Contents

[1. Linux Bridge 1](#_Toc174822901)

[Creating a bridge device 3](#_Toc174822902)

[Adding devices to a bridge 3](#_Toc174822903)

[Showing devices in a bridge 3](#_Toc174822904)

[2. VLAN 4](#_Toc174822905)

[3. VETH 5](#_Toc174822906)

[4. Dummy interface 6](#_Toc174822907)

[5. MACsec 7](#_Toc174822908)

[References: - 7](#_Toc174822909)

# Linux Bridge

A network bridge is a Link Layer device which forwards traffic between networks based on MAC addresses and is therefore also referred to as a Layer 2 device. It makes forwarding decisions based on tables of MAC addresses which it builds by learning what hosts are connected to each network. A software bridge can be used within a Linux host in order to emulate a hardware bridge, for example in virtualization applications for sharing a NIC with one or more virtual NICs.

In the context of KVM, a Linux bridge is used to connect the KVM guest interface to a KVM host network interface.

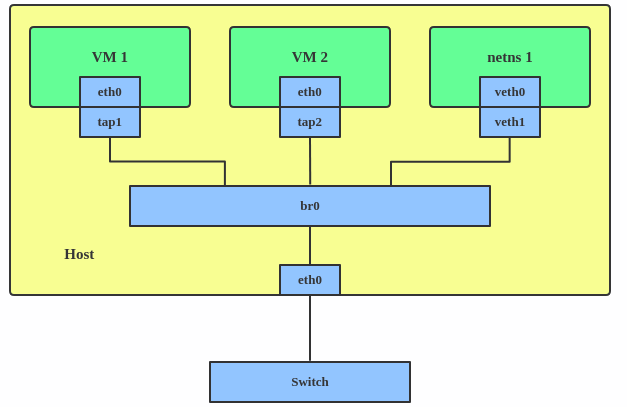
To create a Linux Bridge on a KVM host, use the following command:

**# brctl addbr <bridge-name>**

Next, if KVM guest interfaces that will be connecting to the bridge need to communicate with other systems external to the KVM host, the Linux bridge must be instructed which KVM host interface to use. To attach a Linux bridge to an OSA interface in the KVM host, use the following command:

# brctl addif <bridge-name> <host-interface-name>

Rough idea using below figure can be derived.



“bridge-utils” package can be built from the source code in bridge-utils GIT repository.

$ git clone -b main git://git.kernel.org/pub/scm/network/bridge/bridge-utils.git

$ cd bridge-utils

$ autoconf

$ ./configure

If your bridge-utilities have been correctly built and your kernel and bridge-module are OK, then issuing a **brctl** should show a small command synopsis.

|  |
| --- |
| # brctl  # commands:  addbr <bridge> add bridge  delbr <bridge> delete bridge  addif <bridge> <device> add interface to bridge  delif <bridge> <device> delete interface from bridge  setageing <bridge> <time> set ageing time  setbridgeprio <bridge> <prio> set bridge priority  setfd <bridge> <time> set bridge forward delay  sethello <bridge> <time> set hello time  setmaxage <bridge> <time> set max message age  setpathcost <bridge> <port> <cost> set path cost  setportprio <bridge> <port> <prio> set port priority  show show a list of bridges  showmacs <bridge> show a list of mac addrs  showstp <bridge> show bridge stp info  stp <bridge> <state> turn stp on/off |

## Creating a bridge device

The command

# brctl addbr "bridgename"

creates a logical bridge instance with the name bridgename. You will need at least one logical instance to do any bridging at all. You can interpret the logical bridge as a container for the interfaces taking part in the bridging. Each bridging instance is represented by a new network interface.

The corresponding shutdown command is:

# brctl delbr <bridgename>

## Adding devices to a bridge

The command

# brctl addif <bridgename> <device>

adds the network device “device” to take part in the bridging of “bridgename.” All the devices contained in a bridge act as one big network. It is not possible to add a device to multiple bridges or bridge a bridge device, because it just wouldn't make any sense! The bridge will take a short amount of time when a device is added to learn the Ethernet addresses on the segment before starting to forward.

The corresponding command to take an interface out of the bridge is:

#brctl delif <bridgename> <device>

## Showing devices in a bridge

The brctl show command gives you a summary about the overall bridge status, and the instances running as shown below:

# brctl addbr br549

# brctl addif br549 eth0

# brctl addif br549 eth1

# brctl show

bridge name bridge id STP enabled interfaces

br549 8000.00004c9f0bd2 no eth0

eth1

Once a bridge is running the “brctl showmacs” will show information about network addresses of traffic being forwarded (and the bridge itself).

# brctl showmacs br549

port no mac addr is local? ageing timer

1 00:00:4c:9f:0b:ae no 17.84

1 00:00:4c:9f:0b:d2 yes 0.00

2 00:00:4c:9f:0b:d3 yes 0.00

1 00:02:55:1a:35:09 no 53.84

1 00:02:55:1a:82:87 no 11.53

...

The aging time is the number of seconds a MAC address will be kept in the forwarding database after having received a packet from this MAC address. The entries in the forwarding database are periodically timed out to ensure they won't stay around forever. Normally there should be no need to modify this parameter, but it can be changed with (time is in seconds).

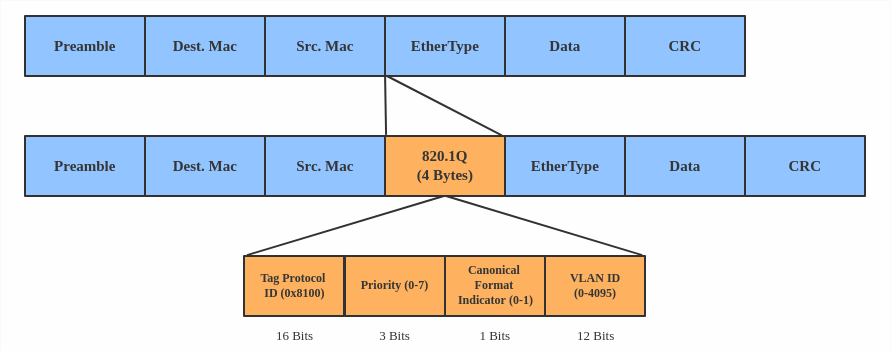
# brctl setageing <bridgename> <time>

Setting ageing time to zero makes all entries permanent.

# VLAN

A VLAN, aka virtual LAN, separates broadcast domains by adding tags to network packets. VLANs allow network administrators to group hosts under the same switch or between different switches.

The VLAN header looks like:



Use a VLAN when we want to separate subnet in VMs, namespaces, or hosts.

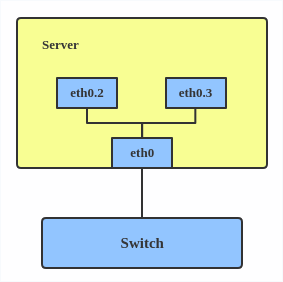
Here's how to create a VLAN:

# ip link add link eth0 name eth0.2 type vlan id 2

# ip link add link eth0 name eth0.3 type vlan id 3

This adds VLAN 2 with name **eth0.2** and VLAN 3 with name **eth0.3**.

The topology looks like this:

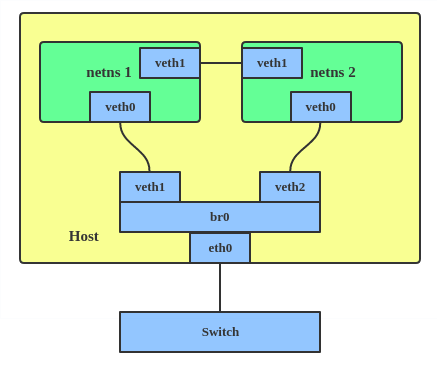


Note*:* When configuring a VLAN, you need to make sure the switch connected to the host is able to handle VLAN tags, for example, by setting the switch port to trunk mode.

# VETH

The VETH (virtual Ethernet) device is a local Ethernet tunnel. Devices are created in pairs, as shown in the diagram below.

Packets transmitted on one device in the pair are immediately received on the other device. When either device is down, the link state of the pair is down.



Use a VETH configuration when namespaces need to communicate to the main host namespace or between each other.

Here's how to set up a VETH configuration:

# ip netns add net1

# ip netns add net2

# ip link add veth1 netns net1 type veth peer name veth2 netns net2

This creates two namespaces, net1 and net2, and a pair of VETH devices, and it assigns veth1 to namespace net1 and veth2 to namespace net2. These two namespaces are connected with this VETH pair. Assign a pair of IP addresses, and you can ping and communicate between the two namespaces.

# Dummy interface

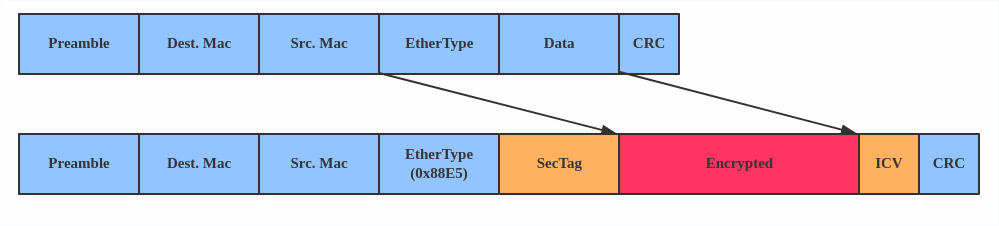
A dummy interface is entirely virtual like, for example, the loopback interface. The purpose of a dummy interface is to provide a device to route packets through without actually transmitting them.

Use a dummy interface to make an inactive SLIP (Serial Line Internet Protocol) address look like a real address for local programs. Nowadays, a dummy interface is mostly used for testing and debugging.

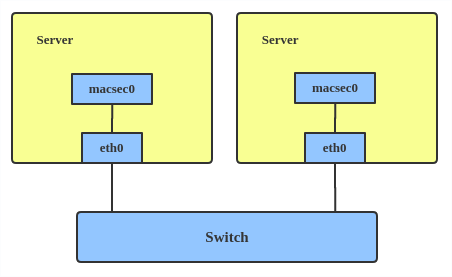
Here's how to create a dummy interface:

# MACsec

MACsec (Media Access Control Security) is an IEEE standard for security in wired Ethernet LANs. Similar to IPsec, as a layer 2 specification, MACsec can protect not only IP traffic but also ARP, neighbor discovery, and DHCP. The MACsec headers look like this:



The main use case for MACsec is to secure all messages on a standard LAN including ARP, NS, and DHCP messages.



Here's how to set up a MACsec configuration:

# ip link add macsec0 link eth1 type macsec

Note: This only adds a MACsec device called macsec0 on interface eth1.

# References: -

1. <https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking#team_device>
2. <https://wiki.linuxfoundation.org/networking/bridge>
3. https://www.ibm.com/docs/en/linux-on-systems?topic=choices-using-linux-bridge