B:6B:ED:81:7D  6C:3B:6B:ED:83:63  172.24.24.1     172.24.24.2     ip:icmp  70  4  no
```

Traffic-Generator packets will not be visible using the packet sniffer on the same interface unless the *fast-path* parameter is set.

Packet Sniffer Protocols

In this submenu, you can see all sniffed protocols and their share of the whole sniffed amount.

```
[admin@MikroTik] /tool sniffer protocol> print
 # PROTOCOL IP-PROTOCOL PORT                                    PACKETS      BYTES       SHARE
 0 802.2                                                            1         60       0.05%
 1 ip                                                             215     100377      99.04%
 2 arp                                                              2        120       0.11%
 3 ipv6                                                             6        788       0.77%
 4 ip       tcp                                                   210      99981      98.65%
 5 ip       udp                                                     3        228       0.22%
 6 ip       ospf                                                    2        168       0.16%
 7 ip       tcp         8291 (winbox)                             210      99981      98.65%
 8 ip       tcp         36771                                     210      99981      98.65%
 9 ip       udp         646                                         3        228       0.22%
```

Packet Sniffer Host

The submenu shows the list of hosts that were participating in the data exchange you've sniffed.

```
[admin@MikroTik] /tool sniffer host> print
 # ADDRESS         RATE                PEEK-RATE           TOTAL
 0 10.5.101.3      0bps/0bps           0bps/720bps         0/90
 1 10.5.101.10     0bps/0bps           175.0kbps/19.7kbps  61231/7011
 2 10.5.101.13     0bps/0bps           0bps/608bps         0/76
 3 10.5.101.14     0bps/0bps           0bps/976bps         0/212
 4 10.5.101.15     0bps/0bps           19.7kbps/175.0kbps  7011/61231
 5 224.0.0.2       0bps/0bps           608bps/0bps         76/0
 6 224.0.0.5       0bps/0bps           1440bps/0bps        302/0
```

Packet Sniffer Connections

Here you can get a list of the connections that have been watched during the sniffing time.

```
[admin@MikroTik] tool sniffer connection> print
Flags: A - active
  #   SRC-ADDRESS       DST-ADDRESS             BYTES     RESENDS   MSS
  0 A 10.0.0.241:1839   10.0.0.181:23 (telnet)  6/42      60/0      0/0
  1 A 10.0.0.144:2265   10.0.0.181:22 (ssh)     504/252   504/0     0/0
```

Ping

Summary

Ping uses the Internet Control Message Protocol (ICMP) Echo messages to determine if a remote host is active or inactive and to determine the round-trip delay when communicating with it. Ping tool sends ICMP (type 8) message to the host and waits for the ICMP echo-reply (type 0). The interval between these events is called a round trip. If the response (that is called pong) has not come until the end of the interval, we assume it has timed out. The second significant parameter reported is TTL (Time to Live). Is decremented at each machine in which the packet is processed. The packet will reach its destination only when the TTL is greater than the number of routers between the source and the destination.

Quick Example

RouterOS Ping tool allows you to configure various additional parameters like:

- arp-ping;
- address;
- src-address;
- count;
- dscp;
- interface;
- interval;
- routing-table;
- size;
- ttl;

Let's take a look ar very simple example:

```
[admin@MikroTik] > /tool/ping address=10.155.126.252 count=5 interval=200ms
  SEQ HOST                                     SIZE TTL TIME
STATUS

    0 10.155.126.252                             56  64 0ms
    1 10.155.126.252                             56  64 0ms
    2 10.155.126.252                             56  64 0ms
    3 10.155.126.252                             56  64 0ms
    4 10.155.126.252                             56  64 0ms
    sent=5 received=5 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```

The same we can achieve with more shorter CLI command:

```
[admin@MikroTik] > /ping 10.155.126.252 count=5 interval=50ms
  SEQ HOST                                     SIZE TTL TIME
STATUS

    0 10.155.126.252                             56  64 0ms
    1 10.155.126.252                             56  64 0ms
    2 10.155.126.252                             56  64 0ms
    3 10.155.126.252                             56  64 0ms
    4 10.155.126.252                             56  64 0ms
    sent=5 received=5 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```

It is also possible to ping multicast address to discover all hosts belonging to multicast group:

```
[admin@MikroTik] > /ping ff02::1
HOST                                     SIZE  TTL TIME  STATUS
fe80::20c:42ff:fe49:fceb                 56    64  1ms   echo reply
fe80::20c:42ff:fe72:a1b0                 56    64  1ms   echo reply
fe80::20c:42ff:fe28:7945                 56    64  1ms   echo reply
fe80::21a:4dff:fe5d:8e56                 56    64  3ms   echo reply
    sent=1 received=4 packet-loss=-300% min-rtt=1ms avg-rtt=1ms max-rtt=3ms
```

Ping by DNS name:

```
[admin@MikroTik]  > /ping www.google.com count=5 interval=50ms
  SEQ HOST                                     SIZE TTL TIME
STATUS

    0 216.58.207.228                             56  51 14ms
    1 216.58.207.228                             56  51 13ms
    2 216.58.207.228                             56  51 13ms
    3 216.58.207.228                             56  51 13ms
    4 216.58.207.228                             56  51 13ms
    sent=5 received=5 packet-loss=0% min-rtt=13ms avg-rtt=13ms max-rtt=14ms
```

When you use the domain name and CLI for ping, router DNS will be used to resolve the address. When you use the Winbox Tools/Ping, your computer's DNS will be used to resolve the given address.

MAC Ping

This submenu allows enabling the mac ping server.

When mac ping is enabled, other hosts on the same broadcast domain can use the ping tool to ping mac address:

```
[admin@MikroTik]  > /tool mac-server ping set enabled=yes
```

Ping MAC address:

```
[admin@MikroTik]  > /ping 00:0C:42:72:A1:B0
HOST                                     SIZE  TTL TIME  STATUS
00:0C:42:72:A1:B0                        56        0ms
00:0C:42:72:A1:B0                        56        0ms
    sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```

Profiler

Summary

The profiler tool shows CPU usage for each process running in RouterOS. It helps to identify which process is using most of the CPU resources. Watch our video about this feature.

```
[admin@MikroTik] > /tool/profile
```

On multi-core systems, the tool allows specifying per core CPU usage.

"CPU" parameter allows specifying integer number which represents a core or two of predefined values **all** and **total:**

- total - this value sets to show the sum of all core usages;
- all - value sets to show CPU usages separately for every available core

In the following example we will take a look at both predefined values:

```
[admin@MikroTik] > /tool/profile cpu=all
NAME              CPU         USAGE
ethernet          1           0%
kvm               0           0%
kvm               1           4.5%
management        0           0%
management        1           0.5%
idle              0           100%
idle              1           93%
profiling         0           0%
profiling         1           2%

[admin@MikroTik] > /tool profile cpu=total
NAME              CPU         USAGE
ethernet          all         0%
console           all         0%
kvm               all         2.7%
management        all         0%
idle              all         97.2%
profiling         all         0%
bridging          all         0%
```

Classifiers

RouterOS processes are classified by type and the CPU usage for each type is displayed separately for ease of debugging.

Property	Description
backup	Backup service
bfd	BFD service
bgp	BGP service
bridging	Bridging service
btest	Bandwidth test.
certificate	Certificate service

console	Console
container	combined container usage
dhcp	DHCP-Server and DHCP-Client services
disk	storage-related services
dns	DNS-related services
dude	The Dude package services
e-mail	e-mail tool
encrypting	encrypting processes
eoip	EoIP
ethernet	Ethernet-related properties like link speed, auto-negotiation, duplex mode, monitor a transceiver diagnostic information, etc.
fetcher	Fetch tool
firewall	Firewall-related processes
firewall-mgmt	Firewall Management: Filtering, NAT, Mangle
flash	storage-related services
ftp	FTP Service
gps	GPS Service
graphing	Graphing tool
gre	GRE
health	system monitoring, workd health
hotspot	Hotspot service
idle	Free CPU resources
igmp-proxy	IGMP Proxy service
internet-detect	Detect Internet tool
ip-pool	IP Pool service
ipsec	IPsec service
kvm	KVM virtual machine functionality
l7-matcher	L7 matcher
lcd	LCD Interfaces system
ldp	Label Distribution Protocol (LDP)
logging	Logging system
management	different subsystems: scheduler, networking, file management, etc.
mpls	MPLS-related features
networking	common set of services included in the networking
ntp	NTP service
ospf	OSPF service
pim	Protocol Independent Multicast

profiling	Profiler service
queue-mgmt	Queues: Simple queues, Queue tree, Queue types
queuing	Intermediate Queuing
radius	RADIUS service
radv	IPv6 radv daemon log messages service
remote-access	accessing the device directly without logging into RouterOS
rip	Routing Information Protocol
routing	Routing-related services
serial	serial console and terminal tool
sniffing	packet Sniffer tool
snmp	SNMP
socks	Socket Secure
spi	storage-related services
ssh	SSH Server
ssl	SSL
supout.rif	supout.rif file generation
telnet	Telnet service
tftp	TFTP service
traffic-accounting	Traffic-Flow log system
traffic-flow	Traffic-Flow system
unclassified	processes or services that are not defined by this classifier
upnp	UPnP protocol
usb	USB features
user-manager	User Manager service
vrrp	VRRP
web-proxy	Web Proxy
winbox	Winbox
wireguard	Wireguard
wireless	common set of services using Wireless systems
www	Webfig HTTP service
zerotier	ZeroTier

Resource

Summary

General

```
/system resource
```

The general resource menu shows overall resource usage and router statistics like uptime, memory usage, disk usage, version, etc.

It also has several sub-menus for more detailed hardware statistics like PCI, IRQ, and USB.

```
[admin@MikroTik] > system/resource/print
                   uptime: 29s
                  version: 7.11.2 (stable)
               build-time: Aug/31/2023 13:55:47
         factory-software: 7.6
              free-memory: 94.2MiB
             total-memory: 224.0MiB
                      cpu: ARM
                cpu-count: 2
            cpu-frequency: 800MHz
                 cpu-load: 2%
           free-hdd-space: 93.5MiB
          total-hdd-space: 128.5MiB
  write-sect-since-reboot: 85
         write-sect-total: 222100
               bad-blocks: 0%
        architecture-name: arm
               board-name: hAP ax lite LTE6
                 platform: MikroTik
```

Properties

All properties are read-only

Property	Description
architecture-name (*string*)	CPU architecture
bad-blocks (*percent*)	Shows percentage of bad blocks on the NAND.
board-name (*string*)	RouterBOARD model name
build-time (*string*)	Installed RouterOS version build-time
cpu (*string*)	CPU model that is on the board
cpu-count (*integer*)	Number of CPUs present on the system. Each core is a separate CPU, Intel HT is also a separate CPU.
cpu-frequency (*string*)	Current CPU frequency
cpu-load (*percent*)	Percentage of used CPU resources. Combines all CPUs. Per-core CPU usage can be seen in CPU submenu
factory-software (*string*)	Minimal RouterOS version
free-hdd-space (*string*)	Free space on hard drive or NAND
free-memory (*string*)	The unused amount of RAM
platform (*string*)	Platform name
total-hdd-space (*string*)	Size of the hard drive or NAND

total-memory (*string*)	Amount of installed RAM
uptime (*time*)	Time interval passed since boot-up
version (*string*)	Installed RouterOS version number
write-sect-since-reboot (*integer*)	A number of sector writes in HDD or NAND since the router was last time rebooted
write-sect-total (*integer*)	A number of sector writes in total

CPU

```
/system resource cpu
```

This submenu shows per-cpu usage, as long as IRQ and Disk usage.

```
[admin@RB1100test] /system resource cpu> print
CPU LOAD IRQ DISK
0 5% 0% 0%
[admin@RB1100test] /system resource cpu>
```

Properties

Read-only properties

Property	Description
cpu (*integer*)	Identification number of CPU which usage is shown.
load (*percent*)	CPU usage in percents
irq (*percent*)	IRQ usage in percents
disk (*percent*)	Disk usage in percents

IRQ

```
/system resource irq
```

The menu shows all used IRQs on the router. It is possible to set up IRQ load balancing on multicore systems by assigning IRQ to a specific core. IRQ assignments are done by hardware and cannot be changed from RouterOS. For example, if all Ethernets are assigned to one IRQ, then you have to deal with hardware: upgrade motherboards BIOS, reassign IRQs manually in BIOS, if none of the above helps then change the hardware.

Properties

Property	Description
cpu (*auto* | *integer*; Default:)	Specifies which CPU is assigned to the IRQ. • **auto** - pick CPU based on a number of interrupts. Uses NAPI to optimize interrupts.

Read-only properties

Property	Description
active-cpu (*integer*)	Shows active CPU in multicore systems.

count (*integer*)	A number of interrupts. On ethernet interfaces interrupt=packet.
irq (*integer*)	IRQ identification number
users (*string*)	Process assigned to IRQ

IRQ Load Balancing

USB

```
/system resource usb
```

This menu displays all available USB controllers on the board. The menu is available only if at least one USB controller is present.

```
[admin@MikroTik] /system resource usb> print detail
0 device="2:1" name="RB400 EHCI" serial-number="rb400_usb" vendor-id="0x1d6b"
device-id="0x0002" speed="480 Mbps" ports=2 usb-version="2.00"
1 device="1:1" name="RB400 OHCI" serial-number="rb400_usb" vendor-id="0x1d6b"
 device-id="0x0001" speed="12 Mbps" ports=2 usb-version="1.10"
```

Properties

Property	Description
device (*string*)	
device-id (*hex*)	Hexadecimal device ID
name (*string*)	Descriptive name of the device retrieved from the driver
ports (*integer*)	How many ports are supported by the USB controller
serial-number (*string*)	
speed (*string*)	Max USB speed that can be used (480Mbps for USBv2 and 12Mbps for USBv1)
usb-version (*string*)	Identifies max supported USB version
vendor (*string*)	Device manufacturer's name.
vendor-id (*hex*)	Hexadecimal vendor ID

PCI

```
/system resource pci
```

PCI submenu shows the information about all PCI devices on the board

```
[admin@RB1100test] /system resource pci> print
# DEVICE VENDOR NAME IRQ
0 06:00.0 Attansic Technology Corp. unknown device (rev: 192) 18
1 05:00.0 Freescale Semiconductor Inc MPC8544 (rev: 17) 0
2 04:00.0 Attansic Technology Corp. unknown device (rev: 192) 17
3 03:00.0 Freescale Semiconductor Inc MPC8544 (rev: 17) 0
4 02:00.0 Attansic Technology Corp. unknown device (rev: 192) 16
5 01:00.0 Freescale Semiconductor Inc MPC8544 (rev: 17) 0
6 00:00.0 Freescale Semiconductor Inc MPC8544 (rev: 17) 0
```

Properties

All properties are read-only

Property	Description
category (*string*)	PCI device type, for example, *ethernet controller*
device (*string*)	
device-id (*hex*)	Hexadecimal device ID
io (*hex-hex*)	I/O memory range
irq (*integer*)	IRQ assigned to the device
memory (*hex-hex*)	Memory range
name (*string*)	Descriptive name of the device retrieved from the driver
vendor (*string*)	Device manufacturer's name.
vendor-id (*hex*)	Hexadecimal vendor ID

S+RJ10 general guidance

Summary

MikroTik S+RJ10 is a unique 6-speed RJ-45 SFP+ module based on a Marvell 88X3310P transceiver. It offers up to 10 Gbps speeds using twisted-pair copper cables. All the current MikroTik devices with an SFP+ cage support the S+RJ10 module. This article serves as a guideline of S+RJ10 usage in MikroTik devices with both passive and active cooling.

General Guidance

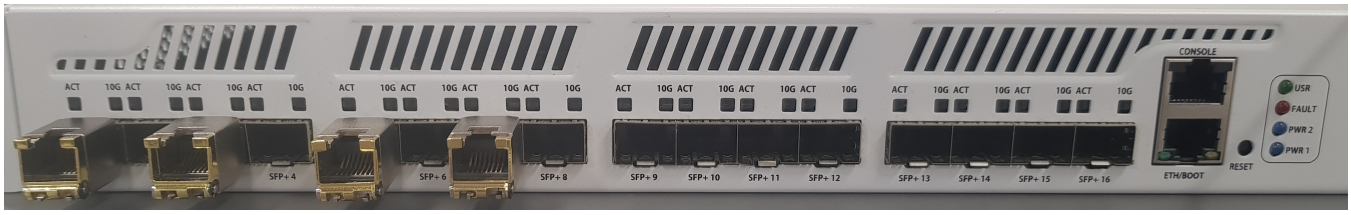
Product specification

The average power consumption of the transceiver is 2.7 W (10GBASE-T, 30 m link) which is relatively high compared with the S+85DLC03D optical module with a maximum 0.8W power consumption. The operating temperature is 0 to +70 C, but the transceiver itself can heat up to 90 C.

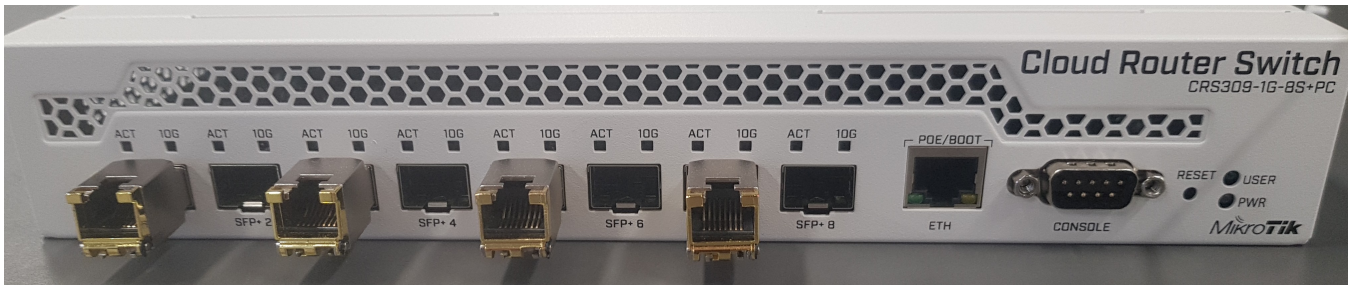
S+RJ10 Positioning in devices

Due to high operating temperatures, it is recommended to use S+RJ10 transceivers while an optical transceiver or an unused SFP+ interface is in between them. Take a look at the transceivers capable distance comparison table.

As mentioned, S+RJ10 heat up more than regular transceivers, and keeping them side by side can result in overheating, especially in devices with 4 linear SFP cages. It is recommended to place S+RJ10 in every second interface while keeping an optical transceiver or an empty port in between them.



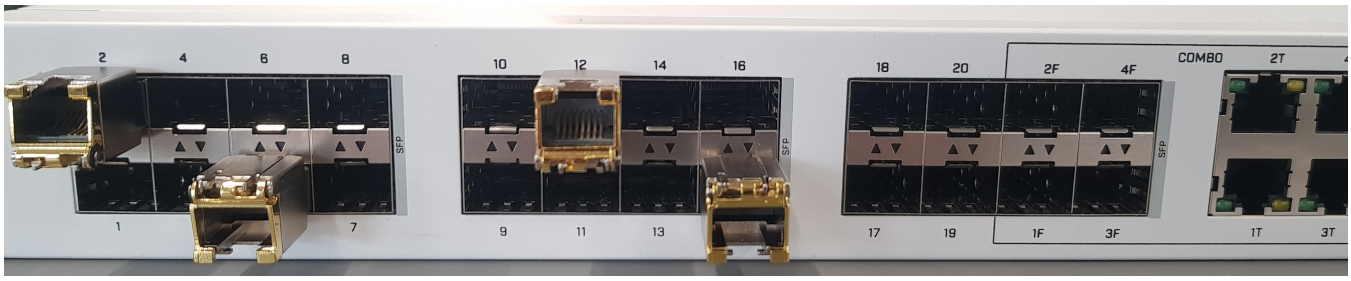
Even when using devices that come with separated SFP+ cages, for example, CRS309-1G-8S+, it is still not recommended to deploy the S+RJ10 transceivers beside each other. Use S+RJ10 in every second interface to avoid transceivers overheating which may cause unpredictable behavior.



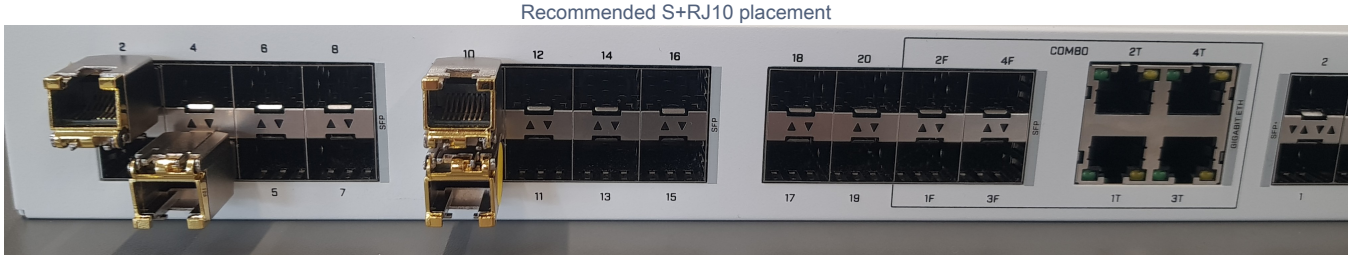
Recommended S+RJ10 placement

It is not recommended to place transceivers side by side

Devices that come with 4 or 8-block SFP+ cages are not exceptions. It is recommended to use one S+RJ10 transceiver per 4xSFP+ cage block and avoid placing them side by side. Keep at least one vertical row empty(without S+RJ10) after plugging the S+RJ10 transceiver.



Recommended S+RJ10 placement



We do not recommend to place transceivers side by side

Using the S+RJ10 Side by Side or with passive cooling devices

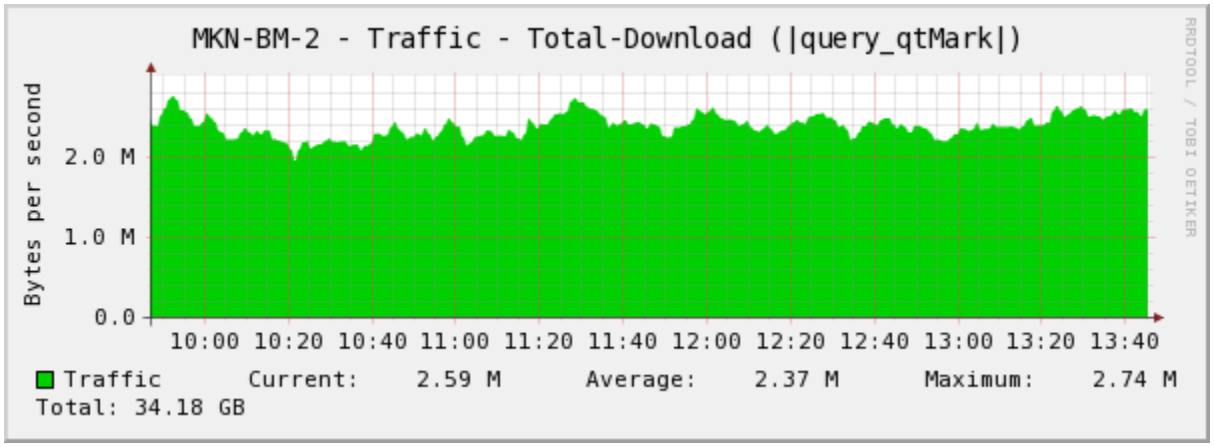
There might be situations when it is not possible to use the recommended layout of the transceivers. In such cases where two or more S+RJ10 transceivers are plugged in beside one another or modules are used in passive cooling devices, the network administrator has to ensure additional cooling. The airflow around the device should be increased or the overall ambient temperature should be lowered to keep the temperature of the transceivers within the recommended range.

SNMP

Overview

Simple Network Management Protocol (SNMP) is an Internet-standard protocol for managing devices on IP networks. SNMP can be used to graph various data with tools such as CACTI, MRTG, or The Dude.

SNMP write support is only available for some OIDs. For supported OIDs SNMP v1, v2 or v3 write is supported.



SNMP will respond to the query on the interface SNMP request was received from forcing responses to have same source address as request destination sent to the router

SNMP tool collects data from different services running on the system. If, for some reason, communication between SNMP and some service is taking longer time than expected (30 seconds per service, 5 minutes for routing service), you will see a warning in the log stating "timeout while waiting for program" or "SNMP did not get OID data within expected time, ignoring OID". After that, this service will deny SNMP requests for a while before even trying to get requested data again.

This error has nothing to do with SNMP service itself. In most cases, such an error is printed when some slow or busy service is monitored through SNMP, and quite often, it is a service that should not be monitored through SNMP, and proper solution in such cases is to skip such OIDs on your monitoring tool.

Quick Configuration

To enable SNMP in RouterOS:

```
[admin@MikroTik] /snmp> print
enabled: no
contact:
location:
engine-id:
trap-community: (unknown)
trap-version: 1
[admin@MikroTik] /snmp> set enabled yes
```

You can also specify administrative contact information in the above settings. All SNMP data will be available to communities configured in the *community* menu.

General Properties

Sub-menu: `/snmp`

This sub menu allows to enable SNMP and to configure general settings.

Property	Description
contact (*string*; Default: **""**)	Contact information
enabled (*yes* | *no*; Default: **no**)	Used to disable/enable SNMP service
engine-id (*string*; Default: **""**)	For SNMP v3, used as part of the identifier. You can configure the suffix part of the engine id using this argument. If the SNMP client is not capable to detect set engine-id value then this prefix hex has to be used 0x80003a8c04
location (*string*; Default: **""**)	Location information
trap-community (*string*; Default: **public**)	Which communities configured in the *community* menu to use when sending out the trap.
trap-generators (*interfaces* | *start-trap*; Default:)	What action will generate traps: • interfaces - interface changes; • start-trap - SNMP server starting on the router
trap-interfaces (*string* | *all*; Default:)	List of interfaces that traps are going to be sent out.
trap-target (*list of IP/IPv6*; Default: **0.0.0.0**)	IP (IPv4 or IPv6) addresses of SNMP data collectors that have to receive the trap
trap-version (*1|2|3*; Default: **1**)	A version of SNMP protocol to use for trap
src-address (*IPv4 or IPv6 address*; Default: **::**)	Force the router to always use the same IP source address for all of the SNMP messages
vrf (*VRF name*; default value: **main**)	Set VRF on which service is listening for incoming connections

⚠ the engine-id field holds the suffix value of engine-id, usually, SNMP clients should be able to detect the value, as SNMP values, as read from the router. However, there is a possibility that this is not the case. In which case, the engine-ID value has to be set according to this rule: <engine-id prefix> + <hex-dump suffix>, so as an example, if you have set 1234 as suffix value you have to provide 80003a8c04 + 31323334, combined hex (the result) is 80003a8c0431323334

Community Properties

Sub-menu: `/snmp community`

This sub-menu allows to set up access rights for the SNMP data.

There is little security in v1 and v2c, just Clear text community string („username“) and the ability for Limiting access by IP address.

In the production environment, SNMP v3 should be used as that provides security - Authorization (User + Pass) with MD5/SHA1, Encryption with DES and AES).

```
[admin@MikroTik] /snmp community> print value-list
name: public
address: 0.0.0.0/0
security: none
read-access: yes
write-access: no
authentication-protocol: MD5
encryption-protocol: DES
authentication-password: *****
encryption-password: *****
```

Default settings only have one community named *public* without any additional security settings. These settings should be considered insecure and should be adjusted according to the required security profile.

Properties

Property	Description
address (*IP/IPv6 address*; Default: **0.0.0.0/0**)	Addresses from which connections to SNMP server is allowed
authentication-password (*string*; Default: **""**)	Password used to authenticate the connection to the server (SNMPv3)
authentication-protocol (*MD5* | *SHA1*; Default: **MD5**)	The protocol used for authentication (SNMPv3)
encryption-password (*string*; Default: **""**)	the password used for encryption (SNMPv3)
encryption-protocol (*DES* | *AES*; Default: **DES**)	encryption protocol to be used to encrypt the communication (SNMPv3). AES (see rfc3826) available since v6.16.
name (*string*; Default:)	
read-access (*yes* | *no*; Default: **yes**)	Whether read access is enabled for this community
security (*authorized* | *none* | *private*; Default: **none**)	
write-access (*yes* | *no*; Default: **no**)	Whether write access is enabled for this community

Management information base (MIB)

The Management Information Base (MIB) is the database of information maintained by the agent that the manager can query. You can download the latest MikroTik RouterOS MIB file from here: www.mikrotik.com/downloads

Used MIBs in RouterOS:

- MIKROTIK-MIB
- MIB-2
- HOST-RESOURCES-MIB
- IF-MIB
- IP-MIB
- IP-FORWARD-MIB

- IPV6-MIB
- BRIDGE-MIB
- DHCP-SERVER-MIB
- CISCO-AAA-SESSION-MIB
- ENTITY-MIB
- UPS-MIB
- SQUID-MIB

Object identifiers (OID)

Each OID identifies a variable that can be read via SNMP. Although the MIB file contains all the needed OID values, you can also print individual OID information in the console with the **print oid** command at any menu level:

```
[admin@MikroTik] /interface> print oid

Flags: D - dynamic, X - disabled, R - running, S - slave
0 R name=.1.3.6.1.2.1.2.2.1.2.1 mtu=.1.3.6.1.2.1.2.2.1.4.1
mac-address=.1.3.6.1.2.1.2.2.1.6.1 admin-status=.1.3.6.1.2.1.2.2.1.7.1
oper-status=.1.3.6.1.2.1.2.2.1.8.1 bytes-in=.1.3.6.1.2.1.2.2.1.10.1
packets-in=.1.3.6.1.2.1.2.2.1.11.1 discards-in=.1.3.6.1.2.1.2.2.1.13.1
errors-in=.1.3.6.1.2.1.2.2.1.14.1 bytes-out=.1.3.6.1.2.1.2.2.1.16.1
packets-out=.1.3.6.1.2.1.2.2.1.17.1 discards-out=.1.3.6.1.2.1.2.2.1.19.1
errors-out=.1.3.6.1.2.1.2.2.1.20.1
```

Traps

SNMP traps enable the router to notify the data collector of interface changes and SNMP service status changes by sending traps. It is possible to send out traps with security features to support SNMPv1 (no security). SNMPv2 and variants and SNMPv3 with encryption and authorization.

For SNMPv2 and v3 you have to set up an appropriately configured community as a *trap-community* to enable required features (password or encryption /authorization).

SNMP write

SNMP write allows changing router configuration with SNMP requests. Consider securing access to the router or to router's SNMP, when SNMP and write-access are enabled.

To change settings by SNMP requests, use the command below to allow SNMP to write for the selected community.

```
/snmp community set <number> write-access=yes
```

System Identity

It's possible to change router system identity by SNMP set command.

```
$ snmpset -c public -v 1 192.168.0.0 1.3.6.1.2.1.1.5.0 s New_Identity
```

- *snmpset* - SNMP application used for SNMP SET requests to set information on a network entity;
- *public* - router's community name;
- *192.168.0.0* - IP address of the router;
- *1.3.6.1.2.1.1.5.0* - SNMP value for router's identity;

SNMPset command above is equal to the RouterOS command:

```
/system identity set identity=New_Identity
```

Reboot

It's possible to reboot the router with SNMP set command, you need to set the value for reboot SNMP settings, which is not equal to 0.

```
$ snmpset -c public -v 1 192.168.0.0 1.3.6.1.4.1.14988.1.1.7.1.0 s 1
```

- **1.3.6.1.4.1.14988.1.1.7.1.0**, SNMP value for the router reboot;
- **s 1**, snmpset command to set value, value should not be equal to 0;

Reboot SNMPset command is equal to the RouterOS command:

```
/system reboot
```

Run Script

SNMP write allows running scripts on the router from the **system script** menu when you need to set value for the SNMP setting of the script.

```
$ snmpset -c public -v 1 192.168.0.0 1.3.6.1.4.1.14988.1.1.8.1.1.3.X s 1
```

- **X**, script number, numeration starts from 1;
- **s 1**, snmpset command to set value, the value should not be equal to 0;

The same command on RouterOS:

```
/system script> print
Flags: I - invalid
0 name="test" owner="admin" policy=ftp,reboot,read,write,policy,
test,winbox,password,sniff last-started=jan/01/1970
01:31:57 run-count=23 source=:beep

/system script run 0
```

Running scripts with GET

It is possible to run **/system scripts** via SNMP GET request of the script OID (since 6.37). For this to work SNMP community with write permission is required. OIDs for scripts can be retrieved via the SNMPWALK command as the table is dynamic.

Add script:

```
/system script
add name=script1 owner=admin policy=ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="
/sy reboot "
add name=script2 owner=admin policy=ftp,reboot,read,write,policy,test,password,sniff,sensitive,romon source="[:
put output]"
```

Get the script OID table

```
$ snmpwalk -v2c -cpublic 192.168.88.1 1.3.6.1.4.1.14988.1.1.8
iso.3.6.1.4.1.14988.1.1.8.1.1.2.1 = STRING: "script1"
iso.3.6.1.4.1.14988.1.1.8.1.1.2.2 = STRING: "script2"
iso.3.6.1.4.1.14988.1.1.8.1.1.3.1 = INTEGER: 0
iso.3.6.1.4.1.14988.1.1.8.1.1.3.2 = INTEGER: 0
```

To run the script use table 18

```
$ snmpget -v2c -cpublic 192.168.88.1 1.3.6.1.4.1.14988.1.1.18.1.1.2.2
iso.3.6.1.4.1.14988.1.1.18.1.1.2.2 = STRING: "output"
```

Speed Test

Introduction

The Speed Test is an easy test tool for measuring ping, jitter, TCP and UDP throughput from one MikroTik device, to another. The "speed-test" command is based on the Ping Tool and Bandwidth Test. In order to use this command - Bandwidth test server needs to be accessible.

General interface properties

The speed-test is based on five configurable properties:

- address - IP address of host;
- connection-count - If a device has more than 20 cores - core count will be used (default is 20);
- password - Password for the remote device;
- test-duration - Duration for each test (*By default: 5 tests * 10 sec duration + 1sec pause between each test = 55sec*);
- user - Remote device username;

Configuration Example

Bandwidth and speed tests should be conducted through the devices, not on them to ensure real-life simulation and not to overload the CPU on the devices under testing,(DUT) due to the traffic generating process.

To run a simple test from device A (192.168.88.1) to device B (192.168.88.2):

```
[admin@MikroTik] > /tool/speed-test address=192.168.88.1
             status: done
      time-remaining: 0s
    ping-min-avg-max: 541us / 609us / 3.35ms
  jitter-min-avg-max: 0s / 76us / 2.76ms
                loss: 0% (0/100)
        tcp-download: 921Mbps local-cpu-load:30%
          tcp-upload: 920Mbps local-cpu-load:30% remote-cpu-load:25%
        udp-download: 917Mbps local-cpu-load:6% remote-cpu-load:21%
          udp-upload: 916Mbps local-cpu-load:20% remote-cpu-load:6%
```

If any of device CPU utilization during test reaches 100% warning message will appear:

```
[admin@MikroTik]] > /tool/speed-test address=192.168.88.1
                 ;;; results can be limited by cpu, note that traffic generation/termination
                     performance might not be representative of forwarding performance
             status: done
      time-remaining: 0s
    ping-min-avg-max: 541us / 609us / 3.35ms
  jitter-min-avg-max: 0s / 76us / 2.76ms
                loss: 0% (0/100)
        tcp-download: 721Mbps local-cpu-load:78%
          tcp-upload: 820Mbps local-cpu-load:100% remote-cpu-load:84%
        udp-download: 906Mbps local-cpu-load:10% remote-cpu-load:54%
          udp-upload: 895Mbps local-cpu-load:55% remote-cpu-load:12%
```

"test-duration" parameter allows changing the duration of all of the 5 tests:

-) Ping test with 50ms delay
-) TCP receive
-) TCP send

-) UDP receive
-) UDP send

Torch

MikroTik Torch is a real-time traffic monitoring tool that can be used to monitor the traffic flow through an interface.
Watch our video about this feature.

> ⚠ Traffic that appears in torch is before it has been filtered by a Firewall. This means you will be able to see packets that might get dropped by your Firewall rules.

```
[admin@MikroTik] > /tool/torch
```

You can monitor traffic classified by:

- source address (IPv4 and IPv6);
- destination address (IPv4 and IPv6);
- port;
- protocol;
- mac-protocol;
- VLAN ID;
- mac-address;
- DSCP;

MikroTik Torch shows the protocols you have chosen and the TX/RX data rate for each of them on the particular interface.

> ⚠ Unicast traffic between Wireless clients with client-to-client forwarding enabled will not be visible to the Torch tool. Packets that are processed with hardware offloading enabled bridge will also not be visible (unknown unicast, broadcast and some multicast traffic will be visible to torch tool).

Traceroute

Overview

Sub-menu: `/tool traceroute`

Traceroute displays the list of the routers that packet travels through to get to a remote host. The **traceroute** or **tracepath** tool is available on practically all Unix-like operating systems and **tracert** on Microsoft Windows operating systems.

Traceroute operation is based on TTL value and ICMP “Time Exceeded” message. Remember that TTL value in IP header is used to avoid routing loops. Each hop decrements TTL value by 1. If the TTL reaches zero, the packet is discarded and ICMP Time Exceeded message is sent back to the sender when this occurs.

Initially by traceroute, the TTL value is set to 1 when next router finds a packet with TTL = 1 it sets TTL value to zero, and responds with an ICMP "time exceeded" message to the source. This message lets the source know that the packet traverses that particular router as a hop. Next time TTL value is incremented by 1 and so on. Typically, each router in the path towards the destination decrements the TTL field by one unit TTL reaches zero.

Using this command you can see how packets travel through the network and where it may fail or slow down. Using this information you can determine the computer, router, switch or other network device that possibly causing network issues or failures.

Quick Example

```
[admin@MikroTik] > tool traceroute 10.255.255.1
     ADDRESS                                  STATUS
   1       10.0.1.17 2ms 1ms 1ms
   2    10.255.255.1 5ms 1ms 1ms
[admin@MikroTik] >
```

Traffic Flow

Introduction

MikroTik Traffic-Flow is a system that provides statistical information about packets that pass through the router. Besides network monitoring and accounting, system administrators can identify various problems that may occur in the network. With help of Traffic-Flow, it is possible to analyze and optimize the overall network performance. As Traffic-Flow is compatible with Cisco NetFlow, it can be used with various utilities which are designed for Cisco's NetFlow.

Traffic Flow is able to process only that traffic which is processed by the router CPU, thus HW offloaded traffic will not be seen in Traffic Flow flows (for example, HW offloaded bridged traffic).

Traffic-Flow supports the following NetFlow formats:

- **version 1** - the first version of NetFlow data format, do not use it unless you have to
- **version 5** - in addition to version 1, version 5 has a possibility to include BGP AS and flow sequence number information. Currently, RouterOS does not include BGP AS numbers.
- **version 9** - a new format which can be extended with new fields and record types thank's to its template-style design

General

Sub-menu: `/ip traffic-flow`

This section lists the configuration properties of Traffic-Flow.

Property	Description
interfaces (*string* | *all*; Default: **all**)	Names of those interfaces will be used to gather statistics for traffic-flow. To specify more than one interface, separate them with a comma.
cache-entries (*128k* | *16k* | *1k* | *256k* | *2k* | *...* ; Default: **4k**)	Number of flows which can be in router's memory simultaneously.
active-flow-timeout (*time*; Default: **30m**)	Maximum life-time of a flow.
inactive-flow-timeout (*time*; Default: **15s**)	How long to keep the flow active, if it is idle. If a connection does not see any packet within this timeout, then traffic-flow will send a packet out as a new flow. If this timeout is too small it can create a significant amount of flows and overflow the buffer.
packet-sampling (*no* | *yes*; Default: **no**)	Enable or disable packet sampling feature.
sampling-interval (*integer*; Default: **0**)	The number of packets that are consecutively sampled.
sampling-space (*integer*; Default: **0**)	The number of packets that are consecutively omitted.

info

Packet sampling available since RouterOS **v7.1rc5**!

In the following example:

```
/ip/traffic-flow/set packet-sampling=yes sampling-interval=2222 sampling-space=1111
```

2222 packet consecutive packets will be sampled and then 1111 will be omitted. Then the sampling cycle repeats in such a manner.

Targets

Sub-menu: `/ip traffic-flow target`

With Traffic-Flow targets we specify those hosts which will gather the Traffic-Flow information from the router.

Property	Description
address (*IP:port*; Default:)	IP address and port (UDP) of the host which receives Traffic-Flow statistic packets from the router.
v9-template-refresh (*integer*; Default: **20**)	Number of packets after which the template is sent to the receiving host (only for NetFlow version 9)
v9-template-timeout (*time*; Default:)	After how long to send the template, if it has not been sent.
version (*1* | *5* | *9*; Default:)	Which version format of NetFlow to use

Notes

By looking at the packet flow diagram you can see that traffic flow is at the end of the input, forward, and output chain stack. It means that traffic flow will count only traffic that reaches one of those chains.

For example, you set up a mirror port on a switch, connect the mirror port to a router and set traffic flow to count mirrored packets. Unfortunately, such a setup will not work, because mirrored packets are dropped before they reach the input chain.

Other interfaces will appear in the report if traffic is passing through them and the monitoring interface.

Examples

This example shows how to configure Traffic-Flow on a router

Enable Traffic-Flow on the router:

```
[admin@MikroTik] ip traffic-flow> set enabled=yes
[admin@MikroTik] ip traffic-flow> print
               enabled: yes
            interfaces: all
         cache-entries: 1k
    active-flow-timeout: 30m
  inactive-flow-timeout: 15s
[admin@MikroTik] ip traffic-flow>
```

Specify IP address and port of the host, which will receive Traffic-Flow packets:

```
[admin@MikroTik] ip traffic-flow target> add dst-address=192.168.0.2 port=2055 version=9
[admin@MikroTik] ip traffic-flow target> print
Flags: X - disabled
 #   SRC-ADDRESS       DST-ADDRESS        PORT     VERSION
 0   0.0.0.0           192.168.0.2        2055     9
[admin@MikroTik] ip traffic-flow target>
```

Now the router starts to send packets with Traffic-Flow information.

> **Note**
>
> To use ntop-ng with MikroTik you need to use Nprobe, which is paid software.

See more

- NetFlow Fundamentals
- Traffic flow with Ntop on MikroTik

Traffic Generator

Summary

Traffic Generator is a tool that allows evaluating the performance of DUT (Device Under Test) or SUT (System Under Test).

The tool can generate and send RAW packets over specific ports. It also collects latency and jitter values, TX/RX rates, counts lost packets, and detects Out-of-Order (OOO) packets.

Traffic Generator can be used similar to bandwidth test tool as well as generate packets that will be routed back to packet generator for advanced status collection.

> In a device, which generates packets, it will not be possible to capture packets generated by the traffic-generator with a sniffer tool, torch, or firewall on the outgoing interface.
>
> Since this tool could be used maliciously, it is possible to lock this functionality by entering the command '/system device-mode update traffic-gen=no' and holding the reset button afterwards.

General

```
/tool traffic-generator
```

This menu allows to set general traffic generator properties and contains commands to quickly start and stop the tool.

Properties

Property	Description
latency-distribution-max (*time*; Default: **100us**)	Maximal latency range for latency distribution measurements. Based on this value, RouterOS will decide what latency range to use as the latency-distribution-measure-interval property
measure-out-of-order (*yes* | *no*; Default:)	Whether to measure Out-of-Order packets. The default value is based on CPU type (multi-core CPU default **no**; single-core CPU default **yes**). When the property is enabled on the multi-core device, a single stream will utilize only a single CPU core
stats-samples-to-keep (*integer*; Default: **100**)	How many data examples to collect
test-id (*integer [0..255]* ; Default: **0**)	

Read-Only Properties

Property	Description
latency-distribution-samples (*integer*)	Shows how many individual time periods the latency-distribution-measure-interval is divided into
latency-distribution-measure-interval (*time*)	Shows total latency measurement range
running (*yes* | *no*)	Shows whether the traffic generator tool is started.

Commands

Property	Description
quick ()	This command allows to quickly start the packet generator and print the stats output to the terminal. Command also accepts several parameters that override settings in packet template and stream settings. Accepted parameters are **duration, entries-to-show, freeze-frame-interval, id, interface, mbps, measure-out-of-order, packet-count, packet-size, port, pps, stream, test-id, tx-template** • tx-template - packet templates to generate traffic (max 16 templates) • duration - how long to run the test • entries-to-show - how many status lines print to the terminal • freeze-frame-interval - how often to update the status to the terminal The rest of the parameters are not command-specific and are described elsewhere. Parameters specified when running quick command override configured values. In case if a parameter is specified only for one header then the value is multiplied for all the other headers (if required).
start ()	Commands start the traffic generator tool in the background. It accepts one parameter **test-id**
stop ()	Command stops the started traffic generator tool by **start** command.
inject ()	Inject raw data into the interface.
inject-pcap ()	Inject raw data directly from pcap file.

Packet Template

```
/tool traffic-generator packet-template
```

This sub-menu allows building packets based on provided parameters. Based on parameters you can build IP packets with VLAN tags and set UDP ports. A raw packet template is generated based on provided parameters.

If you require more low-level packets or take full advantage of the traffic generator, then please use a raw-packet-template builder to build the packet.

If the same type of header is present in the packet more than once then header field values are passed as a comma-separated list. (For example, if there are two IP headers then source addresses are given like "IP-src=1.1.1.1,2.2.2.2").

For quicker header construction many of the header field values are assumed. For example, if the header stack is "mac, IP" then the traffic generator can assume that the mac-protocol value is "IP". Or if the "port" or "interface" setting is specified traffic generator can assume "mac-src" to be the MAC address of the interface). Assumed values have distinct names that start with "assumed-" and are read-only. Manually specified values override assumed ones.

> Assumed values are not automatically updated. New values are assumed after template edit. "packet-template set 0" is enough to trigger new assumed values

Properties

Property	Description
comment (*string*; Default:)	Short description of the packet you are building.

compute-checksum-from-offset (*no-checksum | integer[0..4294967295]*; Default:)	specifies the byte offset from where in the packet 2-byte checksum will be calculated (Example: set to 14, to skip packets Ethernet header when calculating checksum)
data (*incrementing | random | specific-byte | uninitialized*; Default: **uninitialized**)	Specifies how packet payload will be filled: • **uninitialized** - packets data (after the header) is uninitialized, but not zero. Fastest. • **specific-byte** - works together with setting data-byte • **incrementing** - packets data filled with "00 01 02 03" and so on • **random** - packets data filled with random bytes. Slowest.
data-byte (*hex [0..FF]*; Default: **0**)	A byte will be used to fill the packet payload.
interface (*string*; Default:)	Optional parameter of packet template. This is mutually exclusive with the "port" setting. Specifying "interface" allows users not to create a port entry for interface in the port menu. In fact, a port entry is created dynamically. This is useful for running quick tests.
ip-dscp (*list of integer[0..255] (max 16 times)*; Default:)	Single or list of DS Fields that will be set in IP header (DS Field contains DSCP value and the ECN value)
ip-dst (*list of IP/Netmask (max 16 times)*; Default:)	List of destination IP addresses that will be used when generating IP headers.
ip-frag-off (*list of integer[0..65535] (max 16 times)*; Default:)	List of fragmentation offsets in IP header.
ip-gateway (*IP*; Default:)	In situations when sender and receiver are the same devices, it is impossible to determine next-hop automatically from **ip-dst**. If ip-gateway is specified packet template will assume the destination mac address based on resolved ip-gateway.
ip-id (*list of integer [0..65535]*; Default:)	
ip-protocol (*list of IP protocols (max 16 times)*; Default:)	
ip-src (*list of IP/Mask (max 16 times)*; Default:)	
ip-ttl (*list of integer [0..255] (max 16 times)*; Default:)	
mac-dst (*list of MAC/MASK (max 16 times)*; Default:)	
mac-protocol (*list of mac protocols (max 16 times)*; Default:)	
mac-src (*list of MAC/MASK (max 16 times)*; Default:)	
name (*string*; Default:)	Descriptive name of the template.
port (*string*; Default:)	Optional parameter of packet template. This suggests a port through which packets generated using this template should be sent out. Port can also be specified in other places such as in stream settings. This is mutually exclusive with interface setting.
raw-header (*string (max 16 times)*; Default:)	Raw packet header as string in hexadecimal format.
udp-dst-port (*list of port [0..65535]/mask [0..FFFF] (max 16 times)*; Default:)	

udp-src-port (*list of port [0..65535]/mask [0..FFFF] (max 16 times)*; Default:)	
vlan-id (; Default:)	
vlan-priority (; Default:)	
vlan-protocol (; Default:)	
header-stack (*list of ip | mac | raw | udp | vlan (max 16 times)*; Default: **ip**)	A sequence of headers that a generated packet should have. Currently supports: • **mac** - Ethernet header (14 bytes) • **vlan** - Ethernet VLAN tag (4 bytes) • **ip** - IPv4 header (20 bytes) • **udp** - UDP header (8 bytes) • **raw** - arbitrary bytes specified as a hex string Most header types can be present in the header multiple times. There can be only 2 IP headers and 1 UDP header per packet. Some limitations are imposed on possible sequences of headers based on our practical experience with network protocols (for example VLAN header can follow only a mac header or other VLAN header). The traffic generator suggests the first header for a packet template (in the port menu). But it is not enforced.

Port Configuration

```
/tool traffic-generator port
```

This menu allows configuring ports that will be associated with a specific interface and will be used to receive/send generated packets.

Properties

Property	Description
disabled (*yes | no*; Default: **no**)	Whether the port is disabled and does not participate in receiving/sending the packets
name (*string*; Default:)	Descriptive name of the port
interface (*string*; Default:)	Name of the interface associated with the port.

Read-Only Properties

Property	Description
dynamic (*yes | no*)	Whether the port configuration is generated automatically.
first-header (*ip | mac | raw | udp | vlan*)	Shows suggested the first header for packets to be sent out of the specified interface. This information can be used when creating packet templates.
inactive (*yes | no*)	Whether the port is inactive and can't participate in tx/rx of the packets.

Stats

```
/tool traffic-generator stats
```

If the traffic generator is not running in **quick** mode then all statistics about the test is stored in this menu.

Latency Distribution

```
/tool traffic-generator stats latency-distribution
```

This sub-menu shows how many packets are received in a specific latency range. Latency range can be viewed by streams or by sequences (for example, **print stream-num=3**, **print seq=10**)

Here is an example output of the latency graph:

```
[admin@test-host] /tool traffic-generator stats latency-distribution> print
 # LATENCY                 COUNT        SHARE GRAPH
 0 0-15.5us                    0           0%
 1 15.5us-31us                 0           0%
 2 31us-46.5us                 0           0%
 3 46.5us-62.1us               0           0%
 4 62.1us-77.6us               0           0%
 5 77.6us-93.1us               0           0%
 6 93.1us-109us                0           0%
 7 109us-124us                 0           0%
 8 124us-140us                 0           0%
 9 140us-155us                 0           0%
10 155us-171us                 0           0%
11 171us-186us                 4           0% *
12 186us-202us                29           0% *
13 202us-217us                90       0.001% *
14 217us-233us               302       0.004% *
15 233us-248us               630       0.009% *
16 248us-264us               789       0.011% *
17 264us-279us             1 384       0.021% -*
18 279us-295us             1 990        0.03% --*
19 295us-310us             2 966       0.045% ---*
20 310us-326us             4 089       0.062% -----*
21 326us-341us             4 958       0.075% ------*
22 341us-357us             6 059       0.091% -------*
23 357us-372us             6 660       0.101% --------*
24 372us-388us             8 320       0.126% ----------*
25 388us-403us             9 988       0.151% ------------*
26 403us-419us            11 781       0.178% --------------*
27 419us-434us            12 512       0.189% ---------------*
28 434us-450us            13 836        0.21% ----------------*
29 450us-465us            15 681       0.238% --------------------*
30 465us-481us            17 740       0.269% ----------------------*
31 481us-496us            19 913       0.302% -------------------------*
32 496us-512us            21 106        0.32% --------------------------*
33 512us-528us            22 848       0.346% ----------------------------*
34 528us-543us            25 059        0.38% -------------------------------*
35 543us-559us            26 593       0.403% --------------------------------*
36 559us-574us            27 663       0.419% --------------------------------*
37 574us-590us            29 351       0.445% ----------------------------------*
38 590us-605us            31 265       0.474% ------------------------------------*
39 605us-621us            33 224       0.504% --------------------------------------*
40 621us-636us            34 464       0.523% --------------------------------------*
41 636us-652us            35 630        0.54% ----------------------------------------*
42 652us-667us            37 245       0.565% ------------------------------------------*
43 667us-683us            38 158       0.579% ------------------------------------------*
44 683us-698us            38 626       0.586% -------------------------------------------*
45 698us-714us            38 985       0.591% -------------------------------------------*
46 714us-729us            39 061       0.592% -------------------------------------------*
47 729us-745us            39 750       0.603% --------------------------------------------*
48 745us-760us            39 145       0.594% -------------------------------------------*
49 760us-776us            39 162       0.594% -------------------------------------------*
50 776us-791us            38 197       0.579% ------------------------------------------*
51 791us-807us            37 811       0.573% ------------------------------------------*
52 807us-822us            37 364       0.567% -----------------------------------------*
53 822us-838us            36 770       0.558% -----------------------------------------*
54 838us-853us            35 831       0.543% ----------------------------------------*
55 853us-869us            35 380       0.536% ----------------------------------------*
56 869us-884us            34 472       0.523% ---------------------------------------*
57 884us-900us            33 672       0.511% ---------------------------------------*
58 900us-915us            33 799       0.513% ---------------------------------------*
59 915us-931us            32 754       0.497% --------------------------------------*
60 931us-946us            32 339        0.49% --------------------------------------*
61 946us-962us            32 419       0.492% --------------------------------------*
62 962us-977us            32 107       0.487% -------------------------------------*
63 977us-993us            31 552       0.478% -------------------------------------*
64 0-993us             1 221 523       18.54%
```

Properties

Property	Description
count (*integer*)	Number of packets in the current latency range
graph (*string*)	graphical representation of share
latency (*string*)	latency range
share (*percent*)	Percentage of packets falling in this latency range.

Stream Stats

```
/tool traffic-generator stats stream
```

This sub-menu stores statistics sorted by streams. Output is the same as in **quick** mode.

```
[admin@test-host] /tool traffic-generator stats stream> print
 # SEQ    NUM     TX-PACKET   TX-RATE     RX-PACKET   RX-RATE   LOST-PACKET LOST-RATE
 0 1      3          43 064 499.5Mbps        25 180 292.0Mbps        17 884 207.4Mbps
 1 1      4          43 062 499.5Mbps        39 946 463.3Mbps         3 116  36.1Mbps
 2 1      TOT        86 126 999.0Mbps        65 126 755.4Mbps        21 000 243.6Mbps
 3 2      3          43 544 505.1Mbps        30 449 353.2Mbps        13 095 151.9Mbps
 4 2      4          43 543 505.0Mbps        42 982 498.5Mbps           561   6.5Mbps
 5 2      TOT        87 087 1010.2...        73 431 851.7Mbps        13 656 158.4Mbps

...

59 20     TOT        87 277 1012.4...        73 755 855.5Mbps        13 522 156.8Mbps
60 21     3          43 546 505.1Mbps        30 605 355.0Mbps        12 941 150.1Mbps
61 21     4          43 546 505.1Mbps        42 682 495.1Mbps           864  10.0Mbps
62 21     TOT        87 092 1010.2...        73 287 850.1Mbps        13 805 160.1Mbps
63 TOT    3         913 942 504.8Mbps       629 210 347.5Mbps       284 732 157.2Mbps
64 TOT    4         913 939 504.8Mbps       898 374 496.2Mbps        15 565   8.5Mbps
65 TOT    TOT     1 827 881 1009.6...     1 527 584 843.8Mbps       300 297 165.8Mbps
```

Port Stats

```
/tool traffic-generator stats port
```

This sub-menu stores statistics sorted by rx/tx ports.

```
[admin@test-host] /tool traffic-generator stats port> print
 # SEQ    PORT        RX-UNK-PACKET    RX-UNK-BYTE RX-UNK...     TX-PACKET   TX-RATE     RX-PACKET
 0 1      port0:et...             0              0      0bps        43 064 499.5Mbps        39 946
 1 1      port1:et...             0              0      0bps        43 062 499.5Mbps        25 180
 2 1      TOT                     0              0      0bps        86 126 999.0Mbps        65 126
 3 2      port0:et...             0              0      0bps        43 544 505.1Mbps        42 982
 4 2      port1:et...             0              0      0bps        43 543 505.0Mbps        30 449
 5 2      TOT                     0              0      0bps        87 087 1010.2...        73 431
 6 3      port0:et...             0              0      0bps        43 540 505.0Mbps        42 615
 7 3      port1:et...             0              0      0bps        43 540 505.0Mbps        30 191
 8 3      TOT                     0              0      0bps        87 080 1010.1...        72 806
```

Raw Stats

```
/tool traffic-generator stats raw
```

This sub-menu stores raw statistics data.

```
[admin@test-host] /tool traffic-generator stats raw> print
 # SEQ    PORT       NUM     TX-PACKET   TX-RATE     RX-PACKET   RX-RATE   LOST-PACKET LOST-RATE
 0 1      port0:e... 3          43 064 499.5Mbps             0      0bps        43 064 499.5Mbps
 1 1      port1:e... 3               0      0bps        25 180 292.0Mbps       -25 180 292.0Mbps
 2 1      TOT        3          43 064 499.5Mbps        25 180 292.0Mbps        17 884 207.4Mbps
 3 1      port0:e... 4               0      0bps        39 946 463.3Mbps       -39 946 463.3Mbps
 4 1      port1:e... 4          43 062 499.5Mbps             0      0bps        43 062 499.5Mbps
 5 1      TOT        4          43 062 499.5Mbps        39 946 463.3Mbps         3 116  36.1Mbps
 6 1      port0:e... TOT        43 064 499.5Mbps        39 946 463.3Mbps         3 118  36.1Mbps
 7 1      port1:e... TOT        43 062 499.5Mbps        25 180 292.0Mbps        17 882 207.4Mbps
 8 2      port0:e... 3          43 544 505.1Mbps             0      0bps        43 544 505.1Mbps
 9 2      port1:e... 3               0      0bps        30 449 353.2Mbps       -30 449 353.2Mbps
10 2      TOT        3          43 544 505.1Mbps        30 449 353.2Mbps        13 095 151.9Mbps
```

Streams

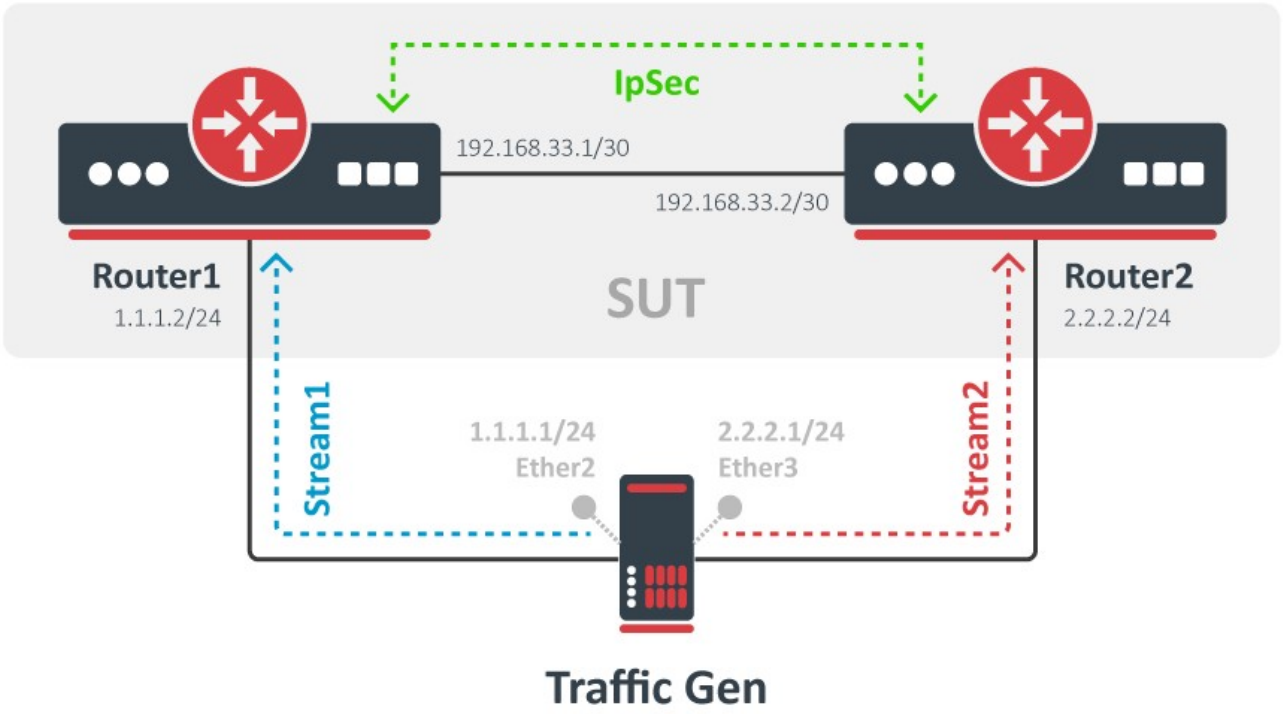
Properties

Property	Description
disabled (*yes* | *no*; Default: **no**)	Whether the stream is disabled
mbps (*integer [0..4294967295]*; Default: **0**)	Value in Megabits per second that stream will try to generate.
name (*string*; Default:)	Descriptive name of the stream.
num (*integer [0..15]*; Default: **0**)	
packet-size (*integer[1..65535] [-integer[1..65535]]*; Default: **0**)	Generated the size of the packets in bytes. Can be set as the range for random packet size generation.
port (*string*; Default:)	Name of the port from the Port menu that will be used to transmit packets.
pps (*integer [0..4294967295]*; Default: **0**)	Packets per second that stream will try to generate.
tx-template (*string*; Default:)	Name of the packet template from packet-template or raw-packet-template menus used as the packet content source.

Configuration Examples

IPsec tunnel performance test

Consider following test setup



System Under Test (SUT) consists of two routers connected to a traffic generator server. The connection between both SUT routers is IPSec encrypted.

The traffic generator will run two streams:

- in a direction from 1.1.1.0/24 network to 2.2.2.0/24 network
- in a direction from 2.2.2.0/24 network to 1.1.1.0/24 network.

R1 routing and IPsec setup

```
/ip address
add address=192.168.33.1/30 interface=ether1
add address=1.1.1.2/24 interface=ether2

/ip route
add dst-address=2.2.2.0/24 gateway=192.168.33.2

/ip ipsec proposal
set default enc-algorithms=aes-128

/ip ipsec peer
add address=192.168.33.2 secret=123

/ip ipsec policy
add sa-src-address=192.168.33.1 sa-dst-address=192.168.33.2 \
    src-address=1.1.1.0/24 dst-address=2.2.2.0/24 tunnel=yes
```

R2 routing and IPsec setup

```
/ip address
add address=192.168.33.2/30 interface=ether1
add address=2.2.2.2/24 interface=ether2

/ip route
add dst-address=1.1.1.0/24 gateway=192.168.33.1

/ip ipsec proposal
set default enc-algorithms=aes-128

/ip ipsec peer
add address=192.168.33.1 secret=123

/ip ipsec policy
add sa-src-address=192.168.33.2 sa-dst-address=192.168.33.1 \
    src-address=2.2.2.0/24 dst-address=1.1.1.0/24 tunnel=yes
```

Traffic generator server setup

```
/ip address
add address=1.1.1.1/24 interface=ether2
add address=2.2.2.1/24 interface=ether3
```

First, we will define which ports will be used as traffic generator tx/rx ports

```
/tool traffic-generator port
add disabled=no interface=ether2 name=port0
add disabled=no interface=ether3 name=port1
```

To construct the actual packet that will be generated, packet-template is used.

```
/tool traffic-generator packet-template
add header-stack=mac,ip,udp ip-dst=2.2.2.1/32 ip-gateway=1.1.1.2 ip-src=1.1.1.1/32 \
    name=routing-1 port=port0
add header-stack=mac,ip,udp ip-dst=1.1.1.1/25 ip-gateway=2.2.2.2 ip-src=2.2.2.1/32 \
    name=routing-2 port=port1
```

Notice that mac addresses were not specified since the template generator can assume the next-hop mac address automatically by sending ARP messages. Since we are doing routing and destination IP is not directly reachable, we have set **the ip-gateway** parameter to determine the next-hop mac-address.

When running **print** you can see all assumed (detected) values including mac addresses.

```
[admin@test-host] /tool traffic-generator packet-template> print
 0 name="routing-1" header-stack=mac,ip,udp port=port0
   assumed-mac-dst=00:0C:42:00:38:9D assumed-mac-src=00:0C:42:40:94:25
   assumed-mac-protocol=ip assumed-ip-dscp=0 assumed-ip-id=0
   assumed-ip-frag-off=0 assumed-ip-ttl=64 assumed-ip-protocol=udp
   ip-src=1.1.1.1/32 ip-dst=2.2.2.1/32 assumed-udp-src-port=100
   assumed-udp-dst-port=200 ip-gateway=1.1.1.2 data=uninitialized data-byte=0

 1 name="routing-2" header-stack=mac,ip,udp port=port1
   assumed-mac-dst=00:0C:42:00:38:D1 assumed-mac-src=00:0C:42:40:94:26
   assumed-mac-protocol=ip assumed-ip-dscp=0 assumed-ip-id=0
   assumed-ip-frag-off=0 assumed-ip-ttl=64 assumed-ip-protocol=udp
   ip-src=2.2.2.1/32 ip-dst=1.1.1.1/32 assumed-udp-src-port=100
   assumed-udp-dst-port=200 ip-gateway=2.2.2.2 data=uninitialized data-byte=0
```

For example, if any router in SUT were to change, assumed mac-addresses would not be updated automatically. To update packet templates simply issue command:

```
/tool traffic-generator packet-template set [find]
```

The last part is to configure streams

```
/tool traffic-generator stream
add disabled=no mbps=500 name=str1 id=3 packet-size=1450 port=port0 pps=0 \
    tx-template=routing-1
add disabled=no mbps=500 name=str3 id=4 packet-size=1450 port=port1 pps=0 \
    tx-template=routing-2
```

Notice that each stream has a unique **id** value. This value identifies stream packets, otherwise, the traffic generator will not work.

Now we are ready to run the test. In this case, **quick** mode will be used:

```
[admin@test-host] /tool traffic-generator> quick mbps=450
SEQ    NUM     TX-PACKET   TX-RATE     RX-PACKET   RX-RATE        RX-OOO   LOST-PACKET LOST-RATE
37     4          39 488 458.0Mbps        39 270 455.5Mbps        15 509          218   2.5Mbps
37     TOT        78 976 916.1Mbps        76 485 887.2Mbps        22 529        2 491  28.8Mbps
38     3          38 957 451.9Mbps        37 657 436.8Mbps         7 078        1 300  15.0Mbps
38     4          38 958 451.9Mbps        38 402 445.4Mbps        14 763          556   6.4Mbps
38     TOT        77 915 903.8Mbps        76 059 882.2Mbps        21 841        1 856  21.5Mbps
39     3          38 816 450.2Mbps        37 893 439.5Mbps         7 307          923  10.7Mbps
39     4          38 815 450.2Mbps        38 642 448.2Mbps        15 110          173   2.0Mbps
39     TOT        77 631 900.5Mbps        76 535 887.8Mbps        22 417        1 096  12.7Mbps
40     3          39 779 461.4Mbps        37 415 434.0Mbps         7 136        2 364  27.4Mbps
40     4          39 780 461.4Mbps        39 567 458.9Mbps        15 908          213   2.4Mbps
40     TOT        79 559 922.8Mbps        76 982 892.9Mbps        23 044        2 577  29.8Mbps
41     3          39 218 454.9Mbps        37 089 430.2Mbps         7 075        2 129  24.6Mbps
41     4          39 218 454.9Mbps        38 663 448.4Mbps        15 752          555   6.4Mbps
41     TOT        78 436 909.8Mbps        75 752 878.7Mbps        22 827        2 684  31.1Mbps
42     3          39 188 454.5Mbps        37 906 439.7Mbps         6 729        1 282  14.8Mbps
42     4          39 187 454.5Mbps        38 954 451.8Mbps        15 565          233   2.7Mbps
42     TOT        78 375 909.1Mbps        76 860 891.5Mbps        22 294        1 515  17.5Mbps
TOT    3       1 645 468 454.4Mbps     1 568 201 433.1Mbps       280 174       77 267  21.3Mbps
TOT    4       1 645 464 454.4Mbps     1 626 896 449.3Mbps       627 480       18 568   5.1Mbps
TOT    TOT     3 290 932 908.9Mbps     3 195 097 882.4Mbps       907 654       95 835  26.4Mbps
```

Stats show throughput of each stream and total throughput of both streams, Out-of-order packet count, Lost rate, latency, and jitter.

Watchdog

Summary

Sub-menu: `/system watchdog`

This menu allows the configuring system to reboot, when a specific IP address does not respond, or when it detects, that the software has locked up. The detection is done in two ways:

- Software watchdog timer (mostly caused by hardware malfunction) device can recover itself with a reboot;
- Ping watchdog can monitor connectivity to a specific IP address and trigger the reboot function.

> **Note:** These are two different Watchdog features and both have their own settings. By default software Watchdog is enabled and ping Watchdog is disabled. You can enable ping Watchdog by specifying an IP address and you can disable the software Watchdog by unsetting the Watchdog Timer option.

> **Note:** Watchdog reboot is not a system failure. Such reboot also will not generate autosupout file. Watchdog reboot is "/system reboot" automatically triggered by operating system when some service is not responding as fast as it should. Reasons for that can vary between damaged hardware, slow software implementation for some service, DDoS attack, bad configuration and others.

Properties

Property	Description
auto-send-supout (*yes* / *no*; Default: **no**)	After the support output file is automatically generated, it can be sent by email.
automatic-supout (*yes* / *no*; Default: **yes**)	When software failure happens, a file named "autosupout.rif" is generated automatically. The previous "autosupout.rif" file is renamed to "autosupout.old.rif".
no-ping-delay (*time*; Default: 5m)	Specifies how long will it wait before trying to reach the watch-address.
ping-timeout (*time*; Default: 60s)	Specifies the time interval in which the device will be pinged 6 times (after "no-ping-delay").
send-email-from (*string* ; Default:)	The e-mail address to send the support output file from. If not set, the value set in /tool e-mail is used.
send-email-to (*string*; Default:)	The e-mail address to send the support output file to.
send-smtp-server (*string* ; Default:)	SMTP server address to send the support output file through. If not set, the value set in /tool e-mail is used.
watch-address (*IP*; Default:)	The system will reboot, in case 6 sequential pings to the given IP address will fail. If set to none this feature is disabled. By default, the router will reboot every 6 minutes if the watch-address is set and not reachable.
watchdog-timer (*yes* / *no*; Default: **yes**)	Whether to reboot if a system is unresponsive for a minute.

Quick Example

To make system generate a support output file and sent it automatically to support@example.com through the 192.0.2.1in case of a software crash:

```
[admin@MikroTik] system/watchdog/ set auto-send-supout=yes \
\... send-to-email=support@example.com send-smtp-server=192.0.2.1
[admin@MikroTik] system watchdog> print
      watch-address: none
     watchdog-timer: yes
      no-ping-delay: 5m
   automatic-supout: yes
   auto-send-supout: yes
   send-smtp-server: 192.0.2.1
      send-email-to: support@example.com
```

Extended features

In This Section:

Back To Home

Overview

Sub-menu: `/ip cloud`
Packages required: `routeros`
RouterOS version required: v7.12 beta
Hardware requirements: ARM/ARM64/TILE architecture devices

Back To Home is a convenience feature, that configures your device for secure VPN access from anywhere in the world to your router and your network, even if your router does not have a public IP address, is behind NAT or Firewall.

Configuration is done with MikroTik VPN companion app (Android, iPhone).

> **Note**
>
> Back to Home is a feature still in active development. Many features are yet to come!

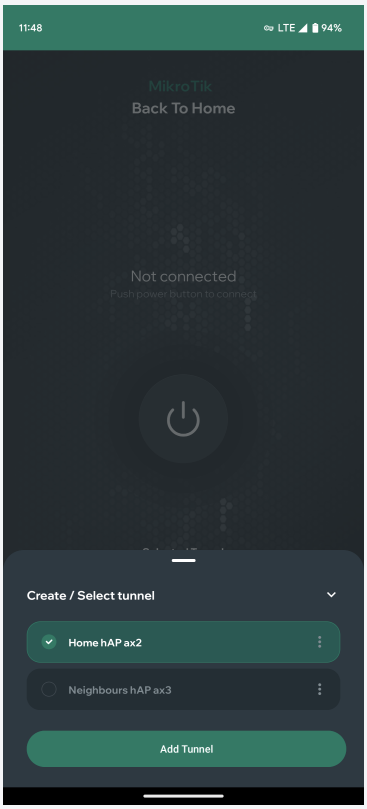
If the VPN server (your home router) has a public IP address, the VPN app will create a direct VPN connection between the phone and the router. However, if the router is not directly reachable from the internet, the connection will be made through the MikroTik relay servers. The connection is always end-to-end encrypted, the relay server or any other device does not have access to the encryption keys, in essence, the relay only helps your device to reach the router. The connection will appear as going out from your router, not from the relay. *In case of going through relay, speed could be limited.*

This feature is a convenient option to access your home network or view content available in your home country, from locations, where some content is not available. It is not meant for anonymity, it is for simple one click access to your home network. For more granular security controls, we recommend to manually configure and secure a VPN connection using the advanced RouterOS options.

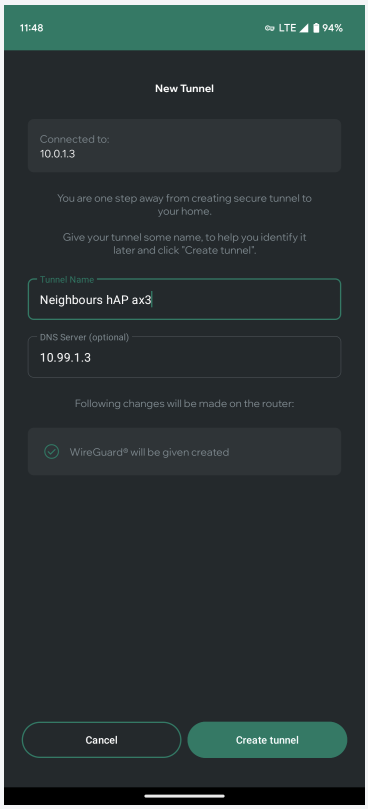
> **Optional**
>
> *You can also use a standard Wireguard application to connect to your BTH enabled devices. The configuration that is required for your laptop or phone Wireguard application is available in the IP CLOUD menu, once BTH is configured.*

Enabling BTH using the app

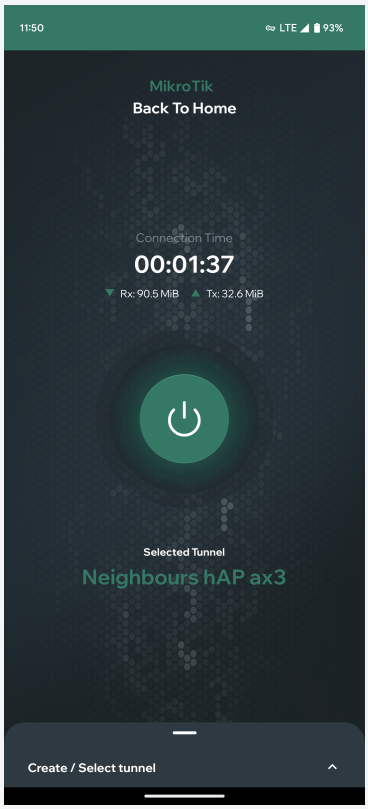
- Connect to router's Wi-Fi using your phone;
- Open MikroTik VPN application (Android, iPhone);
- Open bottom sheet;
- Tap "Add Tunnel";
- Enter your local router IP address (most likely 192.168.88.1), username, and password, tap "Connect";
- Give tunnel a name, optionally enter DNS server address (this can also be 192.168.88.1 or a public DNS like 1.1.1.1), tap "Create tunnel";
- You can disconnect from router's Wi-Fi and connect to other network, like LTE/5G or simply leave your house now;
- Tap power button to toggle connection of selected tunnel.



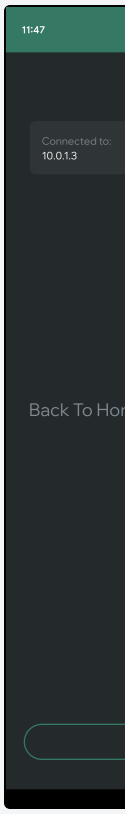
Tap "Add tunnel"



Provide your router info



Connection established



If device is not s

Configuring BTH manually in RouterOS (optional, if no smartphone is available to you)

> **Important notice**
>
> It is important to note, **NOTHING has to be configured in RouterOS to use Back to Home**. Simply use the BTH app (see above section) to enable it. The whole point of Back to Home is to avoid using Winbox or command line. Below instructions are only for debugging or seasoned administrators.

1. Connect to router
2. Enable DDNS Cloud service: `/ip/cloud/set ddns-enabled=yes`
3. Enable Back To Home: `/ip/cloud/set back-to-home-vpn=enabled`
4. Print tunnel configuration: `/ip/cloud/print`
5. Scan QR Code (`vpn-wireguard-client-config-qrcode`) or Copy config (`vpn-wireguard-client-config`) and enter in preferred WireGuard® client. Only one client at a time will be available to use this config.

User manager for Back to Home

Since RouterOS 7.14 there is a new back to home specific user manager available in the menu `/ip/cloud/back-to-home-users>` where you can see all the users that are added by the Back to Home mobile app, change their firewall preference and also add new ones.

```
[boss@mikrotik-ax] /ip/cloud/back-to-home-users> print detail
Flags: X - disabled; A - active

 0  A name="user1" slot=3 expires=never client-address=192.168.216.3/32,fc00:0:0:216::3/128 allow-lan=no
      private-key="OHqR2BZXJp0N6//3JzzoJhBJVb0rrSxV0dxQL/2UdXY=" public-key="
Na7oEq9XLdeK8ouCUX+tC4FIM51vEnZ7mLiFqG9xiUQ="

[boss@mikrotik-ax] /ip/cloud/back-to-home-users>
```

When adding users in this menu, you can view their Wireguard config and QR code with this command `/interface/wireguard/peers/show-client-config user1`

`Allow-lan=no` will add the users into a firewall address list, that only allows internet access, but blocks the user from accessing your internal network. Note that expiry date can't be changed, once a user has been added.

Removing and disabling connections

In the smartphone app, the VPN configuration is added to the System VPN settings. In this regard, the Back to Home app only acts as a wizard. It supplies needed config to the operating system (this is why iPhone will warn you about altering system configuration).

To remove the created connection, go into the smartphone settings app and remove the VPN connections from there.

In the MikroTik router side, you should manually delete the added Peers in the Wireguard menu. Note that "revoke-and-disable" button can't be used to "Pause" the use of the Back to Home feature. Once you revoke-and-disable in RouterOS, all Peers will be disassociated from the Cloud / Relay servers, and you will have to re-create the connection from the Smartphone app. Therefore, once you have used the option "revoke-and-disable" in RouterOS IP Cloud menu, you need to also delete the Peers from the Wireguard menu, as they can't be re-used.

Property reference

IP Cloud

Sub-menu: `/ip/cloud/`

Back to Home shares the menu with IP Cloud. Back to Home parameters:

Property	Description
back-to-home-vpn (*enabled* | *revoked-and-disabled*; Default: **revoked-and-disabled**)	Enables or revokes and disables the Back to Home service. ddns-enabled has to be set to yes, for BTH to function.
vpn-dns-name (*read-only: string*)	Shows the DNS name assigned to the device. Name consists of 12 characters serial number appended by . *sn.mynetname.net*. This field is visible only after at least one ddns-request is successfully completed.
vpn-port (*read-only: integer*)	Port used by BTH VPN.
vpn-status (*read-only: string*)	Contains text string that describes the current BTH state.
vpn-relay-rtts (*read-only;* "region (ip4: time (**ms**), ip6: time (**ms)**")	Round trip time in milliseconds for each available relay, values are shown both for IPv4 and IPv6.
vpn-relay-ipv4-status (*read-only: string*)	Status on connection to relay and detailed information about relay
vpn-relay-ipv6-status (*read-only: string*)	Status on connection to relay and detailed information about relay
vpn-relay-codes (*read-only: string*)	Available VPN relay codes, which can be referenced in vpn-prefer-relay-code. All available relays will be shown here.
vpn-relay-addressess (*read-only: string*)	IPv4 address of the relay
vpn-relay-addressess-ipv6 (*read-only: string*)	IPv6 address of the relay
vpn-private-key (*read-only: string*)	Private key for BTH
vpn-public-key (*read-only: string*)	Public key for BTH
vpn-peer-private-key (*read-only: string*)	Peer private key
vpn-peer-public-key (*read-only: string*)	Peer public key
vpn-prefer-relay-code (*string;*)	You can enter relay code that will be preferred for BTH connection, if not set, relay with smallest RTT will be chosen.

vpn-interface (*read-only: string*)	Name of the created interface for Back to Home WireGuard® tunnel.
vpn-wireguard-client-config (*read-only: string*)	Configuration that can be entered in your preferred WireGuard® client. Only one client at a time will be available to use this config.
vpn-wireguard-client-config-qrcode (*read-only*)	Scannable QR Code for your preferred WireGuard® client. Only one client at a time will be available to use this config.

> When using vpn-wireguard-client-config or vpn-wireguard-client-config-qrcode, both options are equal, you only need to import one of these into your WireGuard client device.

Back to Home users

Sub-menu: `/ip/cloud/back-to-home-users/`

In this menu you can see all the users that are added by the Back to Home mobile app, change their firewall preference and also add new ones.

Property	Description
name (*string*)	Informative name of BTH user
expires (string; never | date: *"YYYY-MM-DD HH:MM:SS";* Default: **never***)*	Expiration time and date for user, cannot be changed once user is created
client-addresss (string: IPv4 | IPv6)	Client address, if not specified one will be made automatically
allow-lan (string: yes | no; Default: **no**)	Will add the user into a firewall address list, that only allows internet access, but blocks the user from accessing your internal network
private-key (string;)	Private key for user, if not set manually, it will be generated by the system
public-key (string;)	Public key for user, if not set manually, it will be generated by the system

Container

Summary

Sub-menu: `/container`
Packages required: `container`

A container is MikroTik's implementation of Linux containers, allowing users to run containerized environments within RouterOS. The container feature was added in RouterOS v7.4beta4.

Disclaimer

- you need physical access to the router to enable support for the container feature, it is disabled by default;
- once the container feature is enabled, containers can be added/configured/started/stopped/removed remotely!
- if the router is compromised, containers can be used to easily install malicious software in your router and over network;
- your router is as secure as anything you run in container;
- if you run container, there is no security guarantee of any kind;
- running a 3rd party container image on your router could open a security hole/attack vector/attack surface;
- an expert with knowledge how to build exploits will be able to jailbreak/elevate to root;

Security risks:

when a security expert publishes his exploit research - anyone can apply such an exploit;
someone will build a container image that will do the exploit AND provide a Linux root shell;
by using a root shell someone may leave a permanent backdoor/vulnerability in your RouterOS system even after the docker image is removed and the container feature disabled;
if a vulnerability is injected into the primary or secondary routerboot (or vendor pre-loader), then even netinstall may not be able to fix it;

Requirements

Container package is compatible with **arm arm64** and **x86** architectures. Using of remote-image (similar to docker pull) functionality requires a lot of free space in main memory, 16MB SPI flash boards may use pre-build images on USB or other disk media.

External disk is highly recommended

Container package needs to be installed

```
/container
```

Properties

Property	Description
cmd (string; Default:)	The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.
comment (*string*; Default:)	Short description
dns (string; Default:)	
domain-name (string; Default:)	
entrypoint (string; Default: **)**	An ENTRYPOINT allows to specify executable to run when starting container. Example: /bin/sh
envlist (*string*; Default:)	list of environmental variables (configured under */container envs*) to be used with container
file (*string; Default:*)	container *tar.gz tarball if the container is imported from a file
hostname (*string; Default:*)	
interface (*string; Default:*)	veth interface to be used with the container
logging (*string; Default:*)	if set to yes, all container-generated output will be shown in the RouterOS log
mounts (*string; Default:*)	mounts from /container/mounts/ sub-menu to be used with this container
remote-image (*string; Default:*)	the container image name to be installed if an external registry is used (configured under /container/config set registry-url=...)
root-dir (*string; Default:*)	used to save container store outside main memory
stop-signal (*string ; Default:*)	
workdir (*string*; Default:)	the working directory for cmd entrypoint

Container configuration

```
/container/config/
```

Property	Description
registry-url	external registry url from where the container will be downloaded
tmpdir	container extraction directory
ram-high	RAM usage limit. (0 for unlimited)
username	Specifies the username for authentication (starting from ROS 7.8)
password	Specifies the password for authentication (starting from ROS 7.8)

Container use example

Prerequisites:

a. RouterOS device with RouterOS v7.4beta or later and **installed Container package**
b. Physical access to a device to enable container mode
c. Attached hard drive or USB drive for storage - formatted as ext3/ext4

Enable Container mode

Device-mode limits container use by default, before granting container mode access - make sure your device is fully secured.

enable container mode

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a press of the reset button, or a cold reboot, if using container on X86.

Create network

Add veth interface for the container:

```
/interface/veth/add name=veth1 address=172.17.0.2/24 gateway=172.17.0.1
```

Create a bridge for containers and add veth to it:

```
/interface/bridge/add name=containers
/ip/address/add address=172.17.0.1/24 interface=containers
/interface/bridge/port add bridge=containers interface=veth1
```

Setup NAT for outgoing traffic:

```
/ip/firewall/nat/add chain=srcnat action=masquerade src-address=172.17.0.0/24
```

Add environment variables and mounts (optional)

Create environment variables for container(optional):

```
/container/envs/add name=pihole_envs key=TZ value="Europe/Riga"
/container/envs/add name=pihole_envs key=WEBPASSWORD value="mysecurepassword"
/container/envs/add name=pihole_envs key=DNSMASQ_USER value="root"
```

Define mounts (optional):

```
/container/mounts/add name=etc_pihole src=disk1/etc dst=/etc/pihole
/container/mounts/add name=dnsmasq_pihole src=disk1/etc-dnsmasq.d dst=/etc/dnsmasq.d
```

⚠ `src=` points to RouterOS location (could also be `src=disk1/etc_pihole` if, for example, You decide to put configuration files on external USB media), `dst=` points to defined location (consult containers manual/wiki/github for information on where to point). If `src` directory does not exist on first time use then it will be populated with whatever container have in `dst` location.

Add container image

If You wish to see container output in log - add `logging=yes` when creating a container, root-dir should point to an external drive formatted in ext3 or ext4. It's not recommended to use internal storage for containers.
There are multiple ways to add containers:

a) get an image from an external library

Set registry-url (for downloading containers from Docker registry) and set extract directory (tmpdir) to attached USB media:

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=disk1/pull
```

pull image:

```
 /container/add remote-image=pihole/pihole:latest interface=veth1 root-dir=disk1/pihole mounts=dnsmasq_pihole,
etc_pihole envlist=pihole_envs
```

The image will be automatically pulled and extracted to root-dir, status can be checked by using

```
/container/print
```

b) import image from PC

These links are `latest` as of 16th of June, 2022. Please make sure to download the right version that matches Your RouterOS device's architecture. Update sha256 sum from docker hub to get the latest image files

```
arm64:
        docker pull pihole/pihole:latest@sha256:4cef8a7b32d318ba218c080a3673b56f396d2e2c74d375bef537ff5e41fc4638
    docker save pihole/pihole > pihole.tar

arm
        docker pull pihole/pihole:latest@sha256:684c59c7c057b2829d19d08179265c79a9ddabf03145c1e2fad2fae3d9c36a94
    docker save pihole/pihole > pihole.tar

amd64
        docker pull pihole/pihole:latest@sha256:f56885979dcffeb902d2ca51828c92118199222ffb8f6644505e7881e11eeb85
    docker save pihole/pihole > pihole.tar
```

After the file has been downloaded and extracted - upload it to Your RouterOS device. Create a container from tar image

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs root-dir=disk1/pihole mounts=dnsmasq_pihole,
etc_pihole hostname=PiHole
```

c) build an image on PC

Steps for Linux systems

To use Dockerfile and make your own docker package - docker needs to be installed as well as buildx or other builder toolkit.

Easiest way is to download and install Docker Engine:
https://docs.docker.com/engine/install/

After install check if extra architectures are available:

```
docker buildx ls
```

should return:

```
NAME/NODE DRIVER/ENDPOINT STATUS  PLATFORMS
default * docker
  default default         running linux/amd64, linux/arm64, linux/riscv64, linux/ppc64le, linux/s390x, linux
/386, linux/arm/v7, linux/arm/v6
```

If not - install extra architectures:

```
docker run --privileged --rm tonistiigi/binfmt --install all
```

pull or create your project with Dockerfile included and build, extract image (adjust --platform if needed):

```
git clone https://github.com/pi-hole/docker-pi-hole.git
cd docker-pi-hole
docker buildx build  --no-cache --platform arm64 --output=type=docker -t pihole .
docker save pihole > pihole.tar
```

Upload *pihole.tar* to Your RouterOS device.

Images and objects on the Linux system can be `pruned`

Create a container from the tar image

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs mounts=dnsmasq_pihole,etc_pihole
hostname=PiHole
```

Start container

Make sure container has been added and `status=stopped` by using `/container/print`

```
/container/start 0
```

You should be able to access the PiHole web panel by navigating to `http://172.17.0.2/admin/` in your web browser.

Forward ports to internal container

Ports can be forwarded using dst-nat (where 192.168.88.1 routers IP address):

```
/ip firewall nat
add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=80 protocol=tcp to-addresses=172.17.0.2 to-
ports=80
```

For Pihole container - set DNS server to containers veth interface IP address -

```
/ip dns set servers=172.17.0.2
```

or change DHCP servers settings to serve Pihole DNS

Tips and tricks

- Containers use up a lot of disk space, USB/SATA,NVMe attached media is highly recommended. For devices with USB ports - USB to SATA adapters can be used with 2.5" drives - for extra storage and faster file operations.
- RAM usage can be limited by using:

```
/container/config/set ram-high=200M
```

this will soft limit RAM usage - if a RAM usage goes over the high boundary, the processes of the cgroup are throttled and put under heavy reclaim pressure.

- For starting containers after router reboot use start-on-boot option (starting from 7.6beta6)

```
/container/print
 0 name="2e679415-2edd-4300-8fab-a779ec267058" tag="test_arm64:latest" os="linux" arch="arm"
interface=veth2
   root-dir=disk1/alpine mounts="" dns="" logging=yes start-on-boot=yes status=running

/container/set 0 start-on-boot=yes
```

- It is possible to get to **running** container shell:

```
/container/shell 0
```

- Enable logging to get output from container:

```
/container/set 0 logging=yes
```

- Starting from 7.11beta5 version multiple addresses and ipv6 addresses can be added:

```
interface/veth add address=172.17.0.3/16,fd8d:5ad2:24:2::2/64 gateway=172.17.0.1 gateway6=fd8d:5ad2:24:
2::1
```

Container - freeradius server

Introduction

The introduction of the container feature into the RouterOS made it possible to run all kinds of servers for all sorts of tasks inside the router. This is especially relevant for people, who want to reduce the number of devices in their network. Instead of running a server on a separate device/machine, why not run it inside the router?

Radius is short for Remote Authentication Dial-In User Service. RouterOS has a RADIUS client feature supported that can authenticate for HotSpot, PPP, PPPoE, PPTP, L2TP, and ISDN connections. Basically, this feature allows you to connect RouterOS to a Radius Server, and then, utilize the user database from the server for client authentication.

In our example, we will showcase **freeradius/freeradius-server** image installation.

Summary

Sub-menu: `/container`

note: **container** package is required.

Make sure to study our container guide before proceeding with the configuration. Make sure to check the disclaimer and requirements sections to understand all the risks and necessary steps you might be required to do.

At the time, when the guide was published, the image was available for linux/**amd64** OS/architecture **only**. Meaning, you are not able to run this scenario on our arm32-bit and arm64-bit architecture RouterOS devices. For arm64, arm you will need to make your own container from FreeRADIUS source.

You can only run it on Cloud Hosted Router, CHR, or x86 installation.

To help you set up a CHR in a Virtual Box, please check our youtube tutorial, or Make your own x86 router.

> This guide demonstrates a basic example! The tests were performed in a local environment! This guide is meant for basic RADIUS "testing" purposes! Not all "freeradius" features were tested!

Configuration

Container mode

Enable container mode:

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a cold reboot if using the container on X86.

Networking

Add veth interface for the container:

```
/interface/veth/add name=veth3 address=172.17.0.2/24 gateway=172.17.0.1
```

Create a bridge for the container, assign an IP network to it, and add veth to the bridge:

```
/interface/bridge/add name=dockerfreeradius
/ip/address/add address=172.17.0.1/24 interface=dockerfreeradius
/interface/bridge/port add bridge=dockerfreeradius interface=veth3
```

Setup NAT for outgoing traffic if required:

```
/ip/firewall/nat/add chain=srcnat action=masquerade src-address=172.17.0.0/24
```

Getting image

To simplify the configuration, we will get the image from an external library but you can also import it via the .tar file.

Make sure that you have "Registry URL" set accordingly, limit RAM usage (if necessary), and set up a directory for the image:

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=pull
```

Pull image with the help of the command:

```
/container/add remote-image=freeradius/freeradius-server:latest interface=veth3 root-dir=freeradius logging=yes
cmd="-X"
```

where `cmd="-X"` enables debug logging (per the "freeradius" documentation).

After running the command, RouterOS should start "extracting" the package. Check "File System" for newly created folders and monitor container status with the command `/container/print`.

Starting the container

After you make sure that the container has been added and the status changed to `status=stopped` after using `/container/print` → you can initiate it:

```
/container/start 0
```

Altering the server's configuration files

To access the server's configuration files (**clients.conf** and **authorize**), we will need to use SFTP (file transfer over SSH) protocol, so make sure that SSH service is enabled.

Open your command terminal ("CMD", as Administrator, for Windows users, or "Linux Shell or Command Terminal" for Linux users) and navigate it to the directory where you want to download the configuration files. For example, to "radius" folder on your "Desktop":

```
C:\WINDOWS\system32>cd C:\Users\Administrator\Desktop\radius

C:\Users\Administrator\Desktop\radius>
```

Initiate SFTP to the device's IP address:

```
C:\Users\DenissPC\Desktop\radius>sftp admin@10.55.8.53
admin@10.55.8.53's password:
Connected to 10.55.8.53.
sftp>
```

Go to the server's configuration file folder (use `dir` or `ls` command to see the content of the folder you are in and `cd` command to go to the folder of our choice).

The first file, "clients.conf" allows you to define RADIUS clients. Per the "freeradius" documentation, it should be under the "/etc/freeradius" directory...so, navigate there and use `get` command to download it:

```
sftp> dir
freeradius          pub                     pull                    skins
sftp> cd freeradius/etc/freeradius
sftp> dir
README.rst          certs               clients.conf        dictionary          experimental.conf
hints
huntgroups          mods-available      mods-config         mods-enabled        panic.gdb           policy.
d
proxy.conf          radiusd.conf        sites-available     sites-enabled       templates.conf      trigger.
conf
users
sftp> get clients.conf
Fetching /freeradius/etc/freeradius/clients.conf to clients.conf
/freeradius/etc/freeradius/clients.conf                                             100% 8323     1.2MB/s
00:00
```

Open "**clients.conf**" via your preferred text editor (notepad or any other). You can study the file to see all the options that you have (additionally, check freeradius.org). This example shows a basic setup, so, we will just overwrite the whole file with the lines shown below:

```
client new {
    ipaddr = 0.0.0.0/0
    secret = client_password
}
```

where we indicate, that our radius client can connect using any possible IP address (**ipaddr=0.0.0.0/0** ensures that, but you also can change it to the actual ip address/mask of your radius client if you require to do so) and that our secret is "client_password" (you can change it to any other secret).

Save the file/overwrite it.

The second file, "authorize" allows you to set up users. Per the "freeradius" documentation, it should be under "/etc/freeradius/mods-config/files". Go there and `get` the file:

```
sftp> dir
freeradius          pub                     pull                    skins
sftp> cd freeradius/etc/freeradius/mods-config/files
sftp> dir
accounting  authorize   dhcp        pre-proxy
sftp> get authorize
Fetching /freeradius/etc/freeradius/mods-config/files/authorize to authorize
/freeradius/etc/freeradius/mods-config/files/authorize                              100% 6594     1.1MB/s
00:00
```

Open "**authorize**" via your preferred text editor (notepad or any other). This example shows a basic setup, so, we will just uncomment (remove the "#" symbol from) the line shown below (leave the rest of the configuration/lines as they are):

```
bob        Cleartext-Password := "hello"
```

which creates a username "bob" and sets the password to "hello" (you can change the username and password).

Save the file/overwrite it.

Upload both files back/overwrite the default files with the help of the `put` command:

```
sftp> dir
freeradius          pub                     pull                    skins
sftp> cd freeradius/etc/freeradius
sftp> dir
README.rst          certs               clients.conf        dictionary          experimental.conf
hints
huntgroups          mods-available      mods-config         mods-enabled        panic.gdb           policy.
d
proxy.conf          radiusd.conf        sites-available     sites-enabled       templates.conf      trigger.
conf
users
sftp> put clients.conf
Uploading clients.conf to /freeradius/etc/freeradius/clients.conf
clients.conf                                                                      100%   67    22.3KB/s
00:00
sftp> cd mods-config/files
sftp> dir
accounting  authorize   dhcp        pre-proxy
sftp> put authorize
Uploading authorize to /freeradius/etc/freeradius/mods-config/files/authorize
authorize                                                                         100% 6626     1.6MB/s
00:00
```

Restart the container:

```
/container/stop 0
/container/start 0
```

Make sure to wait for the container to stop (`status=stopped` should be shown after using `/container/print` command) before initiating it again.

Result verification

In RouterOS, add a new RADIUS client configuration:

```
/radius/add service=login address=172.17.0.2 secret="client_password"
```

,where the `address` is the IP address of the veth3 interface, `secret` is the secret that we configured in the **clients.conf** file and `service` is the allowed service that you wish to use.

Allow "login" with RADIUS users via the command:

```
/user/aaa/set use-radius=yes
```

We have allowed the "login" service for the RADIUS and we can test it using ssh/winbox/webfig connection. For SSH test, issue the command (where you need to indicate the device's management IP and input bob's password "hello" after):

```
/system/ssh 10.55.8.53 user=bob
```

You should be able to verify, that the terminal user changed from "admin@MikroTik" to "bob@MikroTik":

```
[admin@MikroTik] > /system/ssh 10.55.8.53 user=bob
password:
hello









  MMM      MMM       KKK                          TTTTTTTTTTT      KKK
  MMMM    MMMM       KKK                          TTTTTTTTTTT      KKK
  MMM MMMM MMM  III  KKK  KKK  RRRRRR     OOOOOO      TTT     III  KKK  KKK
  MMM  MM  MMM  III  KKKKK     RRR  RRR  OOO  OOO     TTT     III  KKKKK
  MMM      MMM  III  KKK KKK   RRRRRR    OOO  OOO     TTT     III  KKK KKK
  MMM      MMM  III  KKK  KKK  RRR  RRR   OOOOOO      TTT     III  KKK  KKK

  MikroTik RouterOS 7.8alpha173 (c) 1999-2023       https://www.mikrotik.com/

Press F1 for help

[bob@MikroTik] >
```

If you issue the command `/user/active/print`:

```
/user/active/print
Flags: R - RADIUS
Columns: WHEN, NAME, ADDRESS, VIA
#   WHEN                 NAME   ADDRESS     VIA
0   feb/16/2023 16:31:21  admin  xx.xx.xx.xx  winbox
1   feb/16/2023 16:38:46  admin  xx.xx.xx.xx  console
2 R feb/16/2023 16:38:53  bob    10.55.8.53  ssh
```

you will be able to verify, that a new user "bob" is "active" and has a flag "R" assigned, which indicates it is a RADIUS user.

Container - HomeAssistant

Introduction

The introduction of the container feature into the RouterOS made it possible to run all kinds of servers for all sorts of tasks inside the router. This is especially relevant for people, who want to reduce the number of devices in their network. Instead of running a server on a separate device/machine, why not run it inside the router?

In this guide, we will showcase how to install and host Home-Assistant server on RouterOS.

Home-Assistant is a very popular platform that is used to collect statistics from different sensors and supports different integrations.

Summary

Make sure to study our container guide before proceeding with the configuration. Make sure to check the disclaimer and requirements sections to understand all the risks and necessary steps you might be required to do.

You can find supported architectures by following the link.

At the time, when the guide was published, **home-assistant** image was available for ARM32, ARM64, and AMD64 (CHR and x86) devices.

Based on the information from their site, recommended minimal requirements are:

- 2 GB RAM
- 32 GB Storage

Recommended does not mean set in stone, and we will be running this on L009UiGS that has 512 MB RAM and a USB slot for the required storage. It is advised using recommended hardware specifications, but for testing, and low amounts of data, you can get by with 512+ MB RAM.

Container configuration

Sub-menu: `/container`

note: **container** package is required.

Container mode

Enable container mode:

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a press of the reset button, or a cold reboot, if using container on X86.

Networking

Add veth interface:

```
/interface/veth/add name=veth2 address=172.19.0.2/24 gateway=172.19.0.1
```

Create a bridge for both containers and add veth interfaces to it:

```
/interface/bridge/add name=ha
/ip/address/add address=172.19.0.1/24 interface=ha
/interface/bridge/port add bridge=ha interface=veth2
```

Forward TCP 8123 for home-assistant management (where 192.168.88.1 is the device's LAN IP address) if NAT is required (optional):

```
/ip firewall nat add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=8123 protocol=tcp to-
addresses=172.19.0.2 to-ports=8123
```

Environment variables and mounts

Per the home-assist documentation, define mounts for the configuration files (where, `/usb1` is our external USB storage folder):

```
/container mounts add dst=/config name=ha_config src=/usb1/ha_config
```

Create an environmental variable for home-assistant:

```
/container envs add key=TZ name=ha_env value=America/Los_Angeles
```

Getting image

To simplify the configuration, we will get the images from an external library.

Make sure that you have "Registry URL" set accordingly, limit RAM usage (if necessary), and set up a directory for the images:

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=/usb1/pull
```

Pull home-assistant image and wait for it to be extracted:

```
/container/add remote-image=homeassistant/home-assistant:latest interface=veth2 root-dir=/usb1/ha
mounts=ha_config envlist=ha_env logging=yes
```

After running the command, RouterOS should start "extracting" the package. Check "File System" for newly created folders and monitor container status with the command `/container/print`.

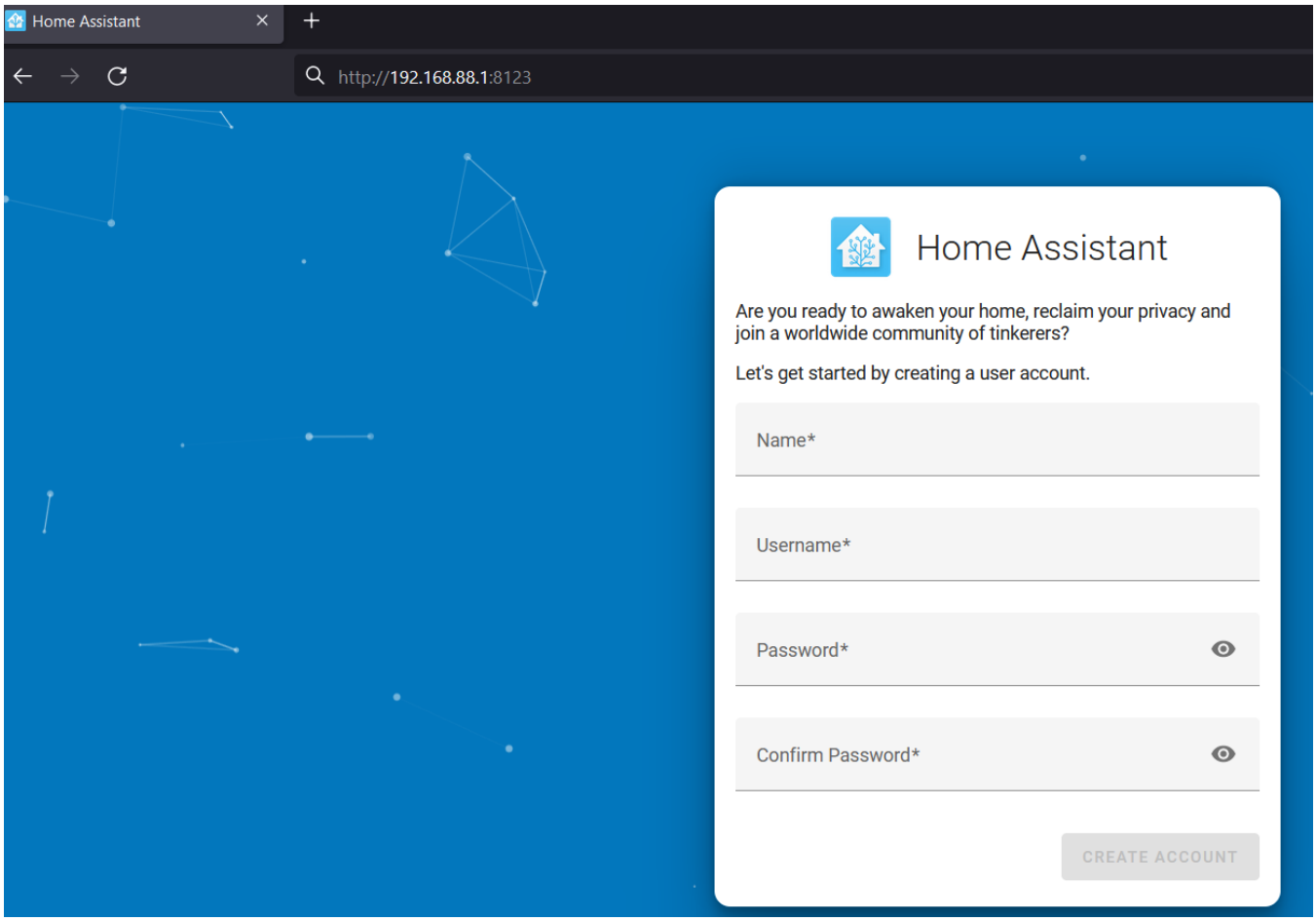
Starting the container

After you make sure that the container has been added and the status changed to `status=stopped` after using `/container/print` → you can initiate it:

```
/container/start 0
```

Home-Assistant setup

Open your preferred web browser and access the Home-Assistant management portal by specifying management port ":8123":



Proceed with the setup. More information is explained in the Home-Assistant onboarding guide.

Resources

Just running the **Home Assistant** on L009 (without any load/traffic) takes up ~300 MB of RAM:

```
/system resource print
                   uptime: 4m27s
                  version: 7.13.3 (stable)
               build-time: Jan/24/2024 13:16:46
         factory-software: 7.10
              free-memory: 143.0MiB
             total-memory: 448.0MiB
                      cpu: ARM
```

With the load, it will increase, but for testing, on this specific board, it should be enough.

Container - mosquitto MQTT server

Introduction

The introduction of the container feature into the RouterOS made it possible to run all kinds of servers for all sorts of tasks inside the router. This is especially relevant for people, who want to reduce the number of devices in their network. Instead of running a server on a separate device/machine, why not run it inside the router?

In this guide, we will showcase how to install a basic MQTT broker (or in other words, server) called eclipse-mosquitto. MQTT protocol is a very popular choice, especially in IoT topologies. It is an open OASIS and ISO standard lightweight, publish-subscribe network protocol that transports messages between devices. A typical topology consists of an MQTT publisher (a device that sends information), an MQTT broker (a server where the data is stored), and an MQTT subscriber (a device that listens to the data published on the server).

RouterOS supports MQTT publish, subscribe feature, and, now, we can also run the MQTT broker as well.

The image that we are going to use, can be found by following the hub.docker link.

Summary

Make sure to study our container guide before proceeding with the configuration. Make sure to check the disclaimer and requirements sections to understand all the risks and necessary steps you might be required to do.

You can find supported architectures by following the link.

At the time, when the guide was published, **eclipse-mosquitto** image was available for ARM32, ARM64, and AMD64 (CHR and x86) devices. In this example, we will run it on an ARM32 architecture device → RB1100AHx4.

A very basic and **quick** configuration will be shown. Make sure to check mosquitto documentation page for more information about additional options and settings you can implement. If you want to use it for production, please **make sure to harden the security** in any way possible:

- Firewall, so that access to the container is allowed only from your trusted IP addresses;
- Increasing security from the mosquitto broker/server-side → use strong passwords, non-standard ports ...etc;
- Use SSL MQTT.

Container configuration

Sub-menu: `/container`

note: **container** package is required.

Container mode

Enable container mode:

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a press of the reset button, or a cold reboot, if using container on X86.

Networking

Add veth interface for the container:

```
/interface/veth/add name=veth2 address=172.19.0.2/24 gateway=172.19.0.1
```

Create a bridge for containers and add veth to it:

```
/interface/bridge/add name=msqt
/ip/address/add address=172.19.0.1/24 interface=msqt
/interface/bridge/port add bridge=msqt interface=veth2
```

Forward TCP 1883 for non-SSL MQTT (where 192.168.88.1 is the device's LAN IP address) for testing purposes if NAT is required (optional):

```
/ip firewall nat add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=1883 protocol=tcp to-
addresses=172.19.0.2 to-ports=1883
```

Environment variables and mounts

Per the eclipse-mosquitto docker hub, define a mount for the configuration file. We will mount not just the configuration file, but the whole folder, because, for SSL MQTT, we will need to upload certificates into the folder as well:

```
/container mounts add src=/mosquitto_mounted dst=/mosquitto/config name=msqt_config
```

Getting image

To simplify the configuration, we will get the image from an external library but you can also import it via the .tar file.

In this example, we will use the device's own storage. RB1100AHx4 has 128 MB disk space and a basic mosquitto installation should not take up more than ~15 MB.

Make sure that you have "Registry URL" set accordingly, limit RAM usage (if necessary), and set up a directory for the image:

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=pull
```

Pull image:

```
/container/add remote-image=eclipse-mosquitto:latest interface=veth2 root-dir=mosquitto mounts=msqt_config
logging=yes
```

After running the command, RouterOS should start "extracting" the package. Check "File System" for newly created folders and monitor container status with the command `/container/print`.

Setting up mosquitto configuration file

To get the **mosquttio.conf** file, we will need to use SFTP (file transfer over SSH) protocol, so make sure that SSH service is enabled. You can also use FTP.

Open your command terminal ("CMD", as Administrator, for Windows users, or "Linux Shell or Command Terminal" for Linux users) and navigate it to the directory where you want to download the configuration file. For example, to the "Container" folder on your "Desktop":

```
C:\WINDOWS\system32>cd C:\Users\Administrator\Desktop\Container

C:\Users\Administrator\Desktop\Container>
```

Initiate SFTP to the device's IP address:

```
C:\Users\Administrator\Desktop\Container>sftp admin@192.168.88.1
The authenticity of host '192.168.88.1 (192.168.88.1)' can't be established.
RSA key fingerprint is SHA256:lfxxs+xMrXlvP7hiHi9ZAEZlPi6/c5US+r6J7ljhkaA.
Are you sure you want to continue connecting (yes/no/[fingerprint])?yes
Warning: Permanently added '192.168.88.1' (RSA) to the list of known hosts.
Connected to 192.168.88.1.
sftp>
```

Go to the mosquitto configuration file folder (use `dir` or `ls` command to see the content of the folder you are in and `cd` command to go to the folder of our choice). By default, the configuration is loaded from the "/mosquitto/config/mosquitto.conf", so, navigate there and use `get` command to download it:

```
sftp> cd mosquitto/mosquitto/config
sftp> dir
mosquitto.conf
sftp> get mosquitto.conf
Fetching /mosquitto/mosquitto/config/mosquitto.conf to mosquitto.conf
/mosquitto/mosquitto/config/mosquitto.conf
```

Open "**mosquitto.conf**" via your preferred text editor (notepad or any other), and just overwrite it with two lines shown below:

> In this section, we will configure a basic non-SSL MQTT setup for testing purposes. Non-SSL MQTT is not secure for a production environment unless you are certain the required security/restrictions are in place.
>
> For a production environment, topologies where the MQTT traffic can be captured/sniffed and/or topologies where the MQTT traffic is routed directly via the internet (not locally), use SSL MQTT. Check the SSL MQTT section for more information.

```
listener 1883
allow_anonymous true
```

- The first line, **listener 1883**, will make the installation listen for incoming network connection on the specified port.
- The second line, **allow_anonymous true**, determines whether clients that connect without providing a username are allowed to connect.

Overwrite the file using the same **mosquitto.conf** name.

After you have created your own custom configuration file, upload it into the mounted directory/folder "**mosquitto_mounted**". If you have not run the container yet, you will not have the "**mosquitto_mounted**" folder and you can create it manually. If you did run it (`/container start 0`), it should have been created automatically:

```
sftp> dir
mosquitto           mosquitto_mounted   pub                 pull                skins
```

Use SFTP from the directory where the edited mosquitto.conf file is located and `put` it into the mounted directory:

```
C:\Users\Administrator\Desktop\Container>dir
 Directory of C:\Users\Administrator\Desktop\Container

02/03/2023  12:09 PM    <DIR>          .
02/03/2023  12:09 PM    <DIR>          ..
02/03/2023  12:09 PM            40,449 mosquitto.conf
               1 File(s)         40,449 bytes
               2 Dir(s)  76,166,430,720 bytes free

C:\Users\Administrator\Desktop\Container>sftp admin@192.168.88.1
Connected to 192.168.88.1.
sftp> dir
mosquitto           mosquitto_mounted   pub                 pull                skins
sftp> cd mosquitto_mounted
sftp> put mosquitto.conf
Uploading mosquitto.conf to /mosquitto_mounted/mosquitto.conf
mosquitto.conf                                                                  100%  162    40.5KB/s
00:00
```

Restart the container:

```
[admin@MikroTik] > /container/stop 0
[admin@MikroTik] > /container/start 0
```

Make sure to wait for the container to stop (`status=stopped` should be shown after using `/container/print` command) before initiating it again.

Starting the container

After you make sure that the container has been added and the status changed to `status=stopped` after using `/container/print` → you can initiate it:

```
/container/start 0
```

If you have enabled container logging, you would see something like this in the Logs section:

```
 12:12:46 container,info,debug 1707214366: mosquitto version 2.0.18 starting
 12:12:46 container,info,debug 1707214366: Config loaded from /mosquitto/config/mosquitto.conf.
 12:12:46 container,info,debug 1707214366: Opening ipv4 listen socket on port 1883.
 12:12:46 container,info,debug 1707214366: Opening ipv6 listen socket on port 1883.
 12:12:46 container,info,debug 1707214366: mosquitto version 2.0.18 running
```

MQTT publish and subscribe

Sub-menu: `/iot mqtt`

note: **iot** package is required.

Add an MQTT broker:

```
/iot/mqtt/brokers/add name=mosquitto username=test address=172.19.0.2
```

Subscribe to the MQTT broker and the required topic:

```
/iot/mqtt/subscribe broker=mosquitto topic=test/topic
```

Publish a static MQTT message:

```
/iot/mqtt/publish broker="mosquitto" topic="test/topic" message="{\"test\":\"123\"}"
```

Check subscriptions for received messages:

```
/iot/mqtt/subscriptions/recv/print
 0 broker=mosquitto topic="test/topic" data="{"test":"123"}"
   time=2023-07-12 10:01:40
```

You can also check the container logs (if enabled), to confirm the mosquitto is operational:

```
 12:47:28 container,info,debug 1675421248: New connection from 172.19.0.1:42240 on port 1883.
 12:47:28 container,info,debug 1675421248: New client connected from 172.19.0.1:42240 as MTD8580EC793C4 (p2,
c1, k60, u'test').
 12:47:38 container,info,debug 1675421258: Client MTD8580EC793C4 disconnected.
```

SSL MQTT

Using **non-SSL MQTT** for a production environment **is not secure**. One can easily capture/sniff the packet exchange between the broker and the publisher and, as a result, will be able to obtain user credentials and other sensitive information.

To increase security, use SSL MQTT.

The first step is to generate the certificates. In this example, we will use a simple Root CA scenario (with no device/client certificate requirement).

Use the official mosquitto-tls user guide for the step-by-step.

Server configuration

You should have generated ca.crt (Certificate Authority file), server.crt (server certificate) and server.key (server's key):

```
C:\Users\Administrator\Desktop\Container>dir
 Directory of C:\Users\Administrator\Desktop\Container

07/12/2023  10:58 AM    <DIR>          .
07/12/2023  10:58 AM    <DIR>          ..
07/12/2023  10:56 AM             1,322 ca.crt
07/12/2023  10:56 AM             1,854 ca.key
07/12/2023  09:57 AM                35 mosquitto.conf
07/12/2023  10:58 AM             1,164 server.crt
07/12/2023  10:57 AM               960 server.csr
07/12/2023  10:56 AM             1,704 server.key
               6 File(s)          7,039 bytes
               2 Dir(s)  52,401,184,768 bytes free
```

Open mounted "**mosquitto.conf**" via your preferred text editor (notepad or any other), and just overwrite it with the lines shown below:

```
tls_version tlsv1.2
port 8883
allow_anonymous true
cafile /mosquitto/config/ca.crt
keyfile /mosquitto/config/server.key
certfile /mosquitto/config/server.crt
```

- **tls_version** line sets minimal TLS version;
- **listener 8883**, will make the installation listen for incoming network connection on the specified port;
- **allow_anonymous true**, determines whether clients that connect without providing a username are allowed to connect;

We are using a basic SSL configuration for testing purposes. **allow_anonymous true** is not a secure setting for the production environment.

- **cafile /mosquitto/config/ca.crt** line specifies a path to the CA certificate file;
- **keyfile /mosquitto/config/server.key** line specifies a path to the server key file;
- **certfile /mosquitto/config/server.crt** line specifies a path to the server certificate file.

Upload the certificate files, and updated SSL-ready mosquitto.conf file into the mounted folder "**mosquitto_mounted**":

```
C:\Users\Administrator\Desktop\Container>sftp admin@192.168.88.1
Connected to 192.168.88.1.
sftp> cd mosquitto_mounted
sftp> dir
mosquitto.conf
sftp> put ca.crt
Uploading ca.crt to /mosquitto_mounted/ca.crt
ca.crt                                                                    100% 1322   323.0KB/s
00:00
sftp> put server.crt
Uploading server.crt to /mosquitto_mounted/server.crt
server.crt                                                                100% 1164   227.3KB/s
00:00
sftp> put server.key
Uploading server.key to /mosquitto_mounted/server.key
server.key                                                                100% 1704   415.7KB/s
00:00
sftp> dir
ca.crt          mosquitto.conf   server.crt       server.key
sftp> put mosquitto.conf
Uploading mosquitto.conf to /mosquitto_mounted/mosquitto.conf
mosquitto.conf                                                            100%  162    32.2KB/s
00:00
```

Restart the container:

```
[admin@MikroTik] > /container/stop 0
[admin@MikroTik] > /container/start 0
```

Confirm that the broker listens on port 8883 using the logs:

```
 11:20:41 container,info,debug 1689160841: mosquitto version 2.0.15 starting
 11:20:41 container,info,debug 1689160841: Config loaded from /mosquitto/config/mosquitto.conf.
 11:20:41 container,info,debug 1689160841: Opening ipv4 listen socket on port 8883.
 11:20:41 container,info,debug 1689160841: Opening ipv6 listen socket on port 8883.
 11:20:41 container,info,debug 1689160841: mosquitto version 2.0.15 running
 11:22:24 system,info,account user admin logged in from 10.5.217.34 via local
```

Testing the connection

Upload CA certificate (**ca.crt**) into RouterOS, into the device's "File List":

```
/file print
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME               TYPE             SIZE  CREATION-TIME
0  skins              directory              1970-01-01 03:00:02
1  pub                directory              2023-01-04 11:05:04
2  disk7              disk                   2023-07-12 09:52:07
3  mosquitto          container store        2023-07-12 09:52:09
4  mosquitto_mounted  container store        2023-07-25 16:38:37
5  pull               directory              2023-07-12 09:52:09
6  ca.crt             .crt file        1322  2023-07-12 11:28:23
```

Import the certificate:

```
/certificate/import file-name=ca.crt passphrase=""
```

Add MQTT broker for SSL connection:

```
/iot/mqtt/brokers/add name=mosquittoSSL username=test address=172.19.0.2 port=8883 ssl=yes
```

Subscribe to the MQTT broker and the required topic:

```
/iot/mqtt/subscribe broker=mosquittoSSL topic=test/topic
```

Publish a static MQTT message:

```
/iot/mqtt/publish broker="mosquittoSSL" topic="test/topic" message="{\"test\":\"123\"}"
```

Check subscriptions for received messages:

```
/iot/mqtt/subscriptions/recv/print
 0 broker=mosquittoSSL topic="test/topic" data="{"test":"123"}"
   time=2023-07-12 10:20:40
```

Container - ThingsBoard MQTT/HTTP server

Introduction

The introduction of the container feature into the RouterOS made it possible to run all kinds of servers for all sorts of tasks inside the router. This is especially relevant for people, who want to reduce the number of devices in their network. Instead of running a server on a separate device/machine, why not run it inside the router?

A lot of users need a server that is able to gather the data, store it and display it in a way it is easy to understand. This is where a platform like ThingsBoard can come into play.

It is primarily positioned as an IoT platform and you can find all sorts of use cases for it that they demonstrate in the link.

The most appealing part, from the RouterOS user standpoint, is that it can be used as an MQTT server (MQTT broker) or an HTTP server. Meaning, you can use MQTT publish or HTTP post to post the data. You can find ThingsBoard MQTT API guide by using the link here and HTTP API by using the link here.

In short, you can utilize scripting to collect RouterOS statistics (like uptime, GPS coordinates, packet statistics, and almost anything else that you print into the terminal), then store this information into variables and structure a JSON message out of those. You can, then send this message using MQTT or HTTP post to the ThingsBoard via a scheduler (that will run this script whenever you need it). You can find an example of a basic script that does it in this guide.

ThingsBoard will store and display the data with the help of widgets, which can be used to help you set up dashboards that visualize the data in graphs, tables, maps, and other ways.

There are 3 versions of the ThingsBoard instances available and each of them uses a different database:

- thingsboard/tb-postgres
- thingsboard/tb-cassandra
- thingsboard/tb

You can find more information in the ThingsBoard/docker documentation.

In our example, we will showcase **tb-postgres** - a single instance of ThingsBoard with PostgreSQL database.

Summary

Sub-menu: `/container`

note: **container** package is required.

RouterOS versions that are older than v7.8 will not be able to run this scenario.

Make sure to study our container guide before proceeding with the configuration. Make sure to check the disclaimer and requirements sections to understand all the risks and necessary steps you might be required to do.

In this example, we will run it on a Cloud Hosted Router, CHR device. To help you set it up in a Virtual Box, please check our youtube tutorial.

At the time, when the guide was published, **thingsboard/tb-postgres** image was available for linux/**arm64** and linux/**amd64** OS/architectures **only**. Meaning, you are not able to run this scenario on our arm32-bit architecture RouterOS devices.

There are a couple of parameters to keep in mind:

- You need to understand that it is a **server** and that you will need to have additional space for the data that is stored there and the image itself. In our tests, 8 GB of disk space was plenty enough but! you might want to consider adding more for real-life applications, especially if you are planning on running more containers. Just remember → it might be better to have a reserve.
- Same as with disk space, RAM memory is also important. Per the ThingsBoard documentation, when using a single instance of ThingsBoard with a PostgreSQL database, it is recommended to allocate at least 1GB of RAM and use a minimum load (a few messages per second). 2-4GB RAM is recommended. In other words, if you want to run it on a RouterBoard device, please understand, that you might not be able to achieve it on devices that have less than 1 GB RAM. That is why → consider having a device with more RAM memory to spare.

Go to the tips and tricks section to understand how to limit RAM.

Configuration

Container mode

Enable container mode:

```
/system/device-mode/update container=yes
```

You will need to confirm the device-mode with a press of the reset button, or a cold reboot, if using container on X86.

Networking

Add veth interface for the container:

```
/interface/veth/add name=veth1 address=172.18.0.2/24 gateway=172.18.0.1
```

Create a bridge for the container, assign an IP network to it, and add veth to the bridge:

```
/interface/bridge/add name=dockertb
/ip/address/add address=172.18.0.1/24 interface=dockertb
/interface/bridge/port add bridge=dockertb interface=veth1
```

Setup NAT for outgoing traffic:

```
/ip/firewall/nat/add chain=srcnat action=masquerade src-address=172.18.0.0/24
```

Forward TCP 9090 for HTTP management (the default HTTP port per ThingsBoard documentation):

> We suggest using HTTP access only when testing locally or through a VPN (when you are certain that the local network is safe).
>
> When you want to access container WEB management from the internet (from the public network/WAN), please, instead, consider using **HTTPS**.

```
/ip firewall nat add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=9090 protocol=tcp to-
addresses=172.18.0.2 to-ports=9090
```

In the `dst-address` field shown in DNAT (dst-nat) rule above, we use the device's local IP address. First, **use local IPs** (local access) **to set everything up and confirm that everything is working**.

> After going through the rest of the steps shown in this guide and verifying that the ThingsBoard management portal works locally → **further secure the setup**:
>
> - (a) make sure that all default ThingsBoard user credentials were changed/removed and strong passwords were implemented (reference ThingsBoard documentation);
> - (b) **enable HTTPS** (the steps will be explained later on in the guide);
> - (c) preferably change HTTPS port to a non-standard one (reference ThingsBoard documentation).
>
> Only when you increase the security and only then → you can consider enabling remote access from WAN (by using your public IP address in the `dst-address` field instead of the local IP used in the example above). Additionally, to further increase security, use `src-address` or `src-address-list` parameter, where you can input your trusted public source IP addresses (a list of known/trusted addresses that, for example, belong to your branch office from where you also want to have access to the server). Please understand that only you are responsible for the security. If you leave a door open, someone may exploit it. You need to have networking knowledge and understand the risks when setting up such scenarios.

Forward TCP 1883 for non-SSL MQTT (the default MQTT port used per ThingsBoard documentation):

> We suggest using non-SSL MQTT (TCP 1883) communication only when testing locally or through a VPN (when you are certain that the local network is safe).
>
> Please consider using **SSL MQTT (TCP port 8883)**, instead of non-SSL MQTT (TCP port 1883), for real-life applications, when it comes down to access from the internet (from the public network). If you use non-SSL MQTT, the communication between the client (MQTT publisher) and the server (MQTT broker) can be easily sniffed/packet captured, and that will compromise authentication data (such as client-ids, usernames and passwords).

```
/ip firewall nat add action=dst-nat chain=dstnat dst-address=192.168.88.1 dst-port=1883 protocol=tcp to-addresses=172.18.0.2 to-ports=1883
```

Same as with HTTP access, in the `dst-address` field shown in DNAT (dst-nat) rule above, we use the device's local IP address. First, **use local IPs** (local access) **to set everything up and confirm that everything is working**.

> After going through the rest of the steps shown in this guide and verifying that the ThingsBoard non-SSL MQTT communication works locally → **further secure the setup**:
>
> - (a) consider removing template devices from the ThingsBoard installation;
> - (b) **enable SSL MQTT** (the steps will be explained later on in the guide);
> - (c) preferably change MQTT port to a non-standard one (reference ThingsBoard documentation).
>
> When you enable SSL MQTT, you can consider opening TCP 8883 (which is the default SSL MQTT port) from WAN (by using your public IP address in the `dst-address` field instead of the local IP, and changing `dst-port` and `to-ports` from 1883 to 8883). Additionally, to further increase security, use `src-address` or `src-address-list` parameters, where you can set up your trusted public IP address list. As a result, only configured trusted IPs will be able to establish an MQTT connection with the ThingsBoard broker.

Environment variables and mounts

Check docker-thingsboard documentation for exact mounts and variables that need to be added.

Environment variables:

```
/container/envs/add name=tb_envs key=TB_QUEUE_TYPE value="in-memory"
```

Mounts:

```
/container/mounts/add name=mytb-data src=tb/mytb-data dst=/data
/container/mounts/add name=mytb-logs src=tb/mytb-logs dst=/var/log/thingsboard
```

Getting image

To simplify the configuration, we will get the image from an external library but you can also import it via the .tar file.

Make sure that you have "Registry URL" set accordingly, limit RAM usage (if necessary), and set up a directory for the image.

```
/container/config/set registry-url=https://registry-1.docker.io tmpdir=pull ram-high=2048.0MiB
```

Pull image:

```
/container/add remote-image=thingsboard/tb-postgres:latest interface=veth1 root-dir=ThingsBoard mounts=mytb-
data,mytb-logs envlist=tb_envs logging=yes
```

After running the command, RouterOS should start "extracting" the package. Check "File System" for newly created folders and monitor container status with the command `/container/print`.

Starting the container

After you make sure that the container has been added and the status changed to `status=stopped` after using `/container/print` → you can initiate it:

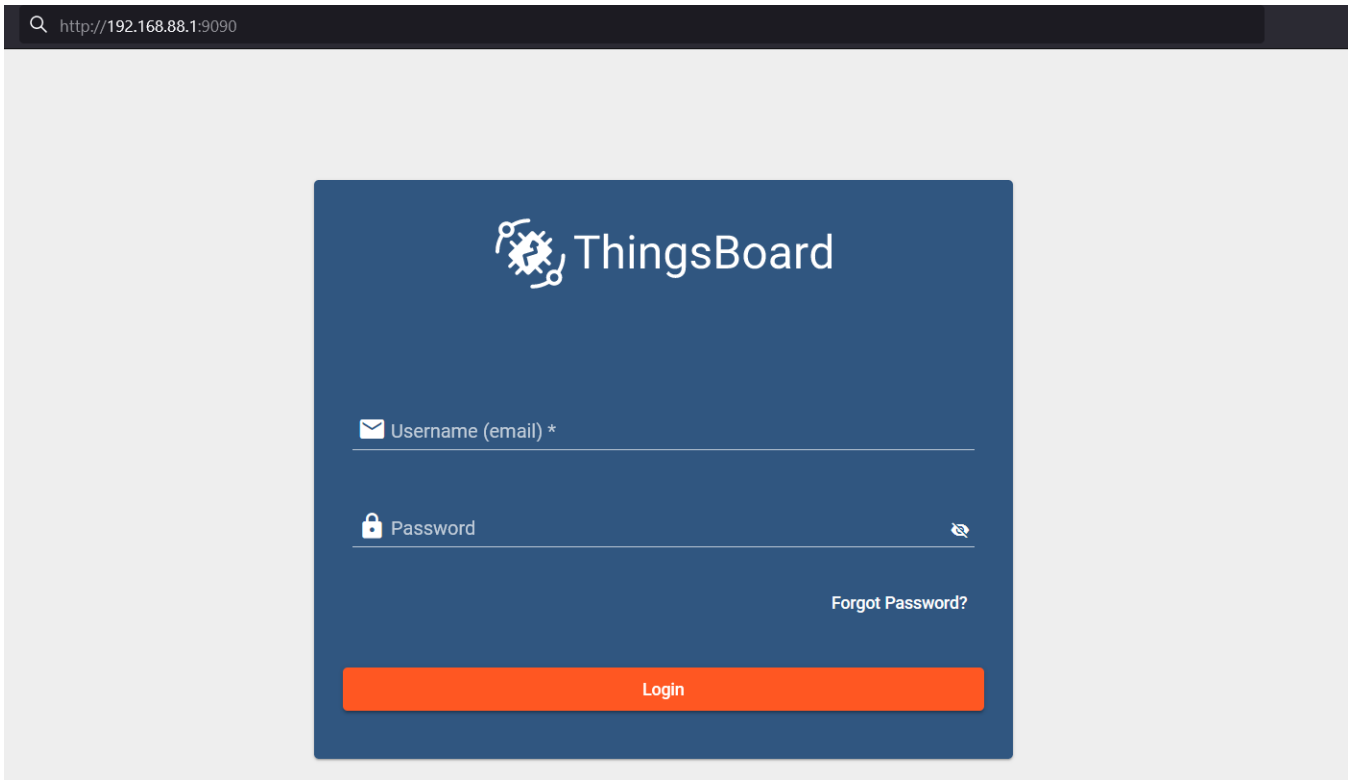
```
/container/start 0
```

Wait for a couple of minutes for the container to fully load.

Verification

Management access

After the container is started and installed, access it using any browser, by going to → http://192.168.88.1:9090 (where the IP address is the address used in the DNAT rule):



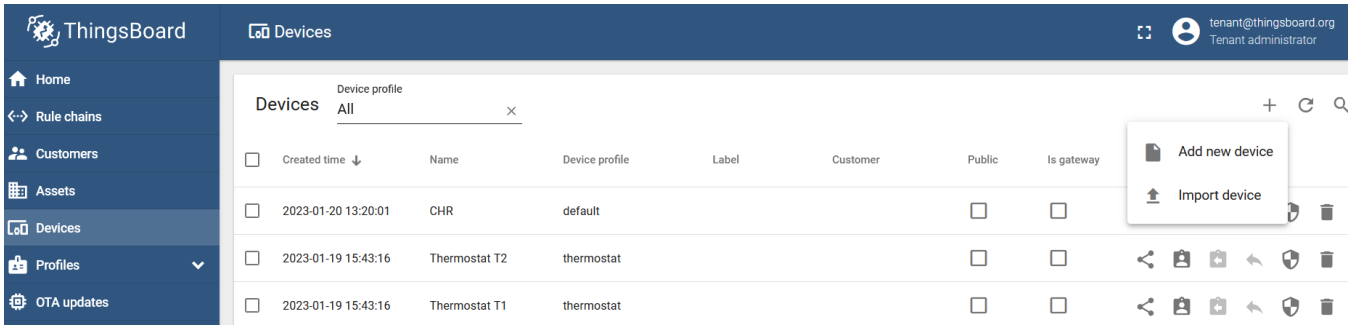
By default, credentials are (Username/Password):

- **Systen Administrator**: sysadmin@thingsboard.org / sysadmin
- **Tenant Administrator**: tenant@thingsboard.org / tenant

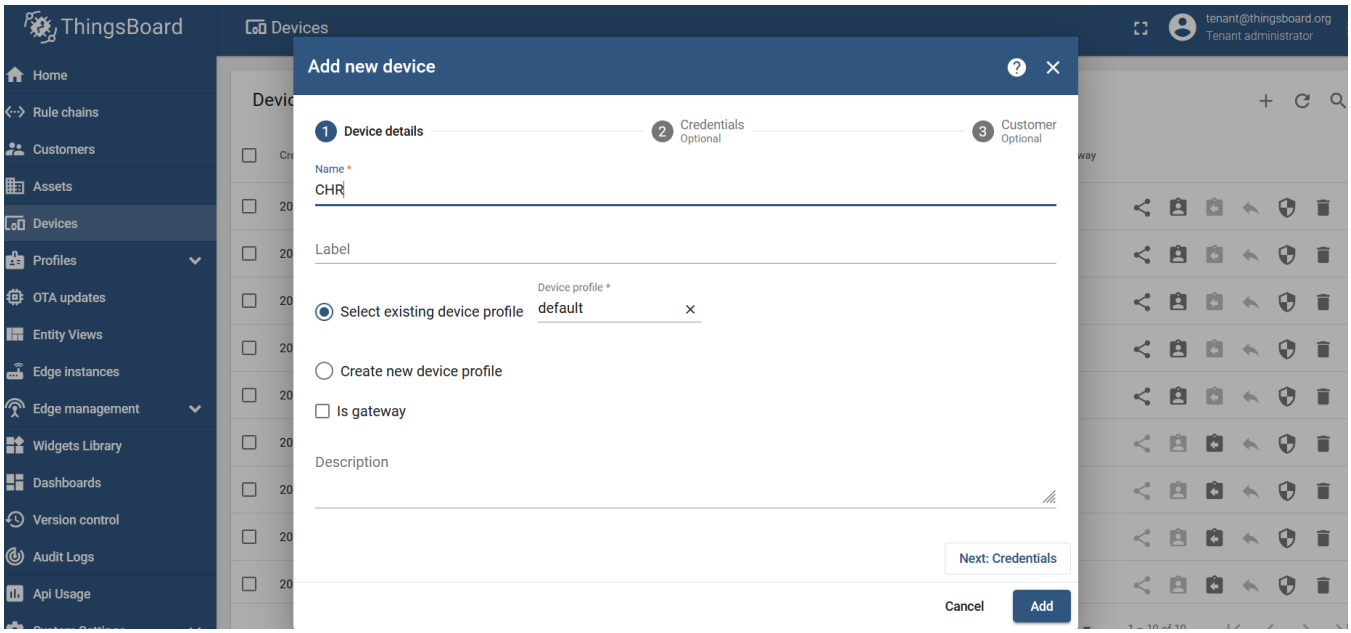
The login prompt should confirm that the server is running.

MQTT test

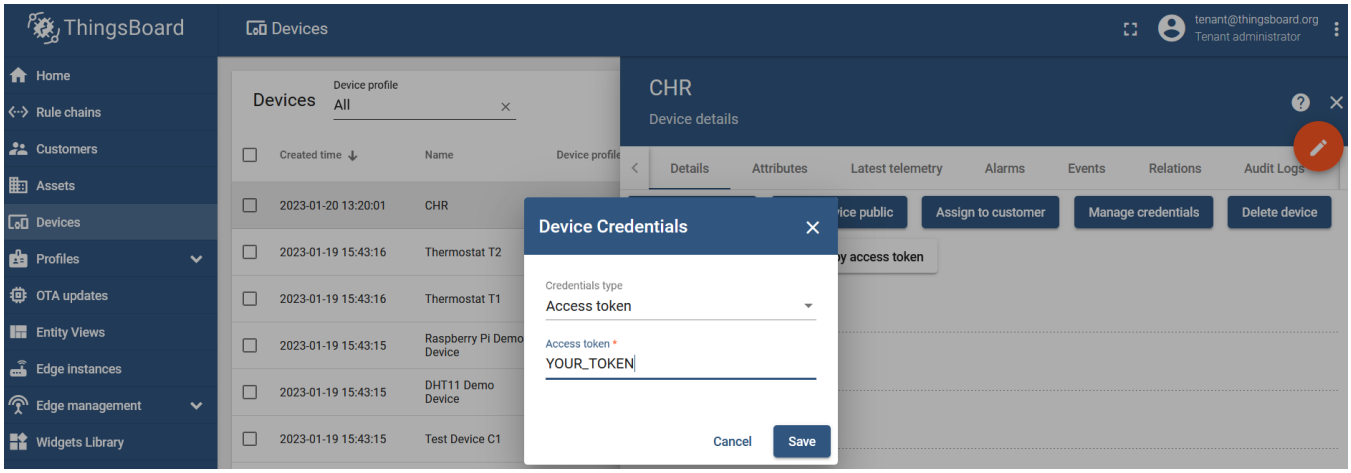
Log in with the **tenant** and create a new device. Go to the "**Devices**" menu, click on the "**+**" (Add Device) button and choose the "**Add new device**" option:



Name it, however, you like, and click on "**Add**":



Check your device access token by clicking on the device you've just created and selecting the "**Manage credentials**" setting (copy the access token generated or type in your own → "YOUR_TOKEN"):



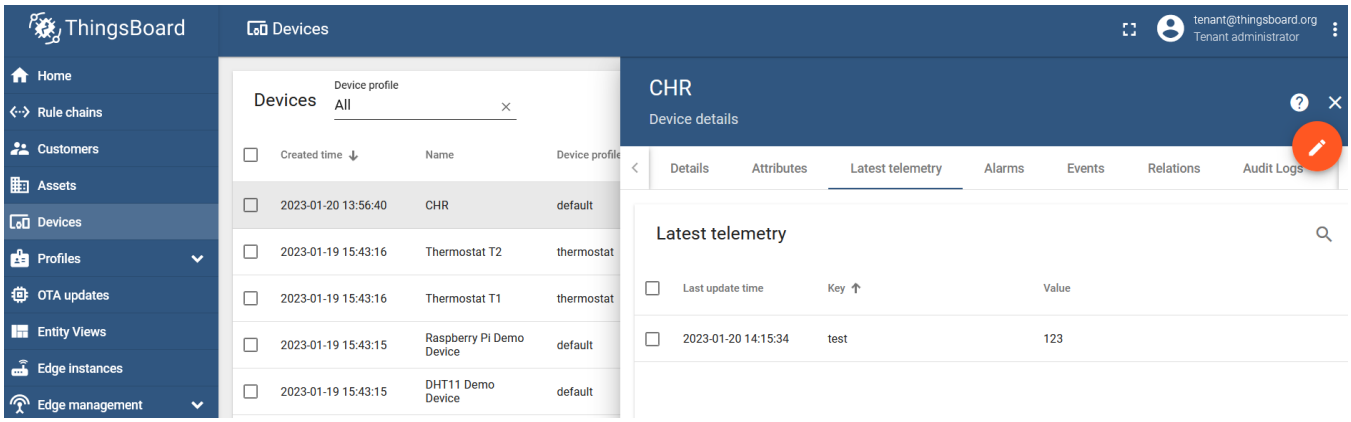
After these steps, go to the RouterOS settings (back to CHR settings) and create a new MQTT broker (**make sure that you have IoT package installed** beca use otherwise, you will not have this menu):

```
/iot/mqtt/brokers/add name=tb address=172.18.0.2 port=1883 username=YOUR_TOKEN
```

Publish a static test MQTT message in the JSON format:

```
/iot/mqtt/publish broker="tb" topic="v1/devices/me/telemetry" message="{\"test\":\"123\"}"
```

Confirm that the message was posted:



Enabling HTTPS and SSL MQTT

By default, HTTP and MQTT protocols are used. As mentioned previously in the "Networking" section, working with non-SSL HTTP and non-SSL MQTT is not very safe (unless they are used within heavily protected networks with a well-configured firewall/restricted access) and **we advise enabling HTTPS** and **SSL MQTT**.

Please check ThingsBoard documentation for more information → HTTP over SSL and MQTT over SSL guides.

First of all, there is no SSL without a certificate and one needs to be made (or purchased).

In short, this section will demonstrate how to generate self-signed certificates for HTTPS and SSL MQTT. Then, you will need to upload them to the correct folder within the ThingsBoard installation and alter the ThingsBoard configuration file accordingly.

In our guide, we will use RouterOS to generate both certificates (but you can also use OpenSSL or other tools you want).

Create certificates

Create a certificate for HTTPS:

```
/certificate add name=TBhttps common-name=172.18.0.2
/certificate sign TBhttps
```

Create a certificate for MQTT:

```
/certificate add name=TBmqtt common-name=172.18.0.2
/certificate sign TBmqtt
```

Confirm that they were added with the help of `/certificate/print` command:

```
[admin@MikroTik] > /certificate/print
Flags: K - PRIVATE-KEY; A - AUTHORITY; T - TRUSTED
Columns: NAME, COMMON-NAME, FINGERPRINT
#     NAME     COMMON-NAME  FINGERPRINT
0 KAT TBhttps  172.18.0.2   863f4547c74ce3ec70c3e82172502711517b52bbc055d18c24ba4aafec46152c
1 KAT TBmqtt   172.18.0.2   ebf3ff5d03ed4cc73546e058da9bc414cdaf24ce45da29b203348045fbbd21ae
```

Export the certificates using PKCS12 format and set up a password/passphrase for them:

```
/certificate/export-certificate file-name=keystore export-passphrase=thingsboard_cert_password type=pkcs12
numbers=0
/certificate/export-certificate file-name=mqttserver export-passphrase=thingsboard_mqttcert_password
type=pkcs12 numbers=1
```

Use your own `export-passphrase` and remember them.

The output from the command above will create certificate **keystore.p12** and **mqttserver.p12** files that you can download from the "File List" menu:

```
[admin@MikroTik] > /file/print
Columns: NAME, TYPE, SIZE, CREATION-TIME
 #  NAME                TYPE            SIZE       CREATION-TIME
 0  tb/mytb-data        container store             jan/19/2023 13:43:16
 1  container-log.0.txt .txt file       2240.5KiB  jan/27/2023 15:37:41
 2  skins               directory                  jan/18/2023 15:12:22
 3  tb/mytb-logs        container store             jan/27/2023 12:24:30
 4  pull                directory                  jan/19/2023 13:41:01
 5  pub                 directory                  jan/18/2023 16:15:29
 6  tb                  directory                  jan/23/2023 15:46:39
 7  tb/data             container store             jan/18/2023 16:50:08
 8  tb/logs             container store             jan/18/2023 16:50:08
 9  mqttserver.p12      .p12 file       2438       jan/27/2023 15:36:26
10  keystore.p12        .p12 file       2448       jan/27/2023 15:08:07
11  ThingsBoard         container store             jan/19/2023 13:40:50
```

Download both files from the router into any directory on your PC. For example, we've downloaded it into `C:\Users\Admin\Desktop\ThingsBoard` folder.

Download the ThingsBoard's configuration file

Open your command terminal ("CMD", as Administrator, for Windows users, or "Linux Shell or Command Terminal" for Linux users) and navigate it to the directory where the certificates are:

```
C:\Windows\System32>cd c:\Users\Admin\Desktop\ThingsBoard
C:\Users\Admin\Desktop\ThingsBoard>dir
Directory of C:\Users\Admin\Desktop\ThingsBoard

27.01.2023  15:36    <DIR>          .
27.01.2023  15:36    <DIR>          ..
27.01.2023  15:09             2 448 keystore.p12
27.01.2023  15:36             2 434 mqttserver.p12
               2 File(s)          4 882 bytes
               2 Dir(s)  51 380 154 368 bytes free
```

From this directory, you will need to connect to the router's IP via the SFTP (which allows you to file transfer using SSH protocol, so you need to make sure that SSH service is enabled beforehand):

```
c:\Users\Admin\Desktop\ThingsBoard>sftp admin@192.168.88.1
The authenticity of host '192.168.88.1 (192.168.88.1)' can't be established.
RSA key fingerprint is SHA256:/WmmZErqWL51SOlS4EaGvSQ0i4HPnSIHCEjnc8AmP2c.
Are you sure you want to continue connecting (yes/no/[fingerprint])?yes
admin@192.168.88.1's password:
Connected to 192.168.88.1.
sftp>
```

While the container is running, go to the ThingsBoard configuration file folder (use `dir` or `ls` command to see the content of the folder you are in and `cd` command to go to the folder of our choice). By default, it should be the folder with the "**thingsboard.yml**" configuration file in it. In our example, we could locate it under:

```
sftp> cd ThingsBoard\usr\share\thingsboard\conf

sftp> dir

banner.txt          i18n                logback.xml         templates           thingsboard.conf    thingsboard.
yml
```

Download the "**thingsboard.yml**" configuration using `get` command. This will download the default ThingsBoard configuration file to your machine (to the directory from where you initiated SFTP):

```
sftp> get thingsboard.yml
Fetching /ThingsBoard/usr/share/thingsboard/conf/thingsboard.yml to thingsboard.yml
/ThingsBoard/usr/share/thingsboard/conf/thingsboard.yml                              100%   67KB   2.0MB/s
00:00
sftp> quit

c:\Users\Admin\Desktop\ThingsBoard>dir
 Directory of c:\Users\Admin\Desktop\ThingsBoard

30.01.2023  10:59    <DIR>          .
30.01.2023  10:59    <DIR>          ..
27.01.2023  15:09             2 448 keystore.p12
27.01.2023  15:36             2 434 mqttserver.p12
30.01.2023  10:59            68 846 thingsboard.yml
               3 File(s)         73 728 bytes
               2 Dir(s)  50 901 626 880 bytes free
```

Alter the ThingsBoard's settings

Open "**thingsboard.yml**" via your preferred text editor (notepad or any other), and alter a few lines. You can backup this file and save it with a different name to have a copy of the default settings, in case, of misconfiguration.

HTTPS-related settings:

1. Enable SSL → Change "SSL_ENABLED:**false**" to "SSL_ENABLED:**true**";
2. Change credentials type → from "SSL_CREDENTIALS_TYPE:**PEM**" to "SSL_CREDENTIALS_TYPE:**KEYSTORE**";
3. Change the path → from "SSL_KEY_STORE:**classpath:keystore/keystore.p12**" to "SSL_KEY_STORE:**keystore.p12**" (optional);
4. Disable key alias setting → comment it → just put the "**#**" symbol in front of the **key_alias: "${SSL_KEY_ALIAS:tomcat}"** line;
5. Input your own certificate password that was used in RouterOS → from "SSL_KEY_STORE_PASSWORD:**thingsboard**" to "SSL_KEY_STORE_PASSWORD:**thingsboard_cert_password**" and from "SSL_KEY_PASSWORD:**thingsboard**" to "SSL_KEY_PASSWORD:**thingsboard_cert_password**".

```
  ssl:
    # Enable/disable SSL support
    enabled: "${SSL_ENABLED:true}"
    # Server SSL credentials
    credentials:
      # Server credentials type (PEM - pem certificate file; KEYSTORE - java keystore)
      type: "${SSL_CREDENTIALS_TYPE:KEYSTORE}"
      # Keystore server credentials
      keystore:
        # Type of the key store (JKS or PKCS12)
        type: "${SSL_KEY_STORE_TYPE:PKCS12}"
        # Path to the key store that holds the SSL certificate
        store_file: "${SSL_KEY_STORE:keystore.p12}"
        # Password used to access the key store
        store_password: "${SSL_KEY_STORE_PASSWORD:thingsboard_cert_password}"
        # Key alias
        #key_alias: "${SSL_KEY_ALIAS:tomcat}"
        # Password used to access the key
        key_password: "${SSL_KEY_PASSWORD:thingsboard_cert_password}"
```

MQTT-related settings:

1. Enable SSL → Change "MQTT_SSL_ENABLED:**false**" to "MQTT_SSL_ENABLED:**true**";
2. Change credentials type → from "MQTT_SSL_CREDENTIALS_TYPE:**PEM**" to "MQTT_SSL_CREDENTIALS_TYPE:**KEYSTORE**";
3. Change type of key → from "MQTT_SSL_KEY_STORE_TYPE:**JKS**" to "MQTT_SSL_KEY_STORE_TYPE:**PKCS12**";
4. Change the path (extension) → from "MQTT_SSL_KEY_STORE:mqttserver**.jks**" to "MQTT_SSL_KEY_STORE:mqttserver**.p12**";
5. Disable key alias setting → comment it → just put the "**#**" symbol in front of the **key_alias: "${MQTT_SSL_KEY_ALIAS:}"** line;
6. Input your own certificate password that was used in RouterOS → from "MQTT_SSL_KEY_STORE_PASSWORD:**server_ks_password**" to "MQTT_SSL_KEY_STORE_PASSWORD:**thingsboard_mqttcert_password**" and from "MQTT_SSL_KEY_PASSWORD:**server_key_password**" to "MQTT_SSL_KEY_PASSWORD:**thingsboard_mqttcert_password**".

```
    ssl:
      # Enable/disable SSL support
      enabled: "${MQTT_SSL_ENABLED:true}"
      # Server SSL credentials
      credentials:
        # Server credentials type (PEM - pem certificate file; KEYSTORE - java keystore)
        type: "${MQTT_SSL_CREDENTIALS_TYPE:KEYSTORE}"
        # Keystore server credentials
        keystore:
          # Type of the key store (JKS or PKCS12)
          type: "${MQTT_SSL_KEY_STORE_TYPE:PKCS12}"
          # Path to the key store that holds the SSL certificate
          store_file: "${MQTT_SSL_KEY_STORE:mqttserver.p12}"
          # Password used to access the key store
          store_password: "${MQTT_SSL_KEY_STORE_PASSWORD:thingsboard_mqttcert_password}"
          # Optional alias of the private key; If not set, the platform will load the first private key from
the keystore;
          #key_alias: "${MQTT_SSL_KEY_ALIAS:}"
          # Optional password to access the private key. If not set, the platform will attempt to load the
private keys that are not protected with the password;
          key_password: "${MQTT_SSL_KEY_PASSWORD:thingsboard_mqttcert_password}"
```

Leave the rest of the settings at default values. Do not delete/change lines that are not shown in the examples above unless you know what you are doing.

Apply the changes to the "**thingsboard.yml**" file (re-save it after editing).

Upload altered ThingsBoard configuration file

All that is left is to overwrite the current configuration file with an altered file and upload both certificates.

Once again, make sure your terminal is pointing to the right folder (where 3 files are located → both certificates and an altered "thingsboard.yml" file), and, from there, SFTP into the container's configuration file directory:

```
c:\Users\Admin\Desktop\ThingsBoard>dir
 Directory of c:\Users\Admin\Desktop\ThingsBoard

30.01.2023  10:59    <DIR>          .
30.01.2023  10:59    <DIR>          ..
27.01.2023  15:09             2 448 keystore.p12
27.01.2023  15:36             2 434 mqttserver.p12
30.01.2023  10:59            68 846 thingsboard.yml
               3 File(s)         73 728 bytes
               2 Dir(s)  50 901 626 880 bytes free
c:\Users\Admin\Desktop\ThingsBoard>sftp admin@192.168.88.1
admin@192.168.88.1's password:
Connected to 192.168.88.1.
sftp> cd ThingsBoard\usr\share\thingsboard\conf
sftp> dir
banner.txt          i18n                logback.xml         templates           thingsboard.conf    thingsboard.
yml
```

Upload these files with the help of `put` command:

```
sftp> put thingsboard.yml
Uploading thingsboard.yml to /ThingsBoard/usr/share/thingsboard/conf/thingsboard.yml
thingsboard.yml                                                                    100%   67KB   2.2MB/s
00:00
sftp> put keystore.p12
Uploading keystore.p12 to /ThingsBoard/usr/share/thingsboard/conf/keystore.p12
keystore.p12                                                                       100% 2448     1.2MB/s
00:00
sftp> put mqttserver.p12
Uploading mqttserver.p12 to /ThingsBoard/usr/share/thingsboard/conf/mqttserver.p12
mqttserver.p12                                                                     100% 2434   608.5KB/s
00:00
sftp> dir
banner.txt          i18n                keystore.p12        logback.xml         mqttserver.p12
templates
thingsboard.conf    thingsboard.yml
```

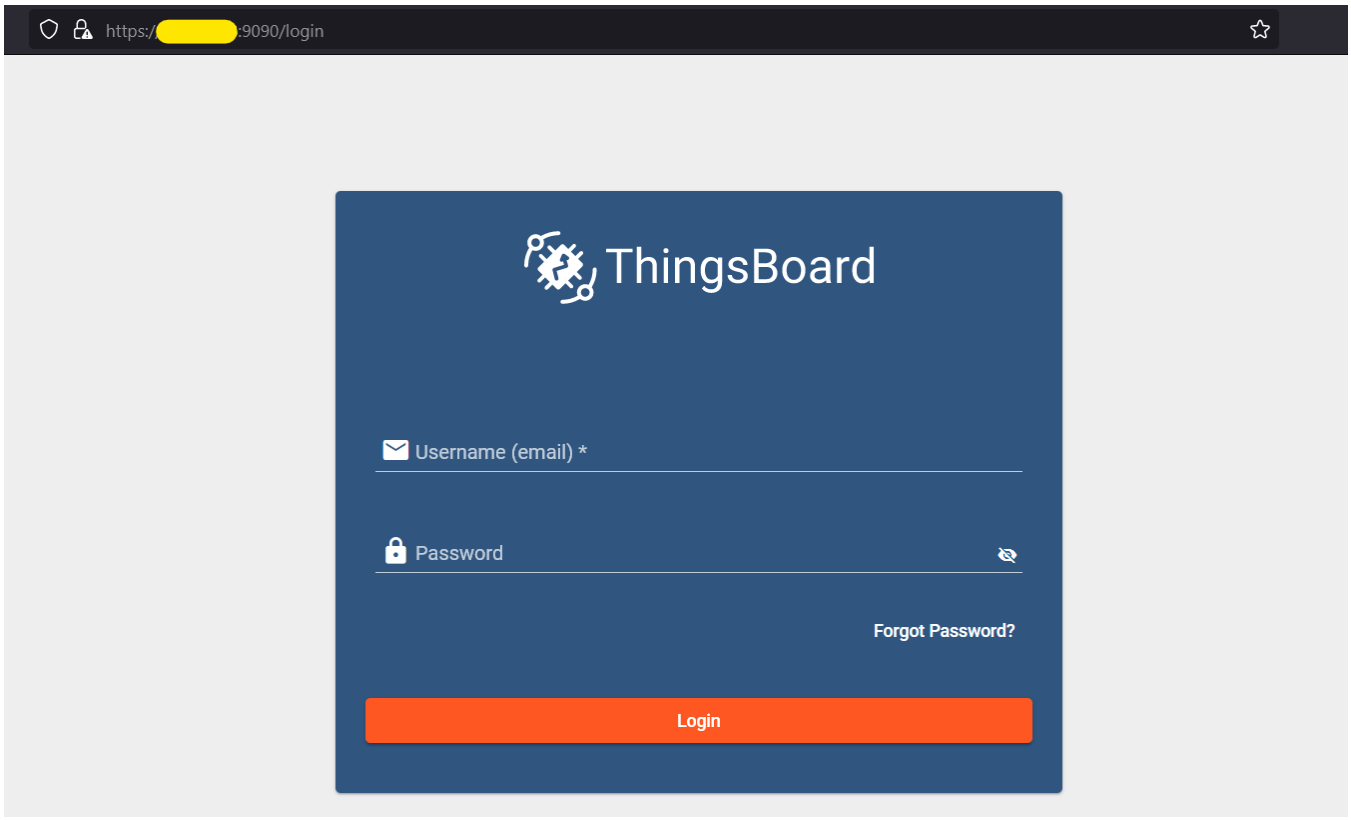
Restart the container:

```
[admin@MikroTik] > /container/stop 0
[admin@MikroTik] > /container/start 0
```

Make sure to wait for the container to stop (`status=stopped` should be shown after using `/container/print` command) before initiating it again.

Confirm HTTPS access

Now, you should be able to access https://your_IP:9090 (where the IP address is the address used in the DNAT rule):



Since we are using a self-signed certificate that was not issued by a trusted authority → an error indicating that the connection is not secure might appear but you can view the certificate through the browser (confirm it is the one), accept the risk, and continue.

Confirm SSL MQTT connection

> **Do not forget to alter the port forwarding rule** that is shown in the "Networking" section by changing **`dst-port`** and **`to-ports`** from 1883 (standard non-SSL MQTT port) **to 8883** (**SSL MQTT port**).

In this example, we will test a one-way SSL communication access token scenario.

Testing with the device that is running the container

> MQTT certificate should already be installed into the device's system (because it is the device that generated it).

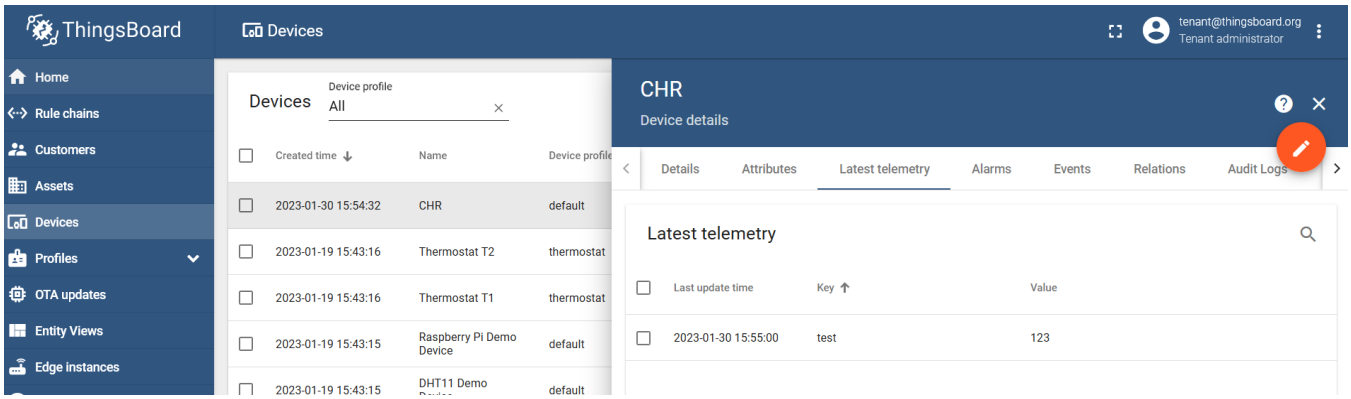
Add MQTT broker:

```
/iot/mqtt/brokers/add name=tbssl address=172.18.0.2 port=8883 username=YOUR_TOKEN ssl=yes
```

Publish a static test MQTT message in the JSON format:

```
/iot/mqtt/publish broker="tbssl" topic="v1/devices/me/telemetry" message="{\"test\":\"123\"}"
```

Confirm that it was received by the MQTT broker:



Testing with another device

When you have two RouterOS devices, one that is running the container (and, in our example, is the same device that generated the certificate) and the other one that you wish to test the MQTT connection from (let's say, an LTAP or any other RouterOS device with IoT package installed) → you will need to import the certificate to the second device.

Drag and drop the exported certificate (**mqttserver.p12**) into the device's "File List":

```
[admin@LTAP] > /file/print
Columns: NAME, TYPE, SIZE, CREATION-TIME
#  NAME            TYPE       SIZE  CREATION-TIME
0  mqttserver.p12  .p12 file  2438  jan/30/2023 13:28:11
1  flash           disk             jul/06/2021 14:51:53
2  flash/pub       directory        jul/06/2021 14:51:53
3  flash/skins     directory        jan/01/1970 02:00:07
[admin@LTAP] >
```

Import the certificate:

```
[admin@LTAP] > /certificate/import file-name=mqttserver.p12 passphrase=thingsboard_mqttcert_password
```

Add MQTT broker, where the address is the IP address "`dst-address`" that is used in the TCP 8883 port-forwarding rule on the ThingsBoard-container router:

```
/iot/mqtt/brokers/add name=tbssl address=192.168.88.1 port=8883 username=YOUR_TOKEN ssl=yes
```

Publish a static test MQTT message in the JSON format:

```
/iot/mqtt/publish broker="tbssl" topic="v1/devices/me/telemetry" message="{\"test\":\"123\"}"
```

And confirm that the broker received it → under the "Latest Telemetry" section on the ThingsBoard.

Media (DLNA)

DLNA is a set of protocols that enables networked devices to share digital media, including videos, photos, and music. Central to the operation of DLNA is the UPnP (Universal Plug and Play) architecture, which facilitates the discovery and control of network devices.

DLNA and UPnP work in tandem to provide a seamless media sharing experience. UPnP supports device discovery and control on the network through protocols like the Simple Service Discovery Protocol (SSDP) and others such as SOAP (Simple Object Access Protocol) for control messages and XML for device and service descriptions. In the context of DLNA, UPnP serves as the foundation, allowing various devices like TVs, computers, and mobile devices to connect and share media content efficiently.

In RouterOS, enable the media server and share movies or music with your household media devices, such as TVs or player apps in your PC, such as the popular VLC.

> Media (DLNA) is not supported on SMIPS devices

Server settings

Property	Description
allowed-hostname	To restrict access to specific hostnames.
allowed-ip	To limit access to specified IP addresses
friendly-name	The name that will be displayed for the DLNA server on the network.
interface	Specifies the network interface that the DLNA server will use
path	The file path where the media content is stored and will be served from.

Configuration examples

To create/enable DLNA server

```
/ip media add friendly-name=Mikrotik interface=bridge1 path=usb1
```

To create/enable multiple DLNA servers with limitations. Usage example - Limit KIDs TV acces only to media/kids

```
/ip media add friendly-name=adults interface=bridge1 path=usb1/adults allowed-hostname=ADULTS_TV
/ip media add friendly-name=kids interface=bridge1 path=usb1/kids allowed-hostname=KIDS_TV
```

ROSE-storage

Summary

Packages required: `rose-storage`

ROSE - package adds additional enterprise data center functionality to RouterOS - for supporting disk monitoring, improved formatting, RAIDs, rsync, iSCSI ,NVMe over TCP, NFS and improved SMB. This functionality currently is supported on **arm, arm64, x86** and **tile** platforms.

```
/disk
```

General inteface properties

Property	Description
crypted-backend	Drive or device used together with type=crypted to make "dm_crypt" encrypted storage
encryption-key	
iscsi-address	
iscsi-export	
iscsi-iqn	
iscsi-port	
nfs-address	
nfs-export	
nfs-share	
nvme-tcp-address	

nvme-tcp-export	
nvme-tcp-host-name	
nvme-tcp-name	
nvme-tcp-password	
nvme-tcp-port	
nvme-tcp-server-allow-host-name	
nvme-tcp-server-password	
nvme-tcp-server-port	
raid-chunk-size	
raid-device-count	
raid-master	
raid-max-component-size	
raid-member-failed	
raid-role	
raid-type	
slot	
smb-address	
smb-encryption	
smb-export	
smb-password	
smb-share	
smb-user	
tmpfs-max-size	
type	

Partitions

GPT partitions are supported starting from RouterOS 7.8beta3

to add 500MB partition:

```
/disk
disk add type=partition parent=sata1 partition-size=500M
```

if next partition will be added it will automatically allocated in available space from start of drive.

partition can be also added with offset:

```
/disk
add type=partition parent=sata1 partition-size=500M partition-offset=10G
```

on partition overlap, RouterOS will return error.

RAID

RAID (Redundant Array of Independent Disks) technology allows storing data on multiple drives - improving data transfer performance, data protection or both by combining them into logical units.

RAID levels

RouterOS supports software RAID levels 0,1,4,5,6,linear and nested RAID.

RAID 0

All data is written evenly over all disks in this RAID, this configuration does not provide any fault tolerance but provides best performance.

RAID 1

Same data is written in all drives (data is mirrored), this configuration provides best fault tolerance, but performance wise write speeds will be equal to slowest disk used in array.

RAID 4

Block-level data is striped to a dedicated disk where parity bits are stored. Performance will be limited to a parity writing speed.

RAID 5

Block-level data is striped evenly over the available disks. Can be recovered from 1 disk failure.

RAID 6

Block-level data is striped evenly over the available disks. Can be recovered from 2 disk failures.

Linear

Data is appended over multiple disks combining them into single large disk. Provides no redundancy and is limited to single disk read/write speed.

Nested RAID

Combination of multiple RAID configurations into other RAID. For example RAID 10 (RAID 1+0) combines disk mirroring (RAID 1) and disk striping (RAID 0)

RAID configuration

In this example we will create RAID 6 with 10 disks
Disks has to be in same size or have to be set in same size partitions or use raid-max-component-size parameter to limit larger volume size to match other elements.

In theory for RAID performance optimization - correct stride and stripe-width should be used. These are dependent on the RAID "raid-chunk-size", filesystem block size, and the number of disks.

```
stride=raid-chunk-size/block_size
```

```
stripe_width=disks*stride
```

RouterOS does this automatically when formatting local RAID device.

Create RAID device:

```
/disk add type=raid raid-type=6 raid-device-count=10 slot=raid1
```

add disks into this raid:

```
/disk set pcie1-nvme1 raid-master=raid1 raid-role=0
/disk set pcie1-nvme2 raid-master=raid1 raid-role=1
/disk set pcie1-nvme3 raid-master=raid1 raid-role=2
/disk set pcie1-nvme4 raid-master=raid1 raid-role=3
/disk set pcie1-nvme5 raid-master=raid1 raid-role=4
/disk set pcie1-nvme6 raid-master=raid1 raid-role=5
/disk set pcie1-nvme7 raid-master=raid1 raid-role=6
/disk set pcie1-nvme8 raid-master=raid1 raid-role=7
/disk set pcie1-nvme9 raid-master=raid1 raid-role=8
/disk set pcie1-nvme10 raid-master=raid1 raid-role=9
```

where pcie1-nvme* your local disk slot name

> Setting "raid-role" manually is optional, but highly recommended. If device has never been in raids before then superblock is empty, and raid-role will be assumed automatically, if not there may be error regarding using same RAID role if that has been already taken.

Raid will now sync:

```
/disk print detail
...
20 bM      type=raid slot="raid1" slot-default="" parent=none device="md0" uuid="3b4d4ec9-e7413ae8-37e7e397-
9cd9152e"
           fs=ext4 model="RAID5 1-parity-disk" size=8 641 770 946 560 free=8 572 463 624 192 raid-type=5
           raid-device-count=10 raid-max-component-size=none raid-chunk-size=1M raid-master=none
           raid-state="clean, resync =  1.8% (17498368/937692160) finish=45.2min speed=339148K/sec"
           nvme-tcp-export=no iscsi-export=no nfs-export=no smb-export=no
```

iSCSI

iSCSI allows accessing storage over an IP-based network. On initiator iSCSI device will appear as block device. RouterOS supports both target and initiator modes.

Target (Host) configuration:

```
/disk
set pcie1-nvme1 iscsi-export=yes
```

Initiator (client):

```
/disk
add type=iscsi iscsi-address=192.168.1.1 iscsi-iqn=pcie1-nvme1
```

iscsi-iqn needs to match slot name on target device, iscsi-address is target address.

NFS

NFS allows sharing local directories over network. RouterOS currently supports NFS v4 only mode.

Host configuration:

```
/disk
set pcie1-nvme1 nfs-export=yes
```

Initiator (client):
RouterOS

```
/disk
add type=nfs nfs-address=192.168.1.1
```

Linux:

```
mkdir /mnt/files
mount -t nfs 192.168.1.1:/ /mnt/files
```

SMB

SMB is popular file sharing protocol. ROSE package currently supports SMB2.1 SMB3.0, SMB3.1.1 dialects (SMB1 is not supported due to security vulnerabilities)

RouterOS also supports older SMB without ROSE package - SMB with legacy protocol support.

Host configuration:

```
/disk
set pcie1-nvme1 smb-export=yes
```

Initiator (client):

```
/disk
add type=smb smb-address=192.168.1.1 smb-share=pcie1-nvme1
```

smb-share needs to match slot name on target device, smb-address is target address.

NVMe over TCP

nvme-tcp allows accessing storage over network as NVMe block device on initiator side. On target side this device can be hdd/ssd/nvme or even raid array.

Target (Host) configuration:

```
/disk
set pcie1-nvme2 nvme-tcp-export=yes nvme-tcp-port=4420
```

Initiator (client):
RouterOS

```
/disk
add type=nvme-tcp nvme-tcp-address=192.168.1.1 nvme-tcp-name=pcie1-nvme1
```

nvme-tcp-name needs to match slot name on target device.

Linux:

load kernel module

```
modprobe nvme_tcp
```

discover available nvme-tcp targets:

```
nvme discover -t tcp -a 192.168.1.1 -s 4420
```
```
Discovery Log Number of Records 1, Generation counter 2
=====Discovery Log Entry 0======
trtype:  tcp
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified, sq flow control disable supported
portid:  4420
trsvcid: 4420
subnqn:  pcie1-nvme1
traddr:  10.155.166.7
sectype: none
```

subnqn should match slot name and will be used as -n parameter:

```
nvme connect -t tcp -a 192.168.1.1 -s 4420 -n pcie1-nvme1
```

Block device now should be available:

```
ls /dev/nvme*
```
```
/dev/nvme0  /dev/nvme0n1  /dev/nvme-fabrics
```

To disconnect:

```
nvme disconnect -d /dev/nvme0
```

where /dev/nvme0 previously mounted device, or disconnect all:

Error rendering macro 'code': Invalid value specified for parameter '[Ljava.lang.Object;@37515560'

```
nvme disconnect-all
```

RAMdisk

RAMdisk - allows using part of RAM as attached device (block device). If compared to tmpfs - this allows using RAM as part of raid, or any other configuration where device instead of folder is required.

```
/disk
disk add type=ramdisk ramdisk-size=500M
```

RAMdisk will be cleared on reboot or power loss

Data encryption

Currently RouterOS supports SED (Self-encrypting drives) and dm_crypt drive encryption.

Self-Encrypting Drives

For using SED - drives has to be Opal-compliant. Please consult drive manufacturers documentation to find out if particular drive supports this feature before buying drives.
RouterOS adds **o (supported inactive)** or **O (supported active)** flags for supported drives:

```
/disk print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING; o - TCG-OPAL-SELF-ENCRYPTION-SUPPORTED
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE, FS, RAID-MASTER
#     SLOT   MODEL                 SERIAL           INTERFACE                  SIZE             FREE  FS
RAID
0 BMo sata1  Samsung SSD 860 2.5in  S3Z9NX0N414510L  SATA 6.0 Gbps  1 000 204 886 016  983 351 111 680  ext4
none
1 BMo sata2  Samsung SSD 860        S5GENG0N307602J  SATA 6.0 Gbps  1 000 204 886 016  983 351 128 064  ext4
none
2 BMO sata3  Samsung SSD 860        S5GENG0N307604H  SATA 6.0 Gbps  1 000 204 886 016  983 351 128 064  ext4
none
3 BMO sata4  Samsung SSD 860 2.5in  S4CSNX0N838150B  SATA 6.0 Gbps  1 000 204 886 016  983 351 128 064  ext4
none
```

To set TCG-OPAL-SELF-ENCRYPTION:

```
/disk
disk set sata1 self-encryption-password=securepassword
```

to unset:

```
/disk
disk unset sata1 self-encryption-password
```

or

```
/disk
disk set sata1 !self-encryption-password
```

Block device encryption

...

File Sync

ROSE package also includes tool for file upload/download and sync.
To sync (push) local folder contents to other RouterOS device:

```
/file sync add local-path=pcie1-nvme1/myfolder/ remote-addrs=192.168.1.1 mode=upload user=admin password=""
remote-path=test/
```

where user/password - other device username and password. For interconnection Winbox port needs to be open.

or to pull files use:

```
/file sync add local-path=pcie1-nvme1/myfolder/ remote-addrs=192.168.1.1 mode=download user=admin password=""
remote-path=test/
```

Folder now will be in sync, all changes to files will be synced between device.

One folder should not have download and upload type to same target, to avoid undefined behavior.

SMB

Summary

Sub-menu: `/ip smb`
Packages required: `system`

SMB server provides file sharing access to configured folders of the router.

⚠ RouterOS only supports SMB2.1 SMB3.0, SMB3.1.1. SMB1 is not supported due to security vulnerabilities.

SMB is not supported on SMIPS devices

Server settings

Property	Description
comment (*string*; Default: **MikrotikSMB**)	Set comment for the server
domain (*string*; Default: **MSHOME**)	Name of Windows Workgroup
enabled (*yes* | *no* | *auto* Default: **auto**)	The default value is 'auto.' This means that the SMB server will automatically be enabled when the first non-disabled SMB share is configured under '/ip smb share'
interface (*string*; Default: **all**)	List of interfaces on which SMB service will be running. all - SMB will be available on all interfaces.

⚠ Starting from version 7.14, the 'allow-guest' option has been replaced by a default guest user located in 'ip/smb/users'. This default guest user can now be disabled or enabled in this section.

Share settings

Sub-menu: `/ip smb shares`

Allows configuring share names and directories that will be accessible by SMB.

If the directory provided in the configuration does not exist it will be created automatically.

Property	Description
comment (*string*; Default: **default share**)	Set a comment for the share
disabled (*yes* | *no*; Default: **no**)	If disabled, the share will not be accessible.
valid-users (list of *strings*; | Default:)	Specifies which users are allowed to access the Samba share. If it is left empty, all users will be able to access the share, once user or users are defined here, only they will be able to access the share
invalid-users (list of *strings*; | Default:)	Used to specify users who are explicitly denied access to the Samba share.
require-encryption (*yes* | *no*; Default: ***no***)	Enforces the use of encryption for all connections to a particular Samba share
name (*string*; Default:)	Name of the SMB share

directory (*string*; Default:)	Directory on router assigned to SMB share. If left empty value of the *name* argument will be used from the root folder.

User setup

Sub-menu: `/ip smb user`

Set up users that can access SMB shares of the router.

Property	Description
comment (*string*; Default:)	Set a description for the user
disabled (*yes* | *no*; Default: **no**)	Defines whether the user is enabled or disabled
name (*string*; Default:)	Login name of the SMB service user
password (*string*; Default:)	Password for SMB user to connect to SMB service
read-only (*yes* | *no*; Default: **yes**)	Sets if the user has only read-only rights when accessing shares or full access rights.

Example

To make RouterOS folder available through SMB service follow these steps:

- create user:

```
/ip/smb/users/add read-only=no name=mtuser password=mtpasswd
```

- add shared folder:

```
/ip/smb/shares/add directory=backup name=backup
```

- enable SMB service:

```
#this step is optional, as the default is "enabled=auto"
/ip/smb/set enabled=yes
```

Now check for results:

- Check general service settings:

```
/ip/smb/print
      enabled: yes
       domain: MSHOME
      comment: MikrotikSMB
   interfaces: all
```

- SMB user settings:

```
/ip smb/users/print
Flags: X - DISABLED; * - DEFAULT; r - READ-ONLY
Columns: NAME, PASSWORD
#     NAME    PASSWORD
0 X*r guest
1     mtuser  mtpasswd
```

- And finally SMB shares settings:

```
/ip/smb/shares/print
Flags: X - DISABLED; * - DEFAULT
Columns: NAME, DIRECTORY, REQUIRE-ENCRYPTION
#    NAME    DIRECTORY  REQUIRE-ENCRYPTION
;;; default share
0 X* pub     /pub       no
1    backup  backup     no
```

Now, additional configuration changes can be done, like disabling the default user and share, etc.

UPS

Summary

Sub-menu: `/system ups`
Standards: APC Smart Protocol

The UPS monitor feature works with APC UPS units that support “smart” signalling over serial RS232 or USB connection. The UPS monitor service is not included in the default set of packages so it needs to be downloaded and installed manually with ups.npk package. This feature enables the network administrator to monitor the UPS and set the router to ‘gracefully’ handle any power outage with no corruption or damage to the router. The basic purpose of this feature is to ensure that the router will come back online after an extended power failure. To do this, the router will monitor the UPS and set itself to hibernate mode when the utility power is down and the UPS battery has less than 10% of its battery power left. The router will then continue to monitor the UPS (while in hibernate mode) and then restart itself when the utility power returns. If the UPS battery is drained and the router loses all power, the router will power back to full operation when the ‘utility’ power returns.

The UPS monitor feature on the MikroTik RouterOS supports

- hibernate and safe reboot on power and battery failure
- UPS battery test and run time calibration test
- monitoring of all "smart" mode status information supported by UPS
- logging of power changes

Connecting the UPS unit

The serial APC UPS (BackUPS Pro or SmartUPS) requires a special serial cable (unless connected with USB). If no cable came with the UPS, a cable may be ordered from APC or one can be made "in-house". Use the following diagram:

Router Side (DB9f)	Signal	Direction	UPS Side (DB9m)
2	Receive	IN	2
3	Send	OUT	1
5	Ground		4
7	CTS	IN	6

If using a RouterBOARD device, make sure to set your "RouterBOOT setup key" to *Delete* instead of the default *Any key*. This is to avoid accidental opening of the setup menu if the UPS unit sends some data to the serial port during RouterBOARD startup. This can be done in the RouterBOOT options during boot time or via the RouterBoard Settings in Winbox :

```
Select key which will enter setup on boot:
 * 1 - any key
   2 - <Delete> key only
your choice:
```

General Properties

Property	Description
alarm-setting (*delayed* | *immediate* | *low-battery* | *none*; Default: **immediate**)	UPS sound alarm setting: - delayed - alarm is delayed to the on-battery event - immediate - alarm immediately after the on-battery event - low-battery - alarm only when the battery is low - none - do not alarm
check-capabilities (*yes* | *no*; Default: **yes**)	Whether to check UPS capabilities before reading information. Disabling it can fix compatibility issues with some UPS models. (Applies to RouterOS version 6, implemented since v6.17)

min-runtime (*time*; Default: **never**)	Minimal run time remaining. After a 'utility' failure, the router will monitor the runtime-left value. When the value reaches the min-runtime value, the router will go to hibernate mode. • never - the router will go to hibernate mode when the "battery low" signal is sent indicating that the battery power is below 10% • 0s - the router will continue to work as long as the battery is supplying sufficient voltage
offline-time (*time*; Default: **0s**)	How long to work on batteries. The router waits that amount of time and then goes into hibernate mode until the UPS reports that the 'utility' power is back • 0s - the router will go into hibernate mode according to the min-runtime setting. In this case, the router will wait until the UPS reports that the battery power is below 10%
port (*string*; Default:)	Communication port of the router.

Read-only properties:

Property	Description
load (*percent*)	The UPS's output load as a percentage of full rated load in Watts. The typical accuracy of this measurement is ±3% of the maximum of 105%
manufacture-date (*string*)	UPS's date of manufacture in the format "mm/dd/yy" (month, day, year).
model (*string*)	Less than 32 ASCII character string consisting of the UPS model name (the words on the front of the UPS itself)
nominal-battery-voltage (*integer*)	UPS's nominal battery voltage rating (this is not the UPS's actual battery voltage)
offline-after (*time*)	When will the router go offline
serial (*string*)	A string of at least 8 characters directly representing the UPS's serial number as set at the factory. Newer SmartUPS models have 12-character serial numbers
version (*string*)	UPS version, consists of three fields: SKU number, firmware revision, country code. The country code may be one of the following: • I - 220/230/240 Vac • D - 115/120 Vac • A - 100 Vac • M - 208 Vac • J - 200 Vac

Note: In order to enable UPS monitor, the serial port should be available.

Example

To enable the UPS monitor for port serial1:

```
[admin@MikroTik] system ups> add port=serial1 disabled=no
[admin@MikroTik] system ups> print
Flags: X - disabled, I - invalid
 0    name="ups" port=serial1 offline-time=5m min-runtime=5m
      alarm-setting=immediate model="SMART-UPS 1000" version="60.11.I"
      serial="QS0030311640" manufacture-date="07/18/00"
      nominal-battery-voltage=24V
[admin@MikroTik] system ups>
```

Runtime Calibration

Command: `/system ups rtc <id>`

The rtc command causes the UPS to start a run time calibration until less than 25% of full battery capacity is reached. This command calibrates the returned run time value.

> **Note:** The test begins only if the battery capacity is 100%.

Monitoring

Command: `/system ups monitor <id>`

Property	Description
battery-charge ()	the UPS's remaining battery capacity as a percent of the fully charged condition
battery-voltage ()	the UPS's present battery voltage. The typical accuracy of this measurement is ±5% of the maximum value (depending on the UPS's nominal battery voltage)
frequency ()	when operating on-line, the UPS's internal operating frequency is synchronized to the line within variations of 3 Hz of the nominal 50 or 60 Hz. The typical accuracy of this measurement is ±1% of the full scale value of 63 Hz
line-voltage ()	the in-line utility power voltage
load ()	the UPS's output load as a percentage of full rated load in Watts. The typical accuracy of this measurement is ±3% of the maximum of 105%
low-battery (*yes* | *no*)	only shown when the UPS reports this status
on-battery (*yes* | *no*)	Whether UPS battery is supplying power
on-line (*yes* | *no*)	whether power is being provided by the external utility (power company)
output-voltage ()	the UPS's output voltage
overloaded-output (*yes* | *no*)	only shown when the UPS reports this status
replace-battery (*yes* | *no*)	only shown when the UPS reports this status
runtime-calibration-running (*yes* | *no*)	only shown when the UPS reports this status
runtime-left (*time*)	the UPS's estimated remaining run time in minutes. You can query the UPS when it is operating in the on-line, bypass, or on-battery modes of operation. The UPS's remaining run time reply is based on available battery capacity and output load
smart-boost-mode (*yes* | *no*)	only shown when the UPS reports this status
smart-ssdd-mode ()	only shown when the UPS reports this status
transfer-cause (*string*)	the reason for the most recent transfer to on-battery operation (only shown when the unit is on-battery)

Example

When running on utility power:

```
[admin@MikroTik] system ups> monitor 0
          on-line: yes
       on-battery: no
      RTC-running: no
     runtime-left: 20m
   battery-charge: 100%
  battery-voltage: 27V
     line-voltage: 226V
   output-voltage: 226V
             load: 45%
      temperature: 39C
        frequency: 50Hz
  replace-battery: no
      smart-boost: no
       smart-trim: no
         overload: no
      low-battery: no

[admin@MikroTik] system ups>
```

When running on battery:

```
[admin@MikroTik] system ups> monitor 0
          on-line: no
       on-battery: yes
   transfer-cause: "Line voltage notch or spike"
      RTC-running: no
     runtime-left: 19m
    offline-after: 4m46s
   battery-charge: 94%
  battery-voltage: 24V
     line-voltage: 0V
   output-voltage: 228V
             load: 42%
      temperature: 39C
        frequency: 50Hz
  replace-battery: no
      smart-boost: no
       smart-trim: no
         overload: no
      low-battery: no

      [admin@MikroTik] system ups>
```

Wake on LAN

Introduction

Sub-menu: `/tool wol`

The Wake on LAN tool can send a UDP Magic Packet to the Broadcast address with a selected MAC address embedded in it.

If the target device supports Wake on LAN (a target computer has specific hardware and software requirements for the Wake on LAN feature to work), it should wake up from sleep or shutdown state.

Currently, secure WoL is not supported.

Property Description

Property	Description
interface (*string; Default:*)	Interface through which the Magic Packet will be sent
mac (*MAC*; Default:)	MAC address of a target computer

Application Example

The command requires a MAC address parameter and interface.

> ⚠ If an interface is not specified, A Magic Packet will be sent as an IP broadcast from a default gateway interface.

```
/tool wol mac=FE:4B:71:05:EA:8B interface=ether1
```