# Manual:Interface/Bridge

< Manual:Interface

**Warning:** This manual is moved to https://help.mikrotik.com/docs/display/ROS/Bridge

**Applies to RouterOS: v3, v4+**

## Contents



# Summary

**Sub-menu:** `/interface bridge`
**Standards:** IEEE 802.1D (https://en.wikipedia.org/wiki/IEEE_802.1D) , IEEE 802.1Q (https://en.wikipedia.org/wiki/IEEE_802.1Q)

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 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 (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 network from functioning normally, as they would lead to avalanche-like packet multiplication. Each bridge runs an algorithm which calculates how the loop can be prevented. STP and RSTP 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 to 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 network topology. (R)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

**Sub-menu:** `/interface bridge`

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 "port-number", and it can change after a reboot. To avoid unwanted MAC address changes, it is recommended to disable "auto-mac", and to manually specify MAC by using "admin-mac".

### Properties

| 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 effect when `dhcp-snooping` is set to `yes`. |
| **admin-mac** (*MAC address*; Default: **none**) | Static MAC address of the bridge. This property only has 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* \| *proxy-arp* \| *reply-only*; Default: **enabled**) | Address Resolution Protocol setting<br>▪ `disabled` - the interface will not use ARP<br>▪ `enabled` - the interface will use ARP<br>▪ `proxy-arp` - the interface will use the ARP proxy feature<br>▪ `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**) | 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. |
| **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 reboot, 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 effect when `vlan-filtering` is set to `yes`. |
| **fast-forward** (*yes* \| *no*; Default: **yes**) | Special and faster case of FastPath 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**) | Time which is spent during the initialization phase of the bridge interface (i.e., after router startup or enabling the interface) in 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 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 general membership queries will be generated. This property only has effect when `igmp-snooping` is set to `yes`. |
| **ingress-filtering** (*yes* \| *no*; Default: **no**) | 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 effect when `vlan-filtering` is set to `yes`. |
| **l2mtu** (*read-only*; Default: ) | L2MTU indicates the maximum size of the frame without 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**) | If a port has `fast-leave` set to `no` and a bridge port receives a IGMP Leave message, then a IGMP Snooping enabled bridge will send a IGMP query to make sure that no devices has subscribed to a certain multicast stream on a bridge port. If a IGMP Snooping enabled bridge does not receive a IGMP membership report after amount of `last-member-interval`, then the bridge considers that no one has subscribed to a certain multicast stream and can stop forwarding it. This property only has effect when `igmp-snooping` is set to `yes`. |
| **last-member-query-count** (*integer: 0..4294967295*; Default: **2**) | How many times should `last-member-interval` pass until a IGMP Snooping bridge will stop forwarding a certain multicast stream. This property only has effect when `igmp-snooping` is set to `yes`. |
| **max-hops** (*integer: 6..40*; Default: **20**) | Bridge count which BPDU can pass in a MSTP enabled network in the same region before BPDU is being ignored. This property only has effect when `protocol-mode` is set to `mstp`. |
| **max-message-age** (*time: 6s..40s*; Default: **00:00:20**) | 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 effect when `protocol-mode` is set to `stp` or `rstp`. |
| **membership-interval** (*time*; Default: **4m20s**) | Amount of time after an entry in the Multicast Database (MDB) is removed if a IGMP membership report is not received on a certain port. This property only has effect when `igmp-snooping` is set to `yes`. |
| **mld-version** (*1* \| *2*; Default: **1**) | Selects the MLD version. Version 2 adds support for source-specific multicast. This property only has effect when RouterOS IPv6 package is enabled and `igmp-snooping` 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. |
| **multicast-querier** (*yes* \| *no*; Default: **no**) | Multicast querier generates IGMP general membership queries to which all IGMP capable devices respond with an IGMP membership report, usually a PIM (multicast) router or IGMP proxy generates these queries. When RouterOS IPv6 package is enabled, the bridge will also generate MLD general membership queries. By using this property you can make an IGMP 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 and IGMP Snooping will not function properly. Only untagged IGMP/MLD general membership queries are generated. This property only has an effect when `igmp-snooping` is set to yes. Additionally, the `igmp-snooping` should be disabled/enabled after changing `multicast-querier` property. |

| | |
|---|---|
| **multicast-router** (*disabled* \| *permanent* \| *temporary-query*; Default: **temporary-query**) | Changes the state of a bridge itself if IGMP membership reports are going to be forwarded to it. This property can be used to forward IGMP membership reports to the bridge for statistics or to analyse them.<br>▪ disabled - IGMP membership reports are not forwarded to the bridge itself regardless what is connected to it.<br>▪ permanent - IGMP membership reports are forwarded through this the bridge itself regardless what is connected to it.<br>▪ temporary-query - automatically detect multicast routers and IGMP Snooping enabled bridges. This property only has effect when igmp-snooping is set to yes. |
| **name** (*text*; Default: **bridgeN**) | Name of the bridge interface |
| **priority** (*integer: 0..65535 decimal format or 0x0000-0xffff hex format*; Default: **32768 / 0x8000**) | Bridge priority, used by STP to determine root bridge, used by MSTP to determine CIST and IST regional root bridge. This property has no effect when protocol-mode is set to none. |
| **protocol-mode** (*none* \| *rstp* \| *stp* \| *mstp*; Default: **rstp**) | Select Spanning tree protocol (STP) or Rapid spanning tree protocol (RSTP) to ensure a loop-free topology for any bridged LAN. RSTP provides for 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 **01:80:C2:00:00:0X** range, this can be done by setting the protocol-mode to none. |
| **pvid** (*integer: 1..4094*; Default: **1**) | 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 effect when vlan-filtering is set to yes. |
| **querier-interval** (*time*; Default: **4m15s**) | Used to change the interval how often a bridge checks if it is the active multicast querier. This property only has effect when igmp-snooping and multicast-querier is set to yes. |
| **query-interval** (*time*; Default: **2m5s**) | Used to change the interval how often IGMP general membership queries are sent out. This property only has effect when igmp-snooping and multicast-querier is set to yes. |
| **query-response-interval** (*time*; Default: **10s**) | Interval in which a IGMP capable device must reply to a IGMP query with a IGMP membership report. This property only has effect when igmp-snooping and multicast-querier is set to yes. |
| **region-name** (*text*; Default: ) | MSTP region name. This property only has effect when protocol-mode is set to mstp. |
| **region-revision** (*integer: 0..65535*; Default: **0**) | MSTP configuration revision number. This property only has effect when protocol-mode is set to mstp. |
| **startup-query-count** (*integer: 0..4294967295*; Default: **2**) | Specifies how many times must startup-query-interval pass until the bridge starts sending out IGMP general membership queries periodically. This property only has effect when igmp-snooping and multicast-querier is set to yes. |
| **startup-query-interval** (*time*; Default: **31s250ms**) | Used to change the amount of time after a bridge starts sending out IGMP general membership queries after the bridge is enabled. This property only has 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 transmission rate. |
| **vlan-filtering** (*yes* \| *no*; Default: **no**) | Globally enables or disables VLAN functionality for bridge. |

**Warning:** 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 all the protocols:

```
[admin@MikroTik] /interface bridge> add
[admin@MikroTik] /interface bridge> print
Flags: X - disabled, R - running
 0  R name="bridge1" mtu=1500 l2mtu=65535 arp=enabled
      mac-address=00:00:00:00:00:00 protocol-mode=none priority=0x8000
      auto-mac=yes admin-mac=00:00:00:00:00:00 max-message-age=20s
      forward-delay=15s transmit-hold-count=6 ageing-time=5m
[admin@MikroTik] /interface bridge>
```

# 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 much benefits, but for larger networks properly configured STP is very crucial, leaving STP related values to default may result in completely unreachable network in case of a 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.

**Warning:** 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 does 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 backwards 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 a STP enabled network, more detailed case studies can be found in the Spanning Tree Protocol section. In RouterOS the protocol-mode property controls the used STP variant.

**Note:** 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 a single or multiple ports. Below you can find some examples for different use cases.

**Warning:** 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 behaviour.

- Don't send out BPDUs from a certain port

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
/interface bridge filter
add action=drop chain=output dst-mac-address=01:80:C2:00:00:00/FF:FF:FF:FF:FF:FF out-interface=ether1
```

In this example BPDUs will not be sent out through **ether1**. In case the bridge is the root bridge, then loop detection will not work on this port. If another bridge is connected to **ether1**, then the other bridge will not receive any BPDUs and therefore might become as a second root bridge. You might want to consider blocking received BPDUs as well.

**Note:** You can use Interface Lists to specify multiple interfaces.

- Dropping received BPDUs on a certain port can be done on some switch chips using ACL rules, but the Bridge Filter Input rules cannot do it if bridge has STP/RSTP/MSTP enabled because then received BPDUs have special processing in the bridge.

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
```

Or 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. This will prevent other bridges on that port becoming a root bridge.

**Warning:** 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 designated bridge. If you have blocked BPDUs only on one side, then a port will flap continuously.

- Don't allow BPDUs on a port

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 bpdu-guard=yes
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether3
```

In this example if **ether1** receives a BPDU, it will block the port and will require you to manually re-enable it.

# Bridge Settings

**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 ( Packet Flow). 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 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 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 FastPath globally. |
| **bridge-fast-path-active** (*yes* \| *no*; Default: *)* | Shows whether a bridge FastPath is active globally, FastPatch status per bridge interface is not displayed. |
| **bridge-fast-path-packets** (*integer*; Default: *)* | Shows packet count forwarded by Bridge FastPath. |
| **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. |

**Note:** In case you want to assign Simple Queues (Simple QoS) 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

**Sub-menu:** `/interface bridge port`

Port submenu is used to enslave interfaces in a particular bridge interface.

| 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 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`.<br>▪ `no` - non-edge port, will participate in learning and listening states in STP.<br>▪ `no-discover` - non-edge port with enabled discovery, will participate in learning and listening states in STP, a port can become edge port if no BPDU is received.<br>▪ `yes` - edge port without discovery, will transit directly to forwarding state.<br>▪ `yes-discover` - edge port with enabled discovery, will transit directly to forwarding state.<br>▪ `auto` - same as `no-discover`, but will additionally detect if bridge port is a Wireless interface with disabled bridge-mode, such interface will be automatically set as an edge port without discovery. |
| **`external-fdb`** (*auto* \| *no* \| *yes*; Default: **auto**) | Whether to use wireless registration table to speed up bridge host learning. If there are no Wireless interfaces in a bridge, then setting `external-fdb` to `yes` will disable MAC learning and the bridge will act as a hub (disables hardware offloading). Replaced with `learn` parameter in RouterOS v6.42 |
| **`fast-leave`** (*yes* \| *no*; Default: **no**) | Enables IGMP Fast leave feature on the port. Bridge will stop forwarding traffic to a bridge port whenever a IGMP Leave message is received for appropriate multicast stream. This property only has 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 effect when `vlan-filtering` is set to `yes`. |
| **`ingress-filtering`** (*yes* \| *no*; Default: **no**) | 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`. |
| **`learn`** (*auto* \| *no* \| *yes*; Default: **auto**) | Changes MAC learning behaviour on a bridge port<br>▪ `yes` - enables MAC learning<br>▪ `no` - disables MAC learning<br>▪ `auto` - detects if bridge port is a Wireless interface and uses 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**) | Changes the state of a bridge port whether IGMP membership reports are going to be forwarded to this port. By default IGMP membership reports (most importantly IGMP Join messages) are only forwarded to ports that have a multicast router or a IGMP Snooping enabled bridge connected to. Without at least one port marked as a `multicast-router` IPTV might not work properly, it can be either detected automatically or forced manually.<br>▪ `disabled` - IGMP membership reports are not forwarded through this port regardless what is connected to it.<br>▪ `permanent` - IGMP membership reports are forwarded through this port regardless what is connected to it.<br>▪ `temporary-query` - automatically detect multicast routers and IGMP Snooping enabled bridges.<br>You can improve security by forcing ports that have IPTV boxes connected to never become ports marked as `multicast-router`. This property only has effect when `igmp-snooping` is set to `yes`. |
| **`horizon`** (*integer 0..429496729*; Default: **none**) | Use split horizon bridging to prevent bridging loops. Set the same value for 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. |
| **`internal-path-cost`** (*integer: 0..4294967295*; Default: **10**) | Path cost to the interface for MSTI0 inside a region. This property only has effect when `protocol-mode` is set to `mstp`. |
| **`interface`** (*name*; Default: **none**) | Name of the interface. |

| | |
|---|---|
| **path-cost** (*integer: 0..4294967295*; Default: **10**) | Path cost to the interface, used by STP to determine the "best" path, used by MSTP to determine "best" path between regions. This property has no effect when protocol-mode is set to none. |
| **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 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. And 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 effect when vlan-filtering is set to yes. |
| **restricted-role** (*yes* \| *no*; Default: **no**) | Enable the restricted role on a port, used by STP to forbid a port becoming a root port. This property only has effect when protocol-mode is set to mstp. |
| **restricted-tcn** (*yes* \| *no*; Default: **no**) | Disable topology change notification (TCN) sending on a port, used by STP to forbid network topology changes to propagate. This property only has effect when protocol-mode is set to mstp. |
| **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 to forward DHCP packets towards DHCP server through this port. Mainly used to limit unauthorized servers to provide malicious information for users. This property only has effect when dhcp-snooping is set to yes. |
| **unknown-multicast-flood** (*yes* \| *no*; Default: **yes**) | When enabled, bridge floods unknown multicast traffic to all bridge egress ports. When disabled, drops unknown multicast traffic on egress ports. Multicast addresses that are in /interface bridge mdb are considered as learned multicasts and therefore will not be flooded to all ports. Without IGMP Snooping all multicast traffic will be dropped on egress ports. Has effect only on an egress port. This option does not limit traffic flood to the CPU. Note that local multicast addresses (224.0.0.0/24) are not flooded when unknown-multicast-flood is disabled, as a result some protocols that rely on local multicast addresses might not work properly, such protocols are RIPv2m OSPF, mDNS, VRRP and others. Some protocols do send a IGMP join request and therefore are compatible with IGMP Snooping, some OSPF implementations are compatible with RFC1584, RouterOS OSPF implementation is not compatible with IGMP Snooping. This property should only be used when igmp-snooping is set to yes. |
| **unknown-unicast-flood** (*yes* \| *no*; Default: **yes**) | When enabled, bridge floods unknown unicast traffic to all bridge egress ports. When disabled, drops unknown unicast traffic on egress ports. If a MAC address is not learned in /interface bridge host, then the traffic is considered as unknown unicast traffic and will be flooded to all ports. MAC address is learnt as soon as a packet on a bridge port is received, then 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 learnt. Has effect only on an egress port. This option does not limit traffic flood to the CPU. |

## Example

To group **ether1** and **ether2** in the already created **bridge1** bridge

```
[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
 #    INTERFACE              BRIDGE              PRIORITY PATH-COST  HORIZON
 0    ether1                 bridge1             0x80     10         none
 1    ether2                 bridge1             0x80     10         none
[admin@MikroTik] /interface bridge port>
```

# 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, you can read more about interface lists at the Interface List section. Below is an example how to add interface list to a bridge:

```
/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 a interface list added to a bridge will show up as dynamic ports:

```
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #      INTERFACE                                  BRIDGE
 0      LAN1                                       bridge1
 1  D   ether1                                     bridge1
 2  D   ether2                                     bridge1
 3      LAN2                                       bridge1
 4  D   ether3                                     bridge1
 5  D   ether4                                     bridge1
```

It is also possible to sort the order of lists in which they appear in the /interface bridge port menu. This can be done using the move command. Below is an example 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
 0      LAN2                                       bridge1
 1  D   ether3                                     bridge1
 2  D   ether4                                     bridge1
 3      LAN1                                       bridge1
 4  D   ether1                                     bridge1
 5  D   ether2                                     bridge1
```

**Note:** 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 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.

# Hosts Table

MAC addresses that have been learned on a bridge interface can be viewed in the `/interface bridge host` menu. Below is a table of parameters and flags that can be viewed.

**Sub-menu:** `/interface bridge host`

| Property | Description |
|---|---|
| **age** (*read-only: time*) | The time since the last packet was received from the host. Appears only for dynamic, non-external and non-local host entries |
| **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     AGE
 0   D E D4:CA:6D:E1:B5:7E      ether2         bridge1
 1   DL  E4:8D:8C:73:70:37      bridge1        bridge1
 2   D   D4:CA:6D:E1:B5:7F      ether3         bridge2    27s
 3   DL  E4:8D:8C:73:70:38      bridge2        bridge2
```

## Static entries

Since RouterOS v6.42 it is possible to add a static MAC address entry into the hosts table. This can be used to forward a certain type of traffic through a specific port. Another use case for static host entries is for protecting the device resources by disabling the dynamic learning and rely 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 hosts 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 to. |
| **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 hosts 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
```

# Bridge Monitoring

**Sub-menu:** `/interface bridge monitor`

Used to monitor the current status of a bridge.

| Property | Description |
|---|---|
| **current-mac-address** (*MAC address*) | Current MAC address of the bridge |
| **designated-port-count** (*integer*) | Number of designated bridge ports |
| **port-count** (*integer*) | Number of the bridge ports |
| **root-bridge** (*yes* \| *no*) | Shows whether 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 |

### Example

To monitor a bridge:

```
[admin@MikroTik] /interface bridge> monitor bridge1
                  state: enabled
    current-mac-address: 00:0C:42:52:2E:CE
            root-bridge: yes
         root-bridge-id: 0x8000.00:00:00:00:00:00
         root-path-cost: 0
              root-port: none
             port-count: 2
  designated-port-count: 0

[admin@MikroTik] /interface bridge>
```

# Bridge Port Monitoring

**Sub-menu:** `/interface bridge port monitor`

Statistics of an interface that belongs to a bridge.

| Property | Description |
|---|---|
| **edge-port** (*yes* \| *no*) | Whether port is an edge port or not. |
| **edge-port-discovery** (*yes* \| *no*) | Whether port is set to automatically detect edge ports. |
| **external-fdb** (*yes* \| *no*) | Whether registration table is used instead of forwarding data base. |
| **forwarding** (*yes* \| *no*) | Shows if the port is not blocked by (R/M)STP. |
| **hw-offload-group** (*switchX*) | Switch chip used by the port. |
| **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. |
| **port-number** (*integer 1..4095*) | port-number will be assigned in the order that ports got added to the bridge, but this is only true until reboot. After reboot internal 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 role of the port:<br>▪ `Disabled port` - not strictly part of STP, a network administrator can manually disable a port<br>▪ `Root port` - a forwarding port that is the best port from Nonroot-bridge to Rootbridge<br>▪ `Alternative port` - an alternate path to the root bridge. This path is different than using the root port<br>▪ `Designated port` - a forwarding port for every LAN segment<br>▪ `Backup port` - a backup/redundant path to a segment where another bridge port already connects. |
| **sending-rstp** (*yes* \| *no*) | Whether the port is sending RSTP or MSTP BPDU types. A port will transit to STP type when RSTP/MSTP enabled port receives a STP BPDU |
| **status** (*in-bridge* \| *inactive*) | Port status:<br>▪ `in-bridge` - port is enabled.<br>▪ `inactive` - port is disabled. |

## Example

To monitor a bridge port:

```
[admin@MikroTik] > /interface bridge port monitor 0
              status: in-bridge
         port-number: 1
                role: designated-port
           edge-port: no
  edge-port-discovery: yes
  point-to-point-port: no
         external-fdb: no
         sending-rstp: no
             learning: yes
           forwarding: yes

[admin@MikroTik] /interface bridge port>
```

# Bridge Hardware Offloading

Since RouterOS v6.41 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. More details about the outdated `master-port` property can be found in the Master-port page.

**Note:** 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 (-):

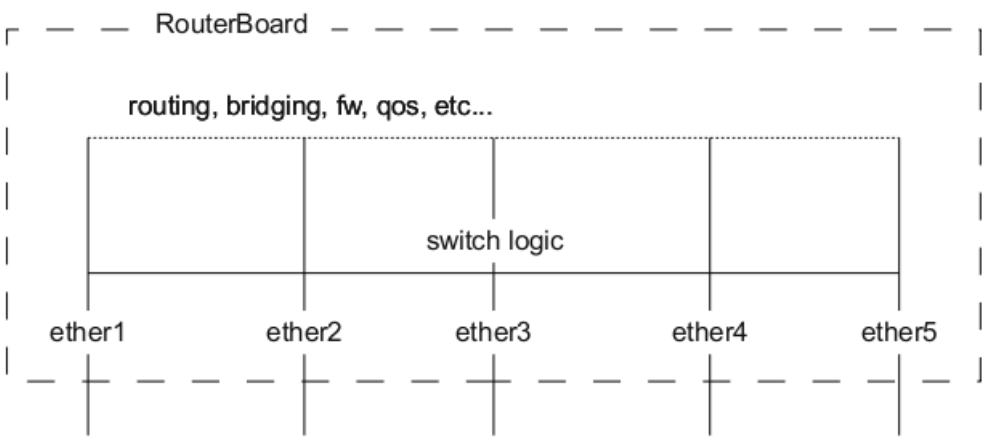
| RouterBoard/[Switch Chip] Model | Features in Switch menu | Bridge STP/RSTP | Bridge MSTP | Bridge IGMP Snooping | Bridge DHCP Snooping | Bridge VLAN Filtering | Bonding |
|---|---|---|---|---|---|---|---|
| CRS3xx series | + | + | + | + | + | + | + |
| CRS1xx/CRS2xx series | + | + | - | + [2] | + [1] | - | - |
| [QCA8337] | + | + | - | - | + [2] | - | - |
| [Atheros8327] | + | + | - | - | + [2] | - | - |
| [Atheros8227] | + | + | - | - | - | - | - |
| [Atheros8316] | + | + | - | - | + [2] | - | - |
| [Atheros7240] | + | + | - | - | - | - | - |
| [MT7621] | + | - | - | - | - | - | - |
| [RTL8367] | + | - | - | - | - | - | - |
| [ICPlus175D] | + | - | - | - | - | - | - |

**NOTES:**

1. 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. Feature will not work properly in VLAN switching setups.

**Note:** 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 processes packets using it's 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 to 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. To allow the CPU process a packet you need to forward the packet to the CPU and not allow the switch chip to forward the packet through a switch port directly, this is usually called passing a packet to the switch CPU port (or the bridge CPU port in bridge VLAN filtering scenario).

By passing a packet to the switch CPU port you are prohibiting the switch chip to forward the packet directly, 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 series switches
- non-CRS series switches

**Warning:** Certain bridge and Ethernet port properties are directly related to switch chip settings, changing such properties can trigger a **switch chip reset**, that will temporarily disable all Ethernet ports that are on the switch chip for the settings to have an effect, this must be taken into account whenever changing properties on production environments. Such properties are DHCP Snooping, IGMP Snooping, VLAN filtering, L2MTU, Flow Control and others (exact settings that can trigger a switch chip reset depends on the device's model).

## Example

Port switching with bridge configuration and enabled hardware offloading since RouterOS v6.41:

```
/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 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
```

**Note:** 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, for more details check the Master-port page.

# Bridge VLAN Filtering

**Note:** Currently only CRS3xx series devices 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 Filtering since RouterOS v6.41 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 compatibilty issues compared to configuration when tunnel-like 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`, 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, bridge itself is also an interface therefore it has Port VLAN ID (pvid).

**Sub-menu:** `/interface bridge vlan`

Bridge VLAN table represents per-VLAN port mapping with an egress VLAN tag action. `tagged` ports send out frames with a learned VLAN ID tag. `untagged` ports remove VLAN tag before sending out frames if the learned VLAN ID matches the port `pvid`.

| 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 VLAN ID range as well as comma separated values. E.g. `vlan-ids=100-115,120,122,128-130`. |

**Warning:** 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.

**Sub-menu:** `/interface bridge host`

Bridge Host table allows monitoring learned MAC addresses and when `vlan-filtering` is enabled shows learned VLAN ID as well.

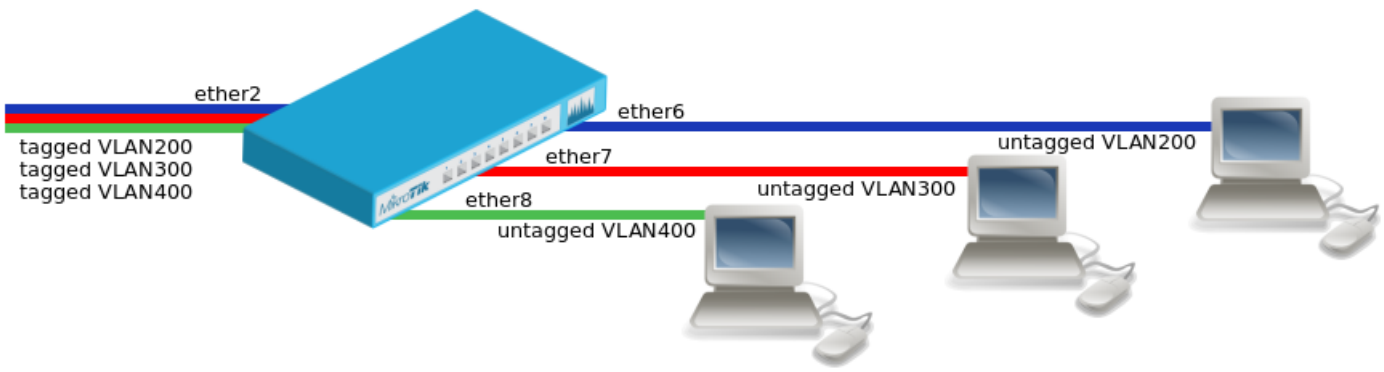
```
[admin@MikroTik] > interface bridge host print where !local
Flags: L - local, E - external-fdb
  BRIDGE                    VID MAC-ADDRESS       ON-INTERFACE                AGE
  bridge1                   200 D4:CA:6D:77:2E:F0 ether3                      7s
  bridge1                   200 E4:8D:8C:1B:05:F0 ether2                      2s
  bridge1                   300 D4:CA:6D:74:65:9D ether4                      3s
  bridge1                   300 E4:8D:8C:1B:05:F0 ether2                      2s
  bridge1                   400 4C:5E:0C:4B:89:5C ether5                      0s
  bridge1                   400 E4:8D:8C:1B:05:F0 ether2                      0s
[admin@MikroTik] >
```

**Note:** 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 CPU from those ports, 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 at the Management port section.

**Warning:** 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.

## VLAN Example #1 (Trunk and Access Ports)

**Note:** 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.




Trunk and Access Ports

- Create a bridge with disabled `vlan-filtering` to avoid losing access to the device before VLANs are completely configured.

```
/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.

```
/interface bridge port
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether6 pvid=200
add bridge=bridge1 interface=ether7 pvid=300
add bridge=bridge1 interface=ether8 pvid=400
```

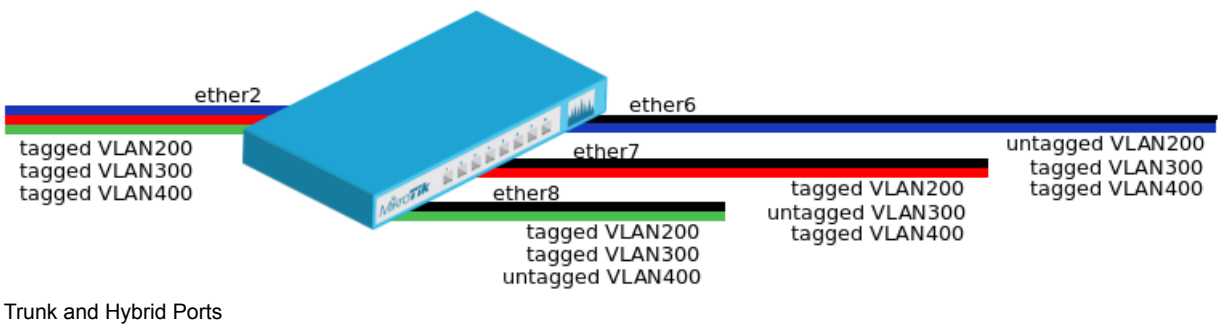
- Add Bridge VLAN entries and specify tagged and untagged ports in them.

```
/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
```

- In the end, when VLAN configuration is complete, enable Bridge VLAN Filtering.

```
/interface bridge set bridge1 vlan-filtering=yes
```

## VLAN Example #2 (Trunk and Hybrid Ports)




Trunk and Hybrid Ports

- Create a bridge with disabled `vlan-filtering` to avoid losing access to the router before VLANs are completely configured.

```
/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.

```
/interface bridge port
add bridge=bridge1 interface=ether2
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 and untagged ports in them. In this example egress VLAN tagging is done on ether6,ether7,ether8 ports too, making them into hybrid ports.

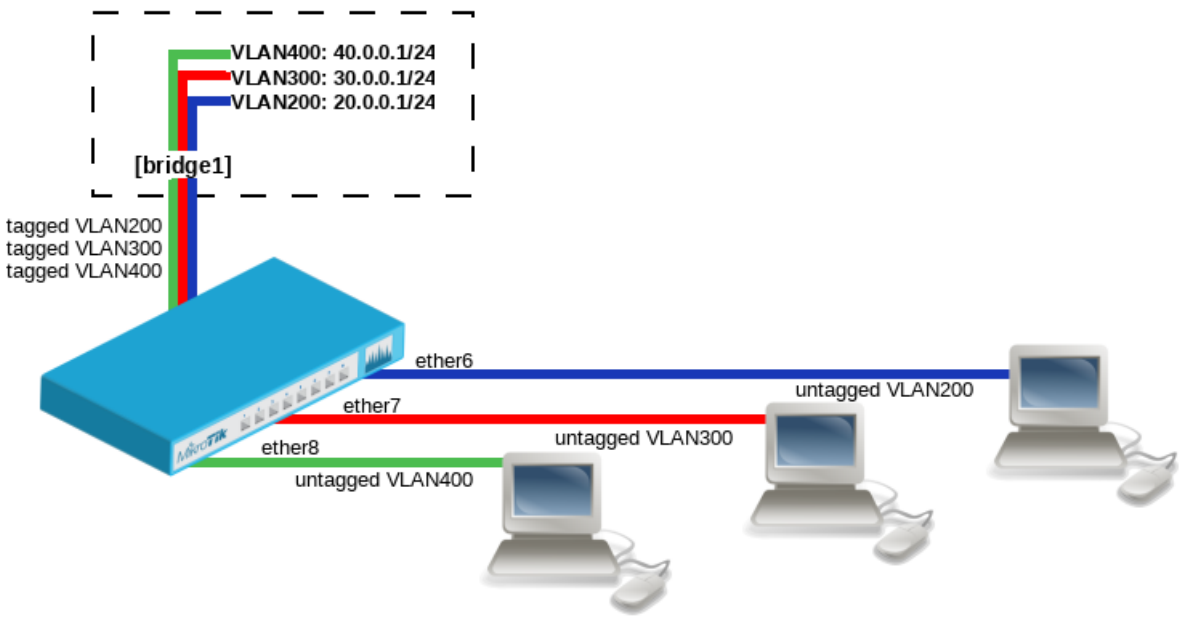
```
/interface bridge vlan
add bridge=bridge1 tagged=ether2,ether7,ether8 untagged=ether6 vlan-ids=200
add bridge=bridge1 tagged=ether2,ether6,ether8 untagged=ether7 vlan-ids=300
add bridge=bridge1 tagged=ether2,ether6,ether7 untagged=ether8 vlan-ids=400
```

- In the end, when VLAN configuration is complete, enable Bridge VLAN Filtering.

```
/interface bridge set bridge1 vlan-filtering=yes
```

**Warning:** You don't have to add access ports as untagged ports, they will be added dynamically as untagged port with the VLAN ID that is specified in `PVID`, you can specify just the trunk port as 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 untagged port for the ports that have the same `PVID`. You can circumvent this behaviour 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 #3 (InterVLAN Routing by Bridge)




InterVLAN Routing by Bridge

Create a bridge with disabled `vlan-filtering` to avoid losing access to the router before VLANs are completely configured:

```
/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:

```
/interface bridge port
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 and untagged ports in them. In this example **bridge1** interface is the VLAN trunk that will send traffic further to do InterVLAN routing:

```
/interface bridge vlan
add bridge=bridge1 tagged=bridge1 untagged=ether6 vlan-ids=200
add bridge=bridge1 tagged=bridge1 untagged=ether7 vlan-ids=300
add bridge=bridge1 tagged=bridge1 untagged=ether8 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
```

# Management access configuration

There are multiple ways to setup 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
```

- 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
```

- In case VLAN filtering is used and access from trunk and/or access ports with tagged traffic is desired

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
```

For example, if you want to allow access to the router/switch from access ports **ether3**, **ether4** and from trunk port **sfp-sfpplus1**, then you must add this entry to the VLAN table:

```
/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
```

- In case VLAN filtering is used and access from trunk and/or access ports with untagged traffic is desired

To allow untagged traffic to access the router/switch, start by creating an IP address on the bridge interface.

```
/ip address
add address=192.168.88.1/24 interface=bridge1
```

It is required to add VLAN 1 to ports from which you want to allow the access to the router/switch, for example, to allow access from access ports **ether3**, **ether4** add this entry to the VLAN table:

```
/interface bridge vlan
add bridge=bridge1 untagged=ether3,ether4 vlan-ids=1
```

Make sure that PVID on the bridge interface matches the PVID value on these ports:

```
/interface bridge set bridge1 pvid=1
/interface bridge port set ether3,ether4 pvid=1
```

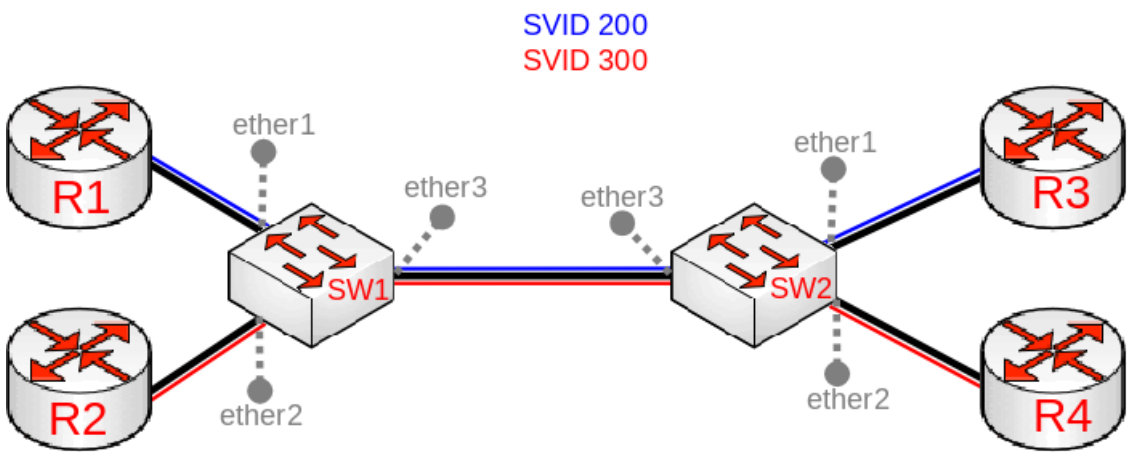
After that you can enable VLAN filtering:

```
/interface bridge set bridge1 vlan-filtering=yes
```

**Note:** If connection to the router/switch through an IP address is not required, then steps adding this IP address can be skipped since connection to the router/switch through Layer2 protocols (e.g. MAC-telnet) will be working either way.

# VLAN Tunneling (Q-in-Q)

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 principals 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**:



Provider bridge topology

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 a SVID and only allow these VLANs on certain ports. Start by enabling `802.1ad` VLAN protocol 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 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
```

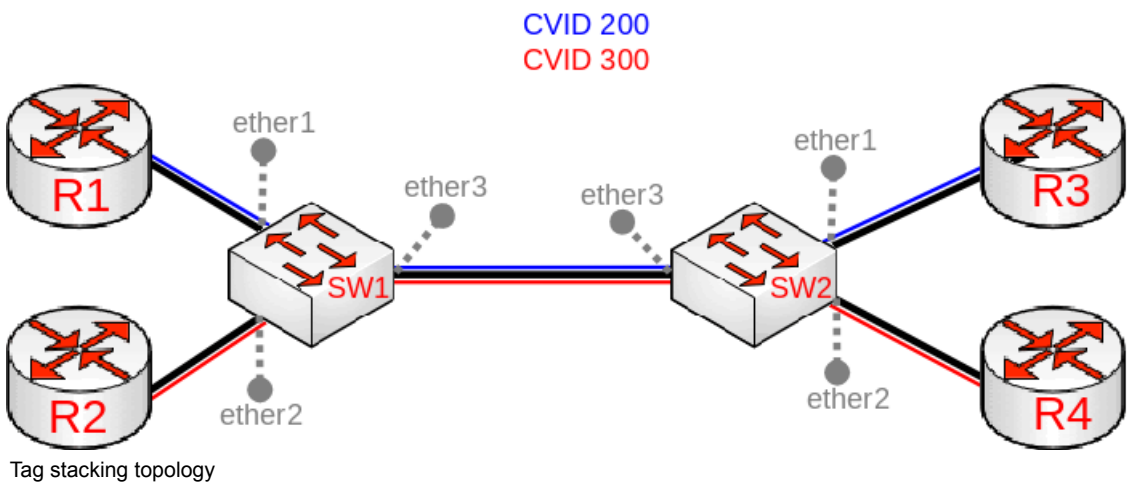
**Warning:** By enabling `vlan-filtering` you will be filtering out traffic destined to the CPU, before enabling VLAN filtering you should make sure that you set up a Management port. The difference between using different EtherTypes is that you must use a Service VLAN interface. Service VLAN interfaces can be created as regular VLAN interface, but the `use-service-tag` parameter toggles if the interface will use Service VLAN tag.

**Note:** Currently only CRS3xx series switches are capable of hardware offloading VLAN filtering based on SVID (Service VLAN ID) tag when `ether-type` is set to 0x88a8.

**Warning:** When `ether-type=0x8100`, then the bridge checks the outer VLAN tag 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 802.1ad protocol is used.

## 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 Provider bridge setup, to achieve the same setup but with multiple CVID tags (CVID stacking) we can use the same topology:



Tag stacking 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 bridge VLAN table is configured, you can enable bridge VLAN filtering, which is required in order for the `PVID` parameter have any effect, use these commands on **SW1** and **SW2**

```
/interface bridge set bridge1 vlan-filtering=yes
```

**Warning:** 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.

# Fast Forward

Fast Forward allows to forward 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 slaves
- `horizon` for both ports is set to `none`

**Note:** 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 trough just one interface.

**Warning:** 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 print
             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: 1279812
    bridge-fast-forward-bytes: 655263744
```

**Note:** If packets are processed by Fast Path, then Fast Forward is not active. Packet count can be used as an indicator whether Fast Forward is active or not.

Since RouterOS 6.44beta28 it is possible to monitor Fast Forward status, for example:

```
[admin@MikroTik] > /interface bridge monitor bridge1
                  state: enabled
    current-mac-address: D4:CA:6D:E1:B5:82
            root-bridge: yes
         root-bridge-id: 0x8000.00:00:00:00:00:00
         root-path-cost: 0
              root-port: none
             port-count: 2
  designated-port-count: 0
           fast-forward: yes
```

**Warning:** 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 Snooping

IGMP Snooping which controls multicast streams and prevents multicast flooding is implemented in RouterOS starting from version 6.41.
It's settings are placed in bridge menu and it works independently in every bridge interface.
Software driven implementation works on all devices with RouterOS but CRS1xx/2xx/3xx series switches also support IGMP Snooping with hardware offloading.

**Sub-menu:** `/interface bridge` `/interface bridge mdb`

- Enabling IGMP Snooping on Bridge.

```
/interface bridge set bridge1 igmp-snooping=yes
```

- Monitoring multicast groups in the Bridge Multicast Database

```
[admin@MikroTik] > interface bridge mdb print
BRIDGE                  VID GROUP                                       PORTS
bridge1                 200 229.1.1.2                                   ether3
                                                                        ether2
                                                                        ether1
bridge1                 300 231.1.3.3                                   ether4
                                                                        ether3
                                                                        ether2
bridge1                 400 229.10.10.4                                 ether4
                                                                        ether3
bridge1                 500 234.5.1.5                                   ether5
                                                                        ether1
```

- Monitoring ports that are connected to a multicast router

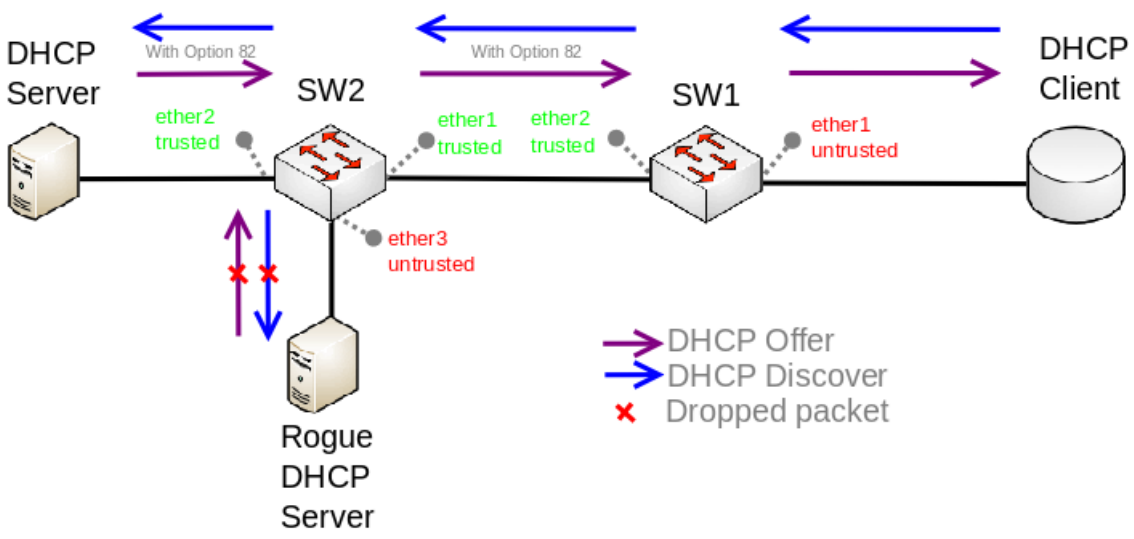
```
[admin@MikroTik] > /interface bridge port monitor [f]
              interface: ether1          ether2
                 status: in-bridge       in-bridge
            port-number: 1               2
                   role: designated-port designated-port
              edge-port: yes             yes
    edge-port-discovery: yes             yes
    point-to-point-port: yes             yes
           external-fdb: no              no
           sending-rstp: yes             yes
               learning: yes             yes
             forwarding: yes             yes
       multicast-router: yes              no
```

**Note:** IGMP membership reports are only forwarded to ports that are connected to a multicast router or to another IGMP Snooping enabled bridge. If no port is marked as a `multicast-router` then IGMP membership reports will not be forwarded to any port.

# DHCP Snooping and DHCP Option 82

**Sub-menu:** `/interface bridge` `/interface bridge port`

Starting from RouterOS version 6.43, bridge supports DHCP Snooping and DHCP Option 82. The DHCP Snooping is a Layer2 security feature, that limits unauthorized DHCP servers from providing a 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 an additional information (Agent Circuit ID and Agent Remote ID) provided by DHCP Snooping enabled devices that allows identifying the device itself and DHCP clients.




DHCP Snooping and Option 82 setup

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 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 form receiving any discovery messages and drop malicious DHCP server messages from ether3.

**Note:** Currently only CRS3xx devices fully support hardware 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 must 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 Bridge Hardware Offloading section with supported features.

# Bridge Firewall

**Sub-menu:** `/interface bridge filter, /interface bridge nat`

The bridge firewall implements packet filtering and thereby provides security functions that are used to manage data flow to, from and through bridge.

Packet flow diagram shows how packets are processed through router. It is possible to force bridge traffic to go through `/ip firewall filter` rules (see: 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 put 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.

## Properties

| 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 a 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 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 packet is matched by the rule:<br>▪ `accept` - accept the packet. Packet is not passed to next firewall rule<br>▪ `drop` - silently drop the packet<br>▪ `jump` - jump to the user defined chain specified by the value of `jump-target` parameter<br>▪ `log` - add a message to the system log containing following data: in-interface, out-interface, src-mac, protocol, src-ip:port->dst-ip:port and length of the packet. After packet is matched it is passed to next rule in the list, similar as `passthrough`<br>▪ `mark-packet` - place a mark specified by the new-packet-mark parameter on a packet that matches the rule<br>▪ `passthrough` - if packet is matched by the rule, increase counter and go to next rule (useful for statistics)<br>▪ `return` - passes control back to the chain from where the jump took place<br>▪ `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)<br>▪ `arp-nak` - negative ARP reply (rarely used, mostly in ATM networks)<br>▪ `drarp-error` - Dynamic RARP error code, saying that an IP address for the given MAC address can not be allocated<br>▪ `drarp-reply` - Dynamic RARP reply, with a temporaty IP address assignment for a host<br>▪ `drarp-request` - Dynamic RARP request to assign a temporary IP address for the given MAC address<br>▪ `inarp-reply` - InverseARP Reply<br>▪ `inarp-request` - InverseARP Request<br>▪ `reply` - standard ARP reply with a MAC address<br>▪ `reply-reverse` - reverse ARP (RARP) reply with an IP address assigned<br>▪ `request` - standard ARP request to a known IP address to find out unknown MAC address<br>▪ `request-reverse` - reverse ARP (RARP) request to a known MAC address to find out 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-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-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-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` property 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 (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)

**out-bridge** (*name*; Default: )

Outgoing bridge interface.

**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 certain packet mark.

| | |
|---|---|
| **packet-type** (*broadcast* \| *host* \| *multicast* \| *other-host*; Default: ) | MAC frame type:<br>▪ broadcast - broadcast MAC packet<br>▪ host - packet is destined to the bridge itself<br>▪ multicast - multicast MAC packet<br>▪ 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-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<br>▪ 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<br>▪ topology-change-ack - topology change acknowledgement 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:<br>▪ config - configuration BPDU<br>▪ tcn - topology change notification |
| **tls-host** (*string*; Default: ) | Allows to match https traffic based on TLS SNI hostname. Accepts GLOB syntax (https://en.wikipedia.org/wiki/Glob_(programming)) for wildcard matching. Note that matcher will not be able to match hostname if 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) |

### Notes

- STP matchers are only valid if 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 (**note**: it is not the industry-standard Ethernet frame format used in most networks worldwide!). These matchers are ignored for other packets.

## Bridge Packet Filter

**Sub-menu:** /interface bridge filter

This section describes bridge packet filter specific filtering options, that are specific to '/interface bridge filter'.

### Properties

| Property | Description |
|---|---|
| **action** (*accept* \| *drop* \| *jump* \| *log* \| *mark-packet* \| *passthrough* \| *return* \| *set-priority*; Default: **accept**) | Action to take if packet is matched by the rule:<br>▪ 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<br>▪ drop - silently drop the packet (without sending the ICMP reject message)<br>▪ jump - jump to the chain specified by the value of the jump-target argument<br>▪ log - add a message to the system log containing 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 next rule in the list, similar as passthrough<br>▪ mark - mark the packet to use the mark later<br>▪ passthrough - ignore this rule and go on to the next one. Acts the same way as a disabled rule, except for ability to count packets<br>▪ return - return to the previous chain, from where the jump took place<br>▪ 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

**Sub-menu:** `/interface bridge nat`

This section describes bridge NAT options, that are specific to `'/interface bridge nat'`.

## Properties

| 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 packet is matched by the rule:<br>▪ `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<br>▪ `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)<br>▪ `drop` - silently drop the packet (without sending the ICMP reject message)<br>▪ `dst-nat` - change destination MAC address of a packet (only valid in dstnat chain)<br>▪ `jump` - jump to the chain specified by the value of the jump-target argument<br>▪ `log` - log the packet<br>▪ `mark` - mark the packet to use the mark later<br>▪ `passthrough` - ignore this rule and go on to the next one. Acts the same way as a disabled rule, except for ability to count packets<br>▪ `redirect` - redirect the packet to the bridge itself (only valid in dstnat chain)<br>▪ `return` - return to the previous chain, from where the jump took place<br>▪ `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><br>▪ `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 series switches
- Swith chip features
- MTU on RouterBOARD
- Layer2 misconfiguration
- Bridge VLAN Table
- Wireless VLAN Trunk
- VLANs on Wireless

[ Top | Back to Content ]