CRS1xx/2xx series switches examples

Summary

Basic use cases and configuration examples for Cloud Router Switch features.

This article applies to CRS1xx and CRS2xx series switches and not to CRS3xx series switches. For CRS3xx series devices read the CRS3xx, CRS5xx series switches and CCR2116, CCR2216 manual.

Port switching

To set up port switching on CRS1xx/2xx series switches, check the Bridge Hardware Offloading page.

It is possible to create multiple isolated switch groups by using multiple bridges with enabled hardware offloading, this is possible only on CRS1xx/2xx series switches. For more complex setups (for example, VLAN filtering) you should use the port isolation feature instead.

Management access configuration

In general, switches are only supposed to forward packets by using the built-in switch chip, but not allow access to the device itself for security reasons. It is possible to use the device's serial port for management access, but in most cases, such an access method is not desired and access using an IP address is more suitable. In such cases, you will need to configure management access.

In all types of management access it is assumed that ports must be switched together, use the following commands to switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

You should also assign an IP address to the bridge interface so the device is reachable using an IP address (the device is also reachable using a MAC address):

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

Untagged

If invalid VLAN filtering is not enabled, management access to the device using tagged or untagged (**VLAN 0**) traffic is already allowed from any port, though this is not a good practice, this can cause security issues and can cause the device's CPU to be overloaded in certain situations (most commonly with a broadcast type of traffic).

If you intend to use invalid VLAN filtering (which you should), then ports, from which you are going to access the switch, must be added to the VLAN table for untagged (**VLAN 0**) traffic, for example, in case you want to access the switch from **ether2**:

```
/interface ethernet switch vlan
add vlan-id=0 ports=ether2,switch1-cpu
```

Tagged

Allowing only tagged traffic to have management access to the device through a specific port is a much better practice. For example, to allow only **VLAN99** to access the device through **ether2** you should first add an entry to the VLAN table, which will allow the selected port and the CPU port (**switch1-cpu**) to forward the selected VLAN ID, therefore allowing management access:

```
/interface ethernet switch vlan
add ports=ether2,switch1-cpu vlan-id=99
```

Packets that will be sent out from the CPU, for example, ping replies will not have a VLAN tag, to solve this you need to specify which ports should always send out packets with a VLAN tag for a specific VLAN ID:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2,switch1-cpu vlan-id=99
```

After a valid VLAN99 configuration has been set up, you can enable unknown/invalid VLAN filtering, which will not allow the management access through different ports than specified in the VLAN table:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether3,ether4,ether5
```

In this example VLAN99 will be used to access the device, a VLAN interface on the bridge must be created and an IP address must be assigned to it.

```
/interface vlan
add interface=bridge1 name=MGMT vlan-id=99
/ip address
add address=192.168.99.1/24 interface=MGMT
```

VLAN

It is recommended to get a Serial Console cable and testing it before configuring VLANs because you may lose access to the CPU and/or the port you are connected to.

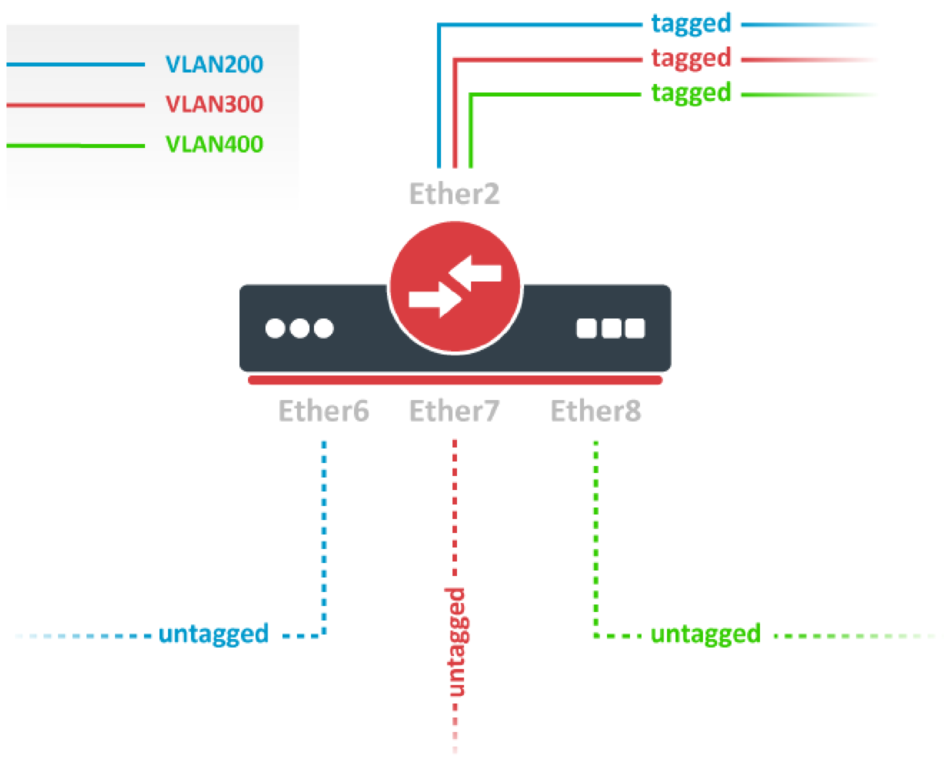
Some changes may take some time to take effect due to already-learned MAC addresses. In such cases flushing the Unicast Forwarding Database can help: `/interface ethernet switch unicast-fdb flush`

Multiple hardware offloaded bridge configuration is designed as a fast and simple port isolation solution, but it limits part of the VLAN functionality supported by the CRS switch-chip. For advanced configurations use one bridge within the CRS switch chip for all ports, configure VLANs, and isolate port groups with port isolation profile configuration.

Port Based VLAN

For CRS3xx series devices, you must use bridge VLAN filtering, you can read more about it in the Bridge VLAN Filtering section.

Example 1 (Trunk and Access ports)



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Specify the VLAN ID that the switch must set on untagged (VLAN0) traffic for each access port:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```

When an entry is created under `/interface ethernet switch ingress-vlan-translation`, then the switch chip will add a VLAN tag on ingress frames on the specified port. To remove the VLAN tag on the same port for egress frames, an `/interface ethernet switch egress-vlan-tag` entry should be created for the same VLAN ID where only tagged ports are specified. If a specific VLAN is forwarded only between access ports, the `/interface ethernet switch egress-vlan-tag` entry should still be created without any tagged ports. Another option is to create extra entries under `/interface ethernet switch egress-vlan-translation` menu to set untagged (VLAN0) traffic.

You must also specify which VLANs should be sent out to the trunk port with a VLAN tag. Use the tagged-ports property to set up a trunk port:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=200
add tagged-ports=ether2 vlan-id=300
add tagged-ports=ether2 vlan-id=400
```

Add entries to the VLAN table to specify VLAN memberships for each port and each VLAN ID:

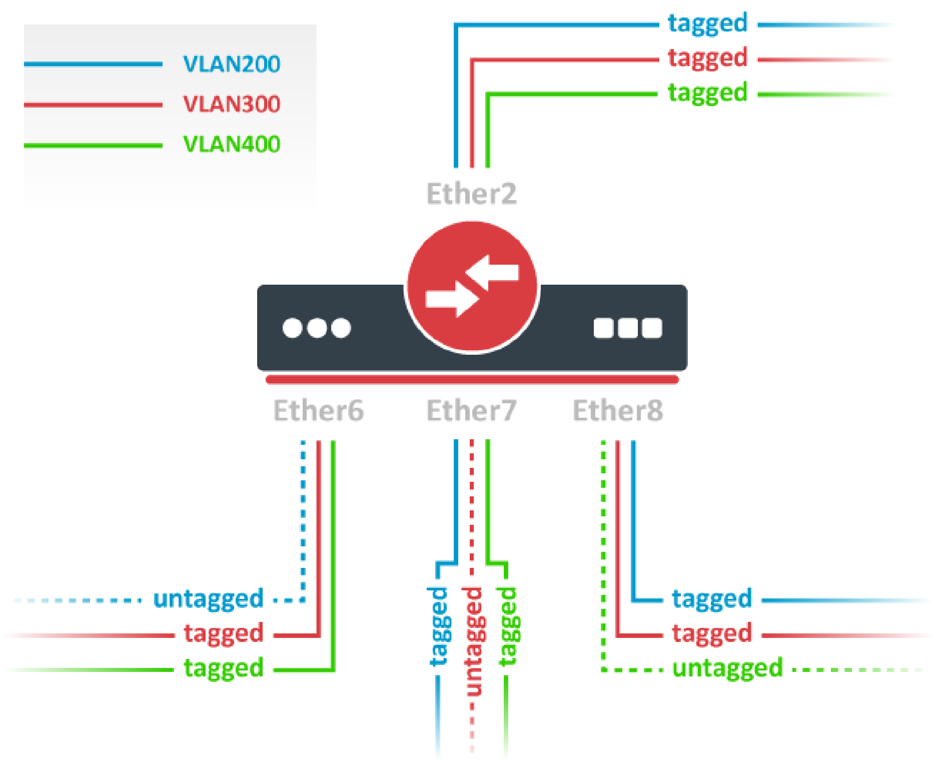
```
/interface ethernet switch vlan
add ports=ether2,ether6 vlan-id=200
add ports=ether2,ether7 vlan-id=300
add ports=ether2,ether8 vlan-id=400
```

After a valid VLAN configuration has been set up, you can enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```

It is possible to use the built-in switch chip and the CPU at the same time to create a Switch-Router setup, where a device acts as a switch and as a router simultaneously. You can find a configuration example in the CRS-Router guide.

Example 2 (Trunk and Hybrid Ports)



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Specify the VLAN ID that the switch must set on untagged (VLAN0) traffic for each access port:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```

By specifying ports as tagged-ports, the switch will always send out packets as tagged packets with the corresponding VLAN ID. Add appropriate entries according to the diagram above:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2,ether7,ether8 vlan-id=200
add tagged-ports=ether2,ether6,ether8 vlan-id=300
add tagged-ports=ether2,ether6,ether7 vlan-id=400
```

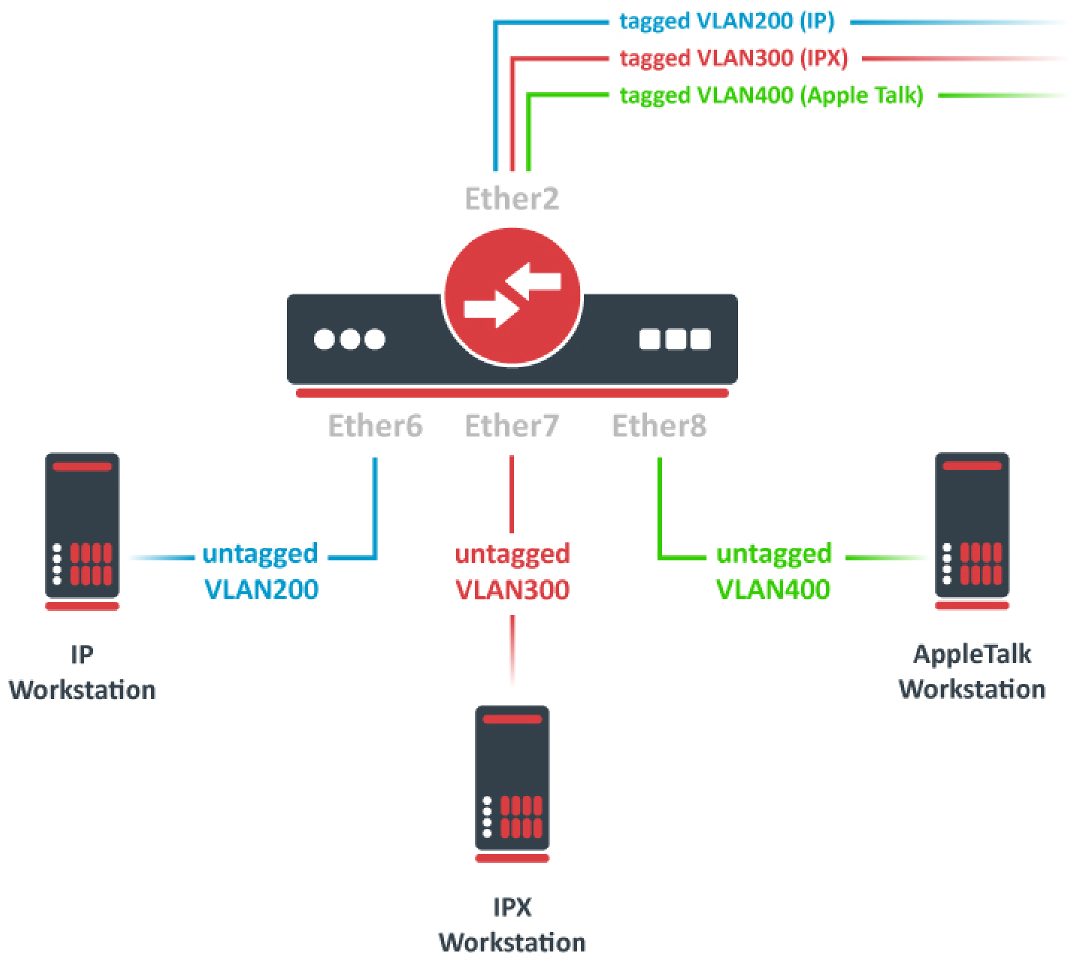
Add entries to the VLAN table to specify VLAN memberships for each port and each VLAN ID:

```
/interface ethernet switch vlan
add ports=ether2,ether6,ether7,ether8 vlan-id=200 learn=yes
add ports=ether2,ether6,ether7,ether8 vlan-id=300 learn=yes
add ports=ether2,ether6,ether7,ether8 vlan-id=400 learn=yes
```

After a valid VLAN configuration has been set up, you can enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```

Protocol Based VLAN



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Set VLAN for IP and ARP protocols:

```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=arp set-customer-vid-for=all new-customer-vid=0
add port=ether6 protocol=arp set-customer-vid-for=all new-customer-vid=200
add port=ether2 protocol=ip set-customer-vid-for=all new-customer-vid=0
add port=ether6 protocol=ip set-customer-vid-for=all new-customer-vid=200
```

Set VLAN for IPX protocol:

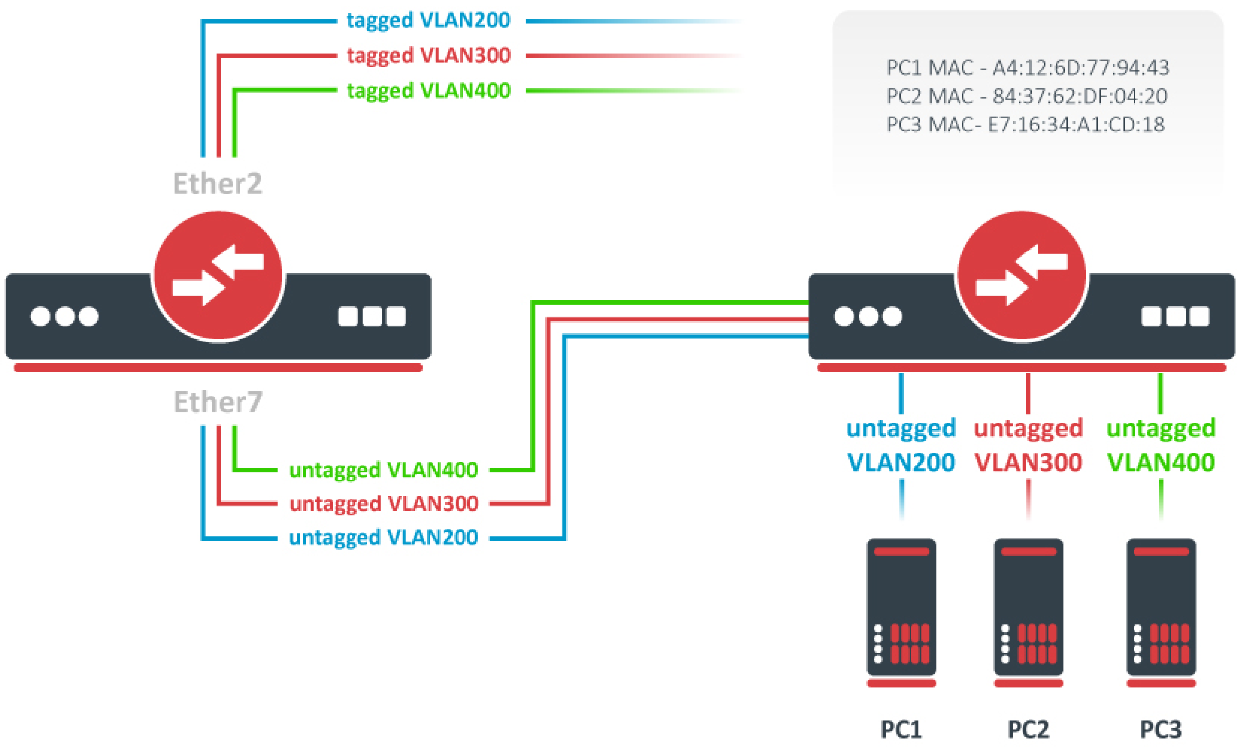
```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=ipx set-customer-vid-for=all new-customer-vid=0
add port=ether7 protocol=ipx set-customer-vid-for=all new-customer-vid=300
```

Set VLAN for AppleTalk AARP and AppleTalk DDP protocols:

```
/interface ethernet switch protocol-based-vlan
add port=ether2 protocol=0x80F3 set-customer-vid-for=all new-customer-vid=0
add port=ether8 protocol=0x80F3 set-customer-vid-for=all new-customer-vid=400
add port=ether2 protocol=0x809B set-customer-vid-for=all new-customer-vid=0
add port=ether8 protocol=0x809B set-customer-vid-for=all new-customer-vid=400
```

MAC Based VLAN

Internally all MAC addresses in MAC-based VLANs are hashed, certain MAC addresses can have the same hash, which will prevent a MAC address from being loaded into the switch chip if the hash matches with a hash from a MAC address that has been already loaded, for this reason, it is recommended to use Port bases VLANs in combination with MAC-based VLANs. This is a hardware limitation.



Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
```

Enable MAC-based VLAN translation on access port:

```
/interface ethernet switch port
set ether7 allow-fdb-based-vlan-translate=yes
```

Add MAC-to-VLAN mapping entries in the MAC-based VLAN table:

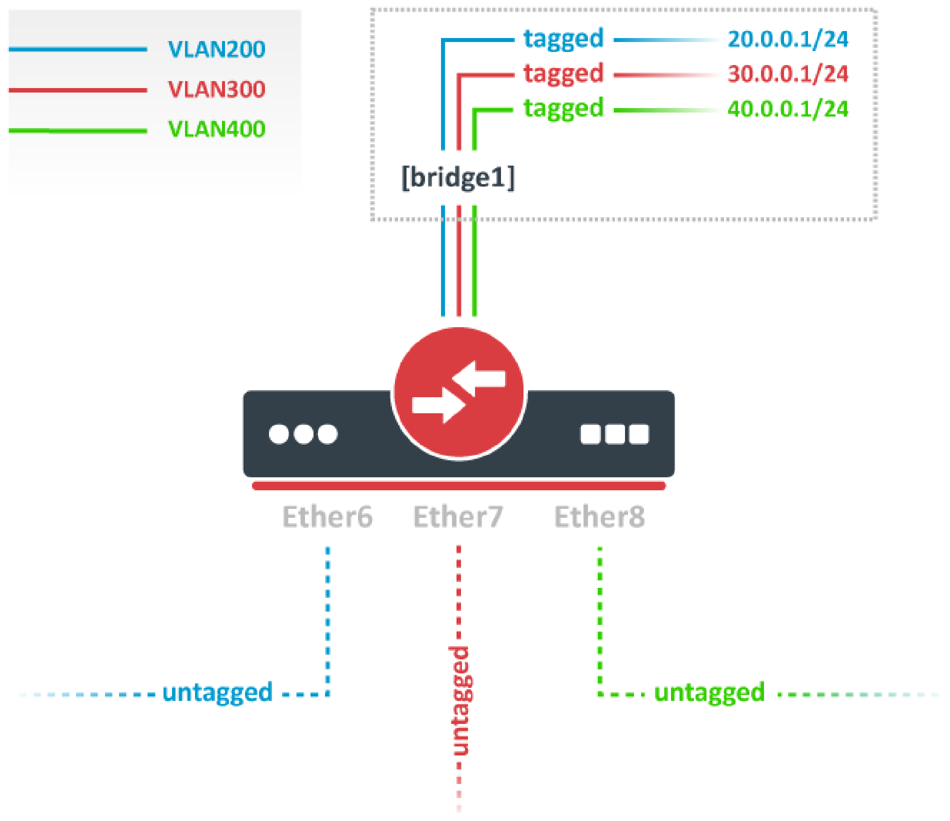
```
/interface ethernet switch mac-based-vlan
add src-mac=A4:12:6D:77:94:43 new-customer-vid=200
add src-mac=84:37:62:DF:04:20 new-customer-vid=300
add src-mac=E7:16:34:A1:CD:18 new-customer-vid=400
```

Add VLAN200, VLAN300, and VLAN400 tagging on the ether2 port to create it as a VLAN trunk port:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=200
add tagged-ports=ether2 vlan-id=300
add tagged-ports=ether2 vlan-id=400
```

Additionally, add entries to the VLAN table, specify VLAN membership for each port, and enable unknown/invalid VLAN filtering, see an example below - U nknown/Invalid VLAN filtering. This is required for network setups where more interfaces are added to the bridge, as it allows to define VLAN boundaries.

InterVLAN Routing



InterVLAN routing configuration consists of two main parts – VLAN tagging in switch-chip and routing in RouterOS. This configuration can be used in many applications by combining it with a DHCP server, Hotspot, PPP, and other features for each VLAN.

Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Set VLAN tagging on the CPU port for all VLANs to make packets tagged before they are routed:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=switch1-cpu vlan-id=200
add tagged-ports=switch1-cpu vlan-id=300
add tagged-ports=switch1-cpu vlan-id=400
```

add ingress VLAN translation rules to ensure that the correct VLAN ID assignment is done on access ports:

```
/interface ethernet switch ingress-vlan-translation
add ports=ether6 customer-vid=0 new-customer-vid=200
add ports=ether7 customer-vid=0 new-customer-vid=300
add ports=ether8 customer-vid=0 new-customer-vid=400
```

Create the VLAN interfaces on top of the bridge interface:

```
/interface vlan
add name=VLAN200 interface=bridge1 vlan-id=200
add name=VLAN300 interface=bridge1 vlan-id=300
add name=VLAN400 interface=bridge1 vlan-id=400
```

Make sure the VLAN interfaces are created on top of the bridge interface instead of any of the physical interfaces. If the VLAN interfaces are created on a slave interface, then the packet might not be received correctly, and therefore routing might fail. More detailed information can be found in the VLAN interface on a slave interface manual page.

Add IP addresses on created VLAN interfaces. In this example, three 192.168.x.1 addresses are added to VLAN200, VLAN300, and VLAN400 interfaces:

```
/ip address
add address=192.168.20.1/24 interface=VLAN200
add address=192.168.30.1/24 interface=VLAN300
add address=192.168.40.1/24 interface=VLAN400
```

Unknown/Invalid VLAN filtering

VLAN membership is defined in the VLAN table. Adding entries with VLAN ID and ports makes that VLAN traffic valid on those ports. After a valid VLAN configuration has been set up, unknown/invalid VLAN filtering can be enabled. This VLAN filtering configuration example applies to the InterVLAN Routing s etup.

```
/interface ethernet switch vlan
add ports=switch1-cpu,ether6 vlan-id=200
add ports=switch1-cpu,ether7 vlan-id=300
add ports=switch1-cpu,ether8 vlan-id=400
```

- Option 1: disable invalid VLAN forwarding on specific ports (more common):

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether2,ether6,ether7,ether8
```

- Option 2: disable invalid VLAN forwarding on all ports:

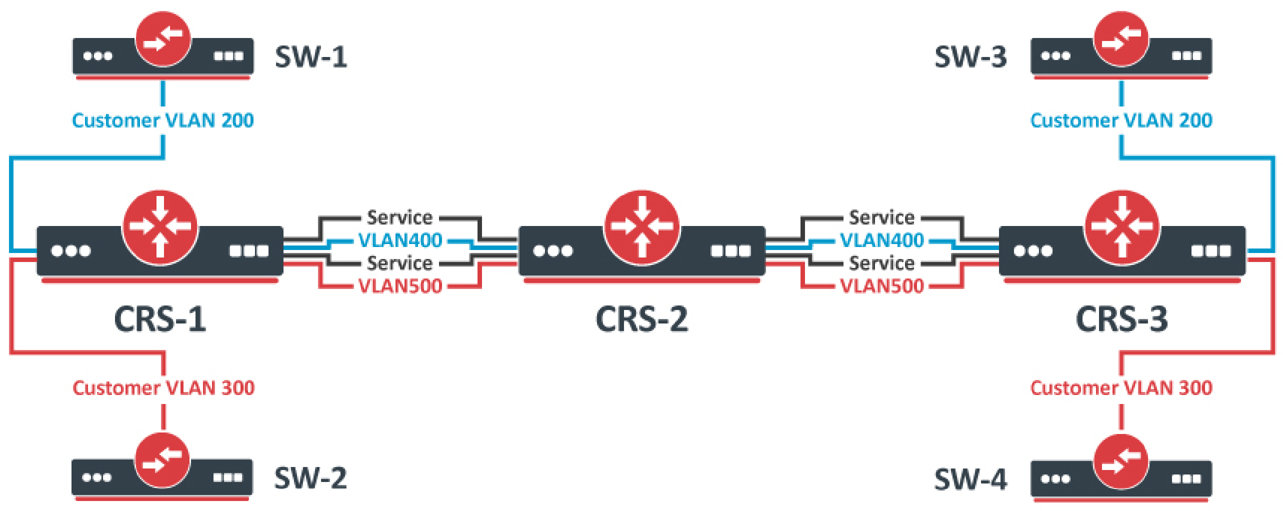
```
/interface ethernet switch
set forward-unknown-vlan=no
```

Using multiple bridges on a single switch chip with enabled unknown/invalid VLAN filtering can cause unexpected behavior. You should always use a single bridge configuration whenever using VLAN filtering. If port isolation is required, then the port isolation feature should be used instead of using multiple bridges.

VLAN Tunneling (Q-in-Q)

This example covers a typical VLAN tunneling use case where service provider devices add another VLAN tag for independent forwarding in the meantime allowing customers to use their own VLANs.

This example contains only the Service VLAN tagging part. It is recommended to additionally set Unknown/Invalid VLAN filtering configuration on ports.



CRS-1: The first switch on the edge of the service provider network has to properly identify traffic from the customer VLAN ID on port and assign a new service VLAN ID with ingress VLAN translation rules. VLAN trunk port configuration for service provider VLAN tags is in the same `egress-vlan-tag` tabl e. The main difference from basic Port-Based VLAN configuration is that the CRS switch-chip has to be set to do forwarding according to service (*outer*) VLAN ID instead of customer (*inner*) VLAN ID.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether9 hw=yes

/interface ethernet switch ingress-vlan-translation
add customer-vid=200 new-service-vid=400 ports=ether1
add customer-vid=300 new-service-vid=500 ports=ether2

/interface ethernet switch egress-vlan-tag
add tagged-ports=ether9 vlan-id=400
add tagged-ports=ether9 vlan-id=500

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CRS-2: The second switch in the service provider network requires only switched ports to do forwarding according to service (*outer*) VLAN ID instead of customer (*inner*) VLAN ID.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether9 hw=yes
add bridge=bridge1 interface=ether10 hw=yes

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CRS-3: The third switch has a similar configuration to CRS-1:

- Ports in a switch group using a bridge;
- Ingress VLAN translation rules to define new service VLAN assignments on ports;
- tagged-ports for service provider VLAN trunks;
- CRS switch-chip set to use service VLAN ID in switching lookup.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether10 hw=yes

/interface ethernet switch ingress-vlan-translation
add customer-vid=200 new-service-vid=400 ports=ether3
add customer-vid=300 new-service-vid=500 ports=ether4

/interface ethernet switch egress-vlan-tag
add tagged-ports=ether10 vlan-id=400
add tagged-ports=ether10 vlan-id=500

/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

CVID Stacking

It is possible to use CRS1xx/CRS2xx series switches for CVID Stacking setups. CRS1xx/CRS2xx series switches are capable of VLAN filtering based on the outer tag of tagged packets that have two CVID tags (double CVID tag), these switches are also capable of adding another CVID tag on top of an existing CVID tag (CVID Stacking). For example, in a setup where **ether1** is receiving tagged packets with CVID 10, but it is required that **ether2** sends out these packets with another tag CVID 20 (VLAN10 inside VLAN20) while filtering out any other VLANs, the following must be configured:

Switch together **ether1** and **ether2**:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
```

Set the switch to filter VLANs based on the service tag (0x88a8):

```
/interface ethernet switch
set bridge-type=service-vid-used-as-lookup-vid
```

Add a service tag SVID 20 to packets that have a CVID 10 tag on **ether1**:

```
/interface ethernet switch ingress-vlan-translation
add customer-vid=10 new-service-vid=20 ports=ether1
```

Specify **ether2** as the tagged/trunk port for SVID 20:

```
/interface ethernet switch egress-vlan-tag
add tagged-ports=ether2 vlan-id=20
```

Allow **ether1** and **ether2** to forward SVID 20:

```
/interface ethernet switch vlan
add ports=ether1,ether2 vlan-id=20
```

Override the SVID EtherType (0x88a8) to CVID EtherType (0x8100) on **ether2**:

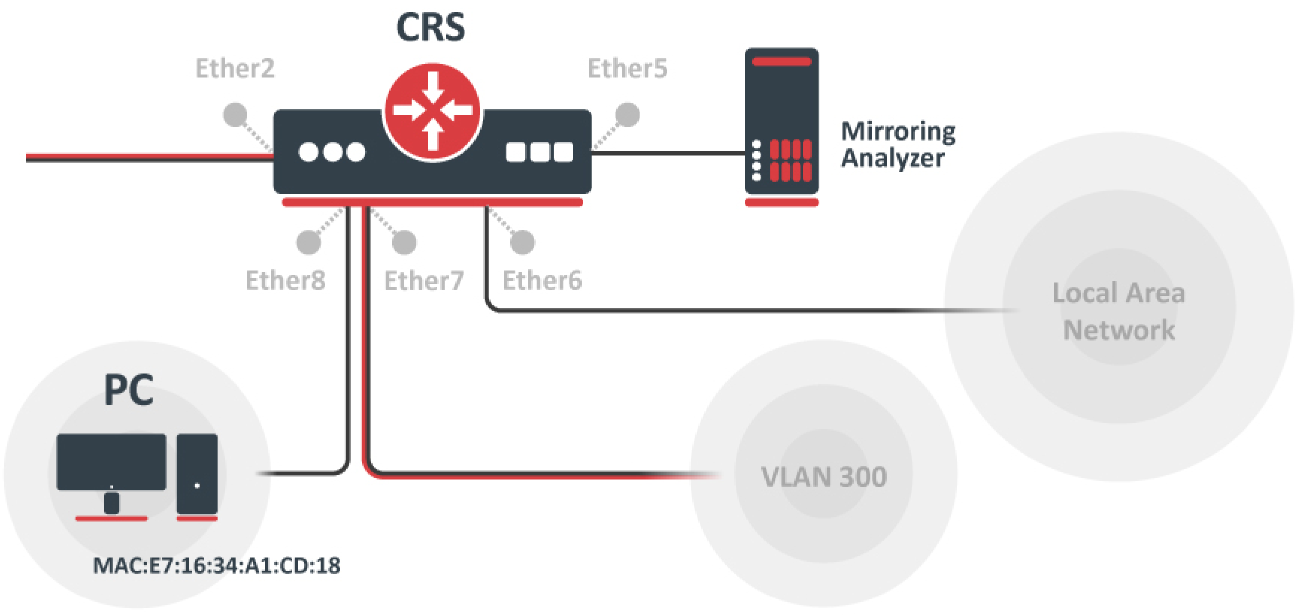
```
/interface ethernet switch port
set ether2 egress-service-tpid-override=0x8100 ingress-service-tpid-override=0x8100
```

Enable unknown/invalid VLAN filtering:

```
/interface ethernet switch
set drop-if-invalid-or-src-port-not-member-of-vlan-on-ports=ether1,ether2
```

Since the switch is set to look up VLAN ID based on the service tag, which is overridden with a different EtherType, then VLAN filtering is only done on the outer tag of a packet, the inner tag is not checked.

Mirroring



The Cloud Router Switches support three types of mirroring. Port-based mirroring can be applied to any switch-chip ports, VLAN-based mirroring works for all specified VLANs regardless of switch-chip ports, and MAC-based mirroring copies traffic sent or received from specific device reachable from the port configured in Unicast Forwarding Database.

Port-Based Mirroring

The first configuration sets the ether5 port as a mirror0 analyzer port for both ingress and egress mirroring, mirrored traffic will be sent to this port. Port-based ingress and egress mirroring are enabled from the ether6 port.

```
/interface ethernet switch
set ingress-mirror0=ether5 egress-mirror0=ether5

/interface ethernet switch port
set ether6 ingress-mirror-to=mirror0 egress-mirror-to=mirror0
```

VLAN Based Mirroring

The second example requires ports to be switched in a group. Mirroring configuration sets the ether5 port as a mirror0 analyzer port and sets the mirror0 port to be used when mirroring from VLAN occurs. VLAN table entry enables mirroring only for VLAN 300 traffic between ether2 and ether7 ports.

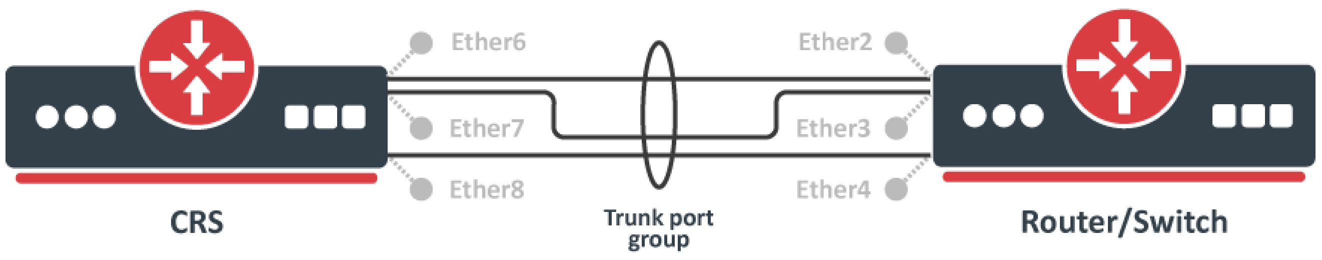
```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether7 hw=yes

/interface ethernet switch
set ingress-mirror0=ether5 vlan-uses=mirror0

/interface ethernet switch vlan
add ports=ether2,ether7 vlan-id=300 learn=yes ingress-mirror=yes
```

MAC Based Mirroring

The third configuration also requires ports to be switched in a group. Mirroring configuration sets the ether5 port as a mirror0 analyzer port and sets the mirror0 port to be used when mirroring from the Unicast Forwarding database occurs. The entry from the Unicast Forwarding database enables mirroring for packets with source or destination MAC address E7:16:34:A1:CD:18 from ether8 port.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether8 hw=yes

/interface ethernet switch
set ingress-mirror0=ether5 fdb-uses=mirror0

/interface ethernet switch unicast-fdb
add port=ether8 mirror=yes svl=yes mac-address=E7:16:34:A1:CD:18
```

Trunking



The Trunking in the Cloud Router Switches provides static link aggregation groups with hardware automatic failover and load balancing. IEEE802.3ad and IEEE802.1ax compatible Link Aggregation Control Protocol is not supported yet. Up to 8 Trunk groups are supported with up to 8 Trunk member ports per Trunk group.

Configuration requires a group of switched ports and an entry in the Trunk table:

```
/interface bridge
add name=bridge1 protocol-mode=none
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes

/interface ethernet switch trunk
add name=trunk1 member-ports=ether6,ether7,ether8
```

This example also shows proper bonding configuration in RouterOS on the other end:

```
/interface bonding
add name=bonding1 slaves=ether2,ether3,ether4 mode=balance-xor transmit-hash-policy=layer-2-and-3
```

You can find a working example for trunking and port-based VLANs on CRS VLANs with Trunks page.

Bridge (R)STP is not aware of the underlying switch trunking configuration and some trunk ports can move to a discarding or blocking state. When trunking member ports are connected to other bridges, you should either disable the (R)STP or filter out any BPDU between trunked devices (e.g. with ACL rules).

Limited MAC Access per Port

Disabling MAC learning and configuring static MAC addresses gives the ability to control what exact devices can communicate to CRS1xx/2xx switches and through them.

Configuration requires a group of switched ports, disabled MAC learning on those ports, and static UFDB entries:

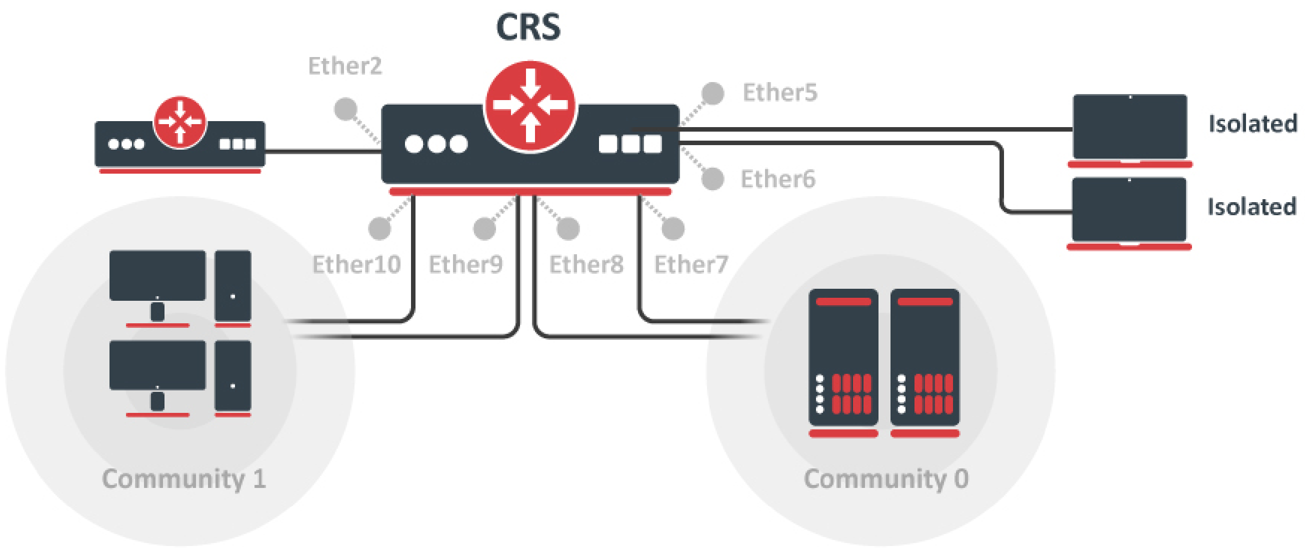
```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes learn=no unknown-unicast-flood=no
add bridge=bridge1 interface=ether7 hw=yes learn=no unknown-unicast-flood=no

/interface ethernet switch unicast-fdb
add mac-address=4C:5E:0C:00:00:01 port=ether6 svl=yes
add mac-address=D4:CA:6D:00:00:02 port=ether7 svl=yes

/interface ethernet switch acl
add action=drop src-mac-addr-state=sa-not-found src-ports=ether6,ether7 table=egress
add action=drop src-mac-addr-state=static-station-move src-ports=ether6,ether7 table=egress
```

CRS1xx/2xx switches also allow to learn one dynamic MAC per port to ensure only one end-user device is connected no matter its MAC address:

```
/interface ethernet switch port
set ether6 learn-limit=1
set ether7 learn-limit=1
```

Isolation

Port Level Isolation



Port-level isolation is often used for Private VLAN, where:

- One or multiple uplink ports are shared among all users for accessing the gateway or router.
- Port group Isolated Ports is for guest users. Communication is through the uplink ports only.
- Port group Community 0 is for department A. Communication is allowed between the group members and through uplink ports.
- Port group Community X is for department X. Communication is allowed between the group members and through uplink ports.

The Cloud Router Switches use port-level isolation profiles for Private VLAN implementation:

- Uplink ports – port-level isolation profile 0
- Isolated ports – port-level isolation profile 1
- Community 0 ports - port-level isolation profile 2
- Community X (X <= 30) ports - port-level isolation profile X

This example requires a group of switched ports. Assume that all ports used in this example are in one switch group.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
add bridge=bridge1 interface=ether9 hw=yes
add bridge=bridge1 interface=ether10 hw=yes
```

The first part of port isolation configuration is setting the Uplink port – set a port profile to 0 for ether2:

```
/interface ethernet switch port
set ether2 isolation-leakage-profile-override=0
```

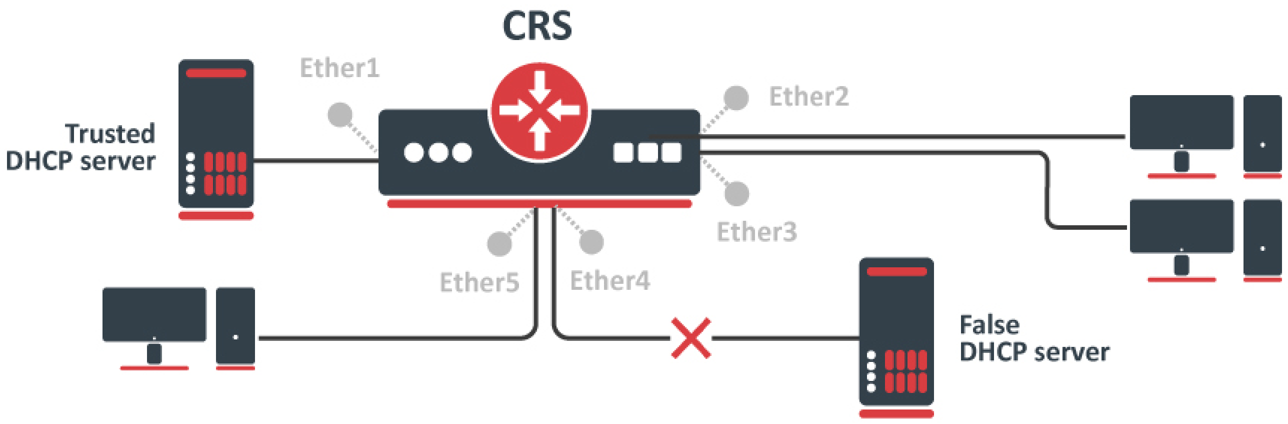
Then continue with setting isolation profile 1 to all isolated ports and adding the communication port for port isolation profile 1:

```
/interface ethernet switch port
set ether5 isolation-leakage-profile-override=1
set ether6 isolation-leakage-profile-override=1

/interface ethernet switch port-isolation
add port-profile=1 ports=ether2 type=dst
```

Configuration to set Community 2 and Community 3 ports is similar:

```
/interface ethernet switch port
set ether7 isolation-leakage-profile-override=2
set ether8 isolation-leakage-profile-override=2

/interface ethernet switch port-isolation
add port-profile=2 ports=ether2,ether7,ether8 type=dst

/interface ethernet switch port
set ether9 isolation-leakage-profile-override=3
set ether10 isolation-leakage-profile-override=3

/interface ethernet switch port-isolation
add port-profile=3 ports=ether2,ether9,ether10 type=dst
```

Protocol Level Isolation



Protocol level isolation on CRS switches can be used to enhance network security. For example, restricting DHCP traffic between the users and allowing it only to trusted DHCP server ports can prevent security risks like DHCP spoofing attacks. The following example shows how to configure it on CRS.

Switch together the required ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
add bridge=bridge1 interface=ether4 hw=yes
add bridge=bridge1 interface=ether5 hw=yes
```

Set the same Community port profile for all DHCP client ports. Community port profile numbers are from 2 to 30.

```
/interface ethernet switch port
set ether2 isolation-leakage-profile-override=2
set ether3 isolation-leakage-profile-override=2
set ether4 isolation-leakage-profile-override=2
set ether5 isolation-leakage-profile-override=2
```

And configure port isolation/leakage profile for selected Community (2) to allow DHCP traffic destined only to the port where the trusted DHCP server is located. registration status and traffic-type properties have to be set empty to apply restrictions only for DHCP protocol.

```
/interface ethernet switch port-isolation
add port-profile=2 protocol-type=dhcpv4 type=dst forwarding-type=bridged ports=ether1 registration-status=""
traffic-type=""
```

Quality of Service (QoS)

QoS configuration schemes

MAC based traffic scheduling and shaping: [MAC address in UFDB] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

VLAN based traffic scheduling and shaping: [VLAN id in VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

Protocol based traffic scheduling and shaping: [Protocol in Protocol VLAN table] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper]

PCP/DEI based traffic scheduling and shaping: [Switch port PCP/DEI mapping] -> [Priority] -> [Queue] -> [Shaper]

DSCP based traffic scheduling and shaping: [QoS DSCP mapping] -> [Priority] -> [Queue] -> [Shaper]

MAC-based traffic scheduling using internal Priority

In Strict Priority scheduling mode, the highest priority queue is served first. The queue number represents the priority and the queue with the highest queue number has the highest priority. Traffic is transmitted from the highest priority queue until the queue is empty, and then moves to the next highest priority queue, and so on. If no congestion is present at the egress port, a packet is transmitted as soon as it is received. If congestion occurs in the port where high-priority traffic keeps coming, the lower-priority queues starve.

On all CRS switches the scheme where MAC-based egress traffic scheduling is done according to internal Priority would be the following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue];
In this example, host1 (E7:16:34:00:00:01) and host2 (E7:16:34:00:00:02) will have higher priority 1 and the rest of the hosts will have lower priority 0 for transmitted traffic on port ether7. Note that CRS has a maximum of 8 queues per port.

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create a QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entries to match specific MACs on ether7 and apply the QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:00:00:01 port=ether7 qos-group=group1 svl=yes
add mac-address=E7:16:34:00:00:02 port=ether7 qos-group=group1 svl=yes
```

Configure ether7 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 qos-scheme-
precedence=da-based
```

MAC-based traffic shaping using internal Priority

The scheme where MAC-based traffic shaping is done according to internal Priority would be following: [MAC address] -> [QoS Group] -> [Priority] -> [Queue] -> [Shaper];
In this example, unlimited traffic will have priority 0 and limited traffic will have priority 1 with a bandwidth limit of 10Mbit. Note that CRS has a maximum of 8 queues per port.

Create a group of ports for switching:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create a QoS group for use in UFDB:

```
/interface ethernet switch qos-group
add name=group1 priority=1
```

Add UFDB entry to match specific MAC on ether8 and apply QoS group1:

```
/interface ethernet switch unicast-fdb
add mac-address=E7:16:34:A1:CD:18 port=ether8 qos-group=group1 svl=yes
```

Configure ether8 port queues to work according to Strict Priority and QoS scheme only for destination address:

```
/interface ethernet switch port
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1 qos-scheme-
precedence=da-based
```

Apply bandwidth limit for queue1 on ether8:

```
/interface ethernet switch shaper
add port=ether8 rate=10M target=queue1
```

If the CRS switch supports Access Control List, this configuration is simpler:

```
/interface ethernet switch acl policer
add name=policer1 yellow-burst=100k yellow-rate=10M

/interface ethernet switch acl
add mac-dst-address=E7:16:34:A1:CD:18 policer=policer1
```

VLAN-based traffic scheduling + shaping using internal Priorities

The best practice is to assign lower internal QoS Priority for traffic limited by shaper to make it also less important in the Strict Priority scheduler. (higher priority should be more important and unlimited)

In this example:
Switch port ether6 is using a shaper to limit the traffic that comes from ether7 and ether8.
When the link has reached its capacity, the traffic with the highest priority will be sent out first.
VLAN10 -> QoS group0 = lowest priority
VLAN20 -> QoS group1 = normal priority
VLAN30 -> QoS group2 = highest priority

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether6 hw=yes
add bridge=bridge1 interface=ether7 hw=yes
add bridge=bridge1 interface=ether8 hw=yes
```

Create QoS groups for use in the VLAN table.

```
/interface ethernet switch qos-group
add name=group0 priority=0
add name=group1 priority=1
add name=group2 priority=2
```

Add VLAN entries to apply QoS groups for certain VLANs.

```
/interface ethernet switch vlan
add ports=ether6,ether7,ether8 qos-group=group0 vlan-id=10
add ports=ether6,ether7,ether8 qos-group=group1 vlan-id=20
add ports=ether6,ether7,ether8 qos-group=group2 vlan-id=30
```

Configure ether6, ether7, and ether8 port queues to work according to Strict Priority and QoS schemes only for VLAN-based QoS.

```
/interface ethernet switch port
set ether6 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
set ether7 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
set ether8 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0" priority-to-queue=0:0,1:1,2:2 qos-scheme-
precedence=vlan-based
```

Apply bandwidth limit on ether6.

```
/interface ethernet switch shaper
add port=ether6 rate=10M
```

PCP-based traffic scheduling

By default, CRS1xx/CRS2xx series devices will ignore the PCP/CoS/802.1p value and forward packets based on FIFO (First-In-First-Out) manner. When the device's internal queue is not full, then packets are sent in a FIFO manner, but as soon as a queue is filled, then higher-priority traffic can be sent out first. Let us consider a scenario when **ether1** and **ether2** are forwarding data to **ether3**, but when **ether3** is congested, then packets are going to be scheduled, we can configure the switch to hold the lowest priority packets until all higher priority packets are sent out, this is a very common scenario for VoIP type setups, where some traffic needs to be prioritized.

To achieve such a behavior, switch together **ether1**, **ether2,** and **ether3** ports:

```
/interface bridge
add name=bridge1
/interface bridge port
add bridge=bridge1 interface=ether1 hw=yes
add bridge=bridge1 interface=ether2 hw=yes
add bridge=bridge1 interface=ether3 hw=yes
```

Enable **Strict Policy** for each internal queue on each port:

```
/interface ethernet switch port
set ether1,ether2,ether3 per-queue-scheduling="strict-priority:0,strict-priority:0,strict-priority:0,strict-
priority:0,strict-priority:0,strict-priority:0,strict-priority:0,strict-priority:0"
```

Map each PCP value to an internal priority value, for convenience reasons simply map PCP to an internal priority 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 pcp-based-qos-priority-mapping=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Since the switch will empty the largest queue first and you need the highest priority to be served first, then you can assign this internal priority to a queue 1-to-1:

```
/interface ethernet switch port
set ether1,ether2,ether3 priority-to-queue=0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7
```

Finally, set each switch port to schedule packets based on the PCP value:

```
/interface ethernet switch port
set ether1,ether2,ether3 qos-scheme-precedence=pcp-based
```

Bandwidth Limiting

Both Ingress Port policer and Shaper provide bandwidth-limiting features for CRS switches.

- Ingress Port Policer sets RX limit on port:

```
/interface ethernet switch ingress-port-policer
add port=ether5 meter-unit=bit rate=10M
```

- Shaper sets TX limit on port:

```
/interface ethernet switch shaper
add port=ether5 meter-unit=bit rate=10M
```

Traffic Storm Control

The same Ingress Port policer also can be used for traffic storm control to prevent disruptions on Layer 2 ports caused by broadcast, multicast, or unicast traffic storms.

- Broadcast storm control example on ether5 port with 500 packet limit per second:

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=500 meter-unit=packet packet-types=broadcast
```

- Example with multiple packet types which includes ARP and ND protocols and unregistered multicast traffic. Unregistered multicast is traffic that is not defined in the Multicast Forwarding Database.

```
/interface ethernet switch ingress-port-policer
add port=ether5 rate=5k meter-unit=packet packet-types=broadcast,arp-or-nd,unregistered-multicast
```

See also

- CRS1xx/2xx series switches examples
- CRS Router
- CRS1xx/2xx VLANs with Trunks
- Basic VLAN switching
- Bridge Hardware Offloading
- Spanning Tree Protocol
- IGMP Snooping
- DHCP Snooping and Option 82
- MTU on RouterBOARD
- Layer2 misconfiguration