MidoNet Operations Guide

5.4 (2017-03-20 11:27 UTC)

midonet

docs.midonet.org

MidoNet Operations Guide

5.4 (2017-03-20 11:27 UTC)

MidoNet is a network virtualization software for Infrastructure-as-a-Service (IaaS) clouds.

It decouples your IaaS cloud from your network hardware, creating an intelligent software abstraction layer between your end hosts and your physical network.

This guide includes instructions on creating routers, bridges, and ports. It also describes rule chains and several MidoNet features, including L4 load balancing, resource protection, NAT configuration, handling IP packet fragments, and L2 address matching.

Note

Please consult the MidoNet Mailing Lists or Chat if you need assistance.

Table of Contents

List of Figures

List of Tables

Preface

Conventions

The MidoNet documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt

Any user, including the root user, can run commands that are prefixed with the $ prompt.

 # prompt

The root user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

1. Configuring uplinks

Table of Contents

This section describes how to configure uplinks from a MidoNet-enabled cloud to an external network.

The basic steps for configuring uplinks are:

1. Connect a virtual port to an exterior port.
2. Set up a router. This includes configuring the routes for the traffic between networks.
3. Configure dynamic routing to allow for the exchange of routes between your local autonomous system (AS) and other autonomous systems. MidoNet supports BGP, which is an exterior routing protocol that allows MidoNet to advertize routes, such as the route to the network associated with floating IPs, and receive routes and reachability information from BGP peers.

Edge Router Setup

Prior to v5.0, with Neutron, you could set up the gateway in only one way, which was to have a special singleton gateway router called the Provider Router created implicitly when an external network was created in Neutron. The provider router sits at the edge of the cloud and interfaces with the uplink router. The Provider Router is where BGP was typically configured. The biggest limitation of this approach was that it took away the scenario in which you wanted to have an L2 network at the edge instead of a router. Another limitation was that only one such router could exist for the entire cloud.

These limitations are removed in v5.0, where you could design your gateway to be either L2 network or router with as many routers as you wish, all using the Neutron API.

There are two main changes:

Edge Router

The Provider Router is no longer implicitly created upon the external network creation. Instead, the edge gateway routers, called the Edge Routers, are created explicitly using standard Neutron API. With this approach, multiple Edge Routers can be created, and they are optional.

Gateway Virtual Topology

In the previous model, the Provider Router was connected directly to the tenant routers, with the external networks hanging off of the Provider Router.

In the new model, the external networks exist between the edge and the tenant routers.

To create the gateway topology issue the following Neutron commands.

Create a standard neutron router:

```
neutron router-create <EDGE_ROUTER_NAME>
```

Attach the edge router to an external network:

```
neutron router-interface-add <EDGE_ROUTER_ID> <EXT_SUBNET_ID>
```

Create a special network called `uplink` network, representing the physical network outside of the cloud:

```
neutron net-create <UPLINK_NET_NAME> --tenant_id admin --
provider:network_type uplink
```

Create a subnet for the uplink network matching the CIDR used in the uplink network (could just be /30 if linked directly to another router):

```
neutron subnet-create --tenant_id admin --disable-dhcp --name
 <UPLINK_SUBNET_NAME> <UPLINK_NET_NAME> <CIDR>
```

Create a port on the uplink network with a specific IP that you want to use and the binding details so that this virtual port gets bound to a specific NIC on the gateway host:

```
neutron port-create <UPLINK_NET_ID> --binding:host_id <HOST_NAME> --
binding:profile type=dict interface_name=<INTERFACE_NAME> --fixed-ip
 ip_address=<IP_ADDR>
```

Attach the uplink port to the Edge Router:

```
neutron router-interface-add <EDGE_ROUTER_ID> port=<UPLINK_PORT_ID>
```

BGP Setup

You set up a BGP link to connect MidoNet with an external Autonomous System (AS). This creates an up-link to an external network.

Typically, you connect a MidoNet network to the Internet through two independent up-link routers. In the simple case of connecting MidoNet to the Internet via two BGP-enabled routers, it's best to create two ports on the virtual router and bind them to network interfaces in two different hosts (two Gateway Nodes). This will distribute load between both Gateway Nodes hosts, as well as eliminate a single point of failure. The two ports must be configured as a stateful port pair, for details, refer to the section called "Stateful port groups" [18].

MidoNet uses quagga's bgpd to terminate BGP sessions on behalf of a virtual router. The routes that bgpd learns from its peers are added to the virtual router in MidoNet's topology. The quagga package is provided in the MidoNet release package repositories. Any system running Midolman should already have everything that it needs to set up the BGP. You have to keep in mind that bgpd processes run in the host where a particular virtual router port is bound.

Important

Make sure you have the following information before setting up the BGP: Local and peer Autonomous System (AS) Numbers for the BGP session. The IP address of the BGP peer.

BGP Uplink Configuration

MidoNet utilizes the Border Gateway Protocol (BGP) for external connectivity.

For production deployments it is strongly recommended to use BGP due to it's scalability and redundancy.

For demo or POC environments, alternatively static routing can be used.

The following instructions assume below sample environment:

- One floating IP network
  - *192.0.2.0/24*
- Two MidoNet gateway nodes
  - *gateway1*, connecting to *bgp1* via *eth1*
  - *gateway2*, connecting to *bgp2* via *eth1*
- Two remote BGP peers
  - *bgp1*, *198.51.100.1*, AS *64513*
  - *bgp2*, *203.0.113.1*, AS *64514*
- Corresponding MidoNet BGP peers
  - *198.51.100.2*, AS *64512*
  - *203.0.113.2*, AS *64512*

Follow these steps to configure the BGP uplinks.

1. Launch the MidoNet CLI and find the Edge Router

   ```
   midonet-cli> router list
   router router0 name Edge Router state up
   router router1 name Tenant Router state up infilter chain0 outfilter
    chain1
   ```

   In this example the Edge Router is **router0**.

2. Create and bind virtual ports for the BGP sessions

   Refer to the section called "Edge Router Setup" [1] for instructions on how to create the necessary ports and bind them to the Gateway hosts' physical network interfaces.

   You can confirm the port configuration within MidoNet CLI by listing the Edge Router's ports:

   ```
   midonet> router router0 port list
   port port0 device router0 state up mac fa:16:3e:11:11:11
    addresses 198.51.100.2/30
   port port1 device router0 state up mac fa:16:3e:22:22:22
    addresses 203.0.113.2/30
   [...]
   ```

3. Configure basic BGP settings

   ```
   midonet> router router0 set asn 64512

   midonet> router router0 add bgp-peer asn 64513 address 198.51.100.1
   router0:peer0
   ```

```
midonet> router router0 add bgp-peer asn 64514 address 203.0.113.1
router0:peer1

midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 198.51.100.1
peer peer1 asn 64514 address 203.0.113.1
```

4. If needed, configure MD5 authentication:

```
midonet> router router0 bgp-peer peer0 set password BGP_PASSWORD
midonet> router router0 bgp-peer peer1 set password BGP_PASSWORD
```

5. If needed, configure custom timers that will take precedence over the default ones defined in the MidoNet configuration:

```
midonet> router router0 bgp-peer peer0 set connect-retry 10
midonet> router router0 bgp-peer peer0 set hold-time 5
midonet> router router0 bgp-peer peer0 set keep-alive 5
midonet> router router0 bgp-peer peer1 set connect-retry 10
midonet> router router0 bgp-peer peer1 set hold-time 5
midonet> router router0 bgp-peer peer1 set keep-alive 5
midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 198.51.100.1 keep-alive 5 hold-time 5
 connect-retry 10
peer peer1 asn 64514 address 203.0.113.1 keep-alive 5 hold-time 5
 connect-retry 10
```

6. Add routes to the remote BGP peers

   In order to be able to establish connections to the remote BGP peers, corresponding routes have to be added.

```
midonet> router router0 route add src 0.0.0.0/0 dst 198.51.100.0/30
 port router0:port0 type normal
router0:route0

midonet> router router0 route add src 0.0.0.0/0 dst 203.0.113.0/30
 port router0:port1 type normal
router0:route1
```

7. Advertise BGP routes

   In order to provide external connectivity for hosted virtual machines, the floating IP network has to be advertised to the BGP peers.

```
midonet> router router0 add bgp-network net 192.0.2.0/24
router0:net0

midonet> router router0 list bgp-network
net net0 net 192.0.2.0/24
```

Adding a second session on the same router port

It may be useful or a good idea to add a second BGP session to this router port if there is a second uplink router available. That has two upsides as the host that owns the port binding for this router port will be able to load balance among both upstream routers and it will not be disconnected if only one of them fails.

To add a second peer to the same router port, you use the same command as for the first peer, adjusting its AS number and IP address. The router port on which MidoNet establishes the BGP session is chosen automatically based on the peer's IP address.

This will add a second peer to the example above:

```
midonet> router router0 add bgp-peer asn 64514 address 10.12.12.3
router0:peer1
midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 10.12.12.2 keep-alive 5 hold-time 5 connect-
retry 10
peer peer1 asn 64514 address 10.12.12.3
midonet>
```

Adding BGP sessions to a second router port

It's also wise to add one or more hosts that will handle North-South traffic for the MidoNet deployment. After all, a single router port routing to upstream is a single point of failure and could also be a performance bottleneck.

The solution is to add a second virtual port to the router and to bind it to a different physical host. With the proper routing setup, MidoNet will balance outgoing traffic among the two ports/hosts and so will upstream routers when routing traffic towards MidoNet.

The first step is to add a second router port:

```
midonet> router router0 add port addresses 10.22.22.1/24
router0:port1
midonet>
midonet> router router0 list port
port port0 device router0 state up plugged no mac ac:ca:ba:ab:ed:b8
 addresses 10.12.12.1/24
port port1 device router0 state up plugged no mac ac:ca:ba:5e:0a:02
 addresses 10.22.22.1/24
```

Now you can add a BGP peer that is reachable through the new port:

```
midonet> router router0 add bgp-peer asn 64515 address 10.22.22.2
router0:peer2
midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 10.12.12.2 keep-alive 5 hold-time 5 connect-
retry 10
peer peer1 asn 64514 address 10.12.12.3
peer peer2 asn 64515 address 10.22.22.2
midonet>
```

And bind the new port to a NIC in a different physical host:

```
midonet> host host1 add binding port router0:port1 interface eth0
host host1 interface eth0 port router0:port1
midonet>
```

At this point the MidoNet agent in host1 will bring up the new router port and set up bgpd to talk to the peer in 10.22.22.2.

As with the first port, adding a second BGP peer on the 10.22.22.0/24 network would enable host1 to load balance across two upstream routers and to keep functioning as a gateway even if one of its two BGP peers fails.

BGP failover configuration on a BGP peer

The default BGP failover time is 2-3 minutes. However, you can reduce this time by changing some parameters on both ends of the BGP session.

You must make the change in mn-conf(1) on the MidoNet side, and the remote end BGP peer configuration.

The basic BGP timers are 'keepalive' and 'holdtime'. By default, keepalive timer is 60 seconds. The hold-down timer is, by convention, 3 times the keepalive interval, 180 seconds. You can go as low as 1 and 3 seconds for these values, but keep in mind that this may potentially result in the BGP session flapping.

Another important BGP timer that you can tweak is the BGP 'connect_retry' timer, also called the 'connect' timer. You can use this timer to set the amount of time between retries to establish a connection to configured peers which have gone down for some reason.

The example below shows how to reduce the default BGP values to 1, 1, and 3 seconds, for keep-alive, hold-time, and connect-retry, respectively, on the BGP peer's session end (e.g. Quagga or Cisco):

```
neighbor 192.0.2.1 timers 1 3
neighbor 192.0.2.1 timers connect 1
```

To match those settings you would have to set the following parameters on the MidoNet end of the BGP session:

```
agent {
    midolman {
        bgp_connect_retry : 1s
        bgp_holdtime : 3s
        bgp_keepalive : 1s
    }
}
```

Static Setup

If you are not connecting through a BGP link, or you just want to use static routing follow this section.

This creates a static up-link to connect VMs to the external network.

1. Create fake uplink

   We are going to create the following topology to allow the VMs reach external networks:

```
             +---------------+
                             |
                             | 172.19.0.1/30
          +------------------+---------------+
          |                                  |
          |     Fakeuplink linux bridge      |
          |                                  |
          +------------------+---------------+        UNDERLAY
                             | veth0
                             |
                             |
                             |
  +------+  +------+  +-------------+  +------+  +------+
                             |
                             |
                             |
               172.19.0.2/30 | veth1
          +------------------+----------------+        OVERLAY
```

```
          |                                    |
          |            Edge Router             |
          |                                    |
          +------------------+-----------------+
                             |  200.200.200.0/24
                             |
          +------------------+-----------------+
```

2. Create a veth pair

```
# ip link add type veth
# ip link set dev veth0 up
# ip link set dev veth1 up
```

3. Create a bridge, set an IP address and attach veth0

```
# brctl addbr uplinkbridge
# brctl addif uplinkbridge veth0
# ip addr add 172.19.0.1/30 dev uplinkbridge
# ip link set dev uplinkbridge up
```

4. Enable IP forwarding

```
# sysctl -w net.ipv4.ip_forward=1
```

5. Route packets to 'external' network to the bridge

```
# ip route add 200.200.200.0/24 via 172.19.0.2
```

6. Create a port on the Edge Router and bind it to the veth:

```
$ midonet-cli
midonet> router list
router router0 name Edge Router state up
midonet> router router0 add port addresses 172.19.0.2/30
router0:port0
midonet> router router0 add route src 0.0.0.0/0 dst 0.0.0.0/0 type
 normal port router router0 port port0 gw 172.19.0.1
midonet> host list
host host0 name controller alive true
midonet> host host0 add binding port router router0 port port0 interface
 veth1
host host0 interface veth1 port router0:port
```

7. Add masquerading to your external interface so connections coming from the overlay with addresses that belong to the "fake" external network are NATed. Also make sure these packets can be forwarded:

```
# iptables -t nat -I POSTROUTING -o eth0 -s 200.200.200.0/24 -j
 MASQUERADE
# iptables -I FORWARD -s 200.200.200.0/24 -j ACCEPT
```

Now we can reach VMs from the underlay host using their floating IPs, and VMs can reach external networks as well (as long as the host has external connectivity).

2. Authentication and authorization

Table of Contents

The MidoNet Application Programming Interface (API) provides its authentication and authorization services by integrating with external identity services.

It does not provide an API to create or delete tenants, but it does have an API to tag resources and filter queries with tenant IDs. Tenants are managed completely by an external identity service, and where appropriate, the string representations of tenant IDs are sent in the requests to the MidoNet API. This design makes possible a federated identity service model where one identity service provides authentication and authorization to all the services in the cloud environment, including the MidoNet API.

Although the MidoNet Cluster does not provide its own identity service, it does provide simple authentication (for validating the user) and authorization (for checking the access level of the user) functionality. For authentication, it simply takes the token included in the HTTP header and forwards it to the external identity service. To generate a new token, it also provides an API to log in to the external identity service with username/password credentials. (See the MidoNet REST API document for more information on the token.) For authorization, the MidoNet API provides a very simple Role-Based Access Control (RBAC) mechanism explained in the next section.

See the MidoNet REST API document for information on implementing authentication and authorization using the MidoNet API. Go to http://docs.openstack.org/ and follow the links to Documentation and API for information about the OpenStack API.

Available authentication services in MidoNet

The MidoNet Cluster package includes two authentication providers: *Keystone* authentication and *mock* authentication. You can select the current authentication provider by modifying the value of the `cluster.auth.provider_class` key using `mn-conf` to one of the following values:

Table 2.1. Authentication Provider Classes

Class	Description
`org.midonet.cluster.auth.keystone.KeystoneService`	Uses external Keystone identity service, version 2 or version 3.
`org.midonet.cluster.auth.MockAuthService`	Disables authentication.

This section describes the Keystone authentication service, the mock authentication and some additional configurations options specific to each provider.

Keystone authentication

In order to use the OpenStack Keystone authentication service with MidoNet, you must configure the `KeystoneService` provider class:

```
echo "cluster.auth.provider_class : org.midonet.cluster.auth.keystone.
KeystoneService" | mn-conf set -t default
```

Note

MidoNet version 5.0.2 introduced support for Keystone version 3, and a new authentication provider class `org.midonet.cluster.auth.keystone.KeystoneService`, which replaces the previous `org.midonet.cluster.auth.keystone.v2_0.KeystoneService`. However, the legacy provider is kept for compatibility reasons and provides the same capabilities as the new Keystone authentication provider, supporting both Keystone version 2 and 3.

Important

Since MidoNet version 5.0.2, the default Keystone authentication is Keystone version 3. If you are upgrading from a previous MidoNet version, you must modify the Keystone configuration in order to continue using Keystone version 2. See the section called “Enabling Keystone authentication” [10].

Important

You must restart all MidoNet Cluster instances after changing the authentication provider.

To enable Keystone authentication, you must configure the MidoNet cluster with the credentials of an administrative user. The cluster uses this credentials to authenticate third-party requests to the MidoNet API. MidoNet supports two Keystone authentication methods:

- *Password authentication*, where you configure the user name and password credentials of an administrative user.
- *Token authentication*, where you configure the administrative token used by Keystone.

Important

The token authentication takes precedence over the password authentication.

For additional Keystone configuration options, see the section called “Enabling Keystone authentication” [10]

Mock authentication

Mock authentication disables the authentication system by returning fake tokens to authenticating clients, and ignoring the sent tokens during authorization. To enable the mock authentication configure the `MockAuthService` provider class.

```
echo "cluster.auth.provider_class : org.midonet.cluster.auth.
MockAuthService" | mn-conf set -t default
```

Warning

Mock authentication is the default authentication provider for the MidoNet Cluster. However, this mode is used for testing purposes but should not be used in production.

Using the Keystone authentication service

This section explains how to use the Keystone authentication service with MidoNet.

Enabling Keystone authentication

In order to use the OpenStack Keystone authentication service with MidoNet, you must configure the following keys in the MidoNet configuration.

cluster.auth.provider_class

The fully qualified path of the Java class that provides the Keystone authentication service.

```
org.midonet.cluster.auth.keystone.KeystoneService
```

Note

Since MidoNet version 5.0.2, the legacy authentication provider `org.midonet.cluster.auth.keystone.v2_0.KeystoneService` is available for compatibility reasons, but it provides the same capabilities as the new provider, supporting both Keystone version 2 and 3. The default Keystone version is 3.

cluster.auth.admin_role

Identifies the name of the admin role in MidoNet. The admin has read and write access to all the resources. We recommend re-using the OpenStack *admin* role. Optionally you can create a separate admin role for MidoNet.

cluster.auth.keystone.version

The version of the Keystone API. The following values are allowed:

Table 2.2. Keystone Version

Version	Description
2	Uses the Keystone version 2 API.
3	Uses the Keystone version 3 API (default).

cluster.auth.keystone.protocol

Identifies the protocol used for the Keystone service. The following values are allowed:

Table 2.3. Keystone Protocol

Class	Description
`http`	Uses plain text HTTP to communicate with the Keystone service (default).
`https`	Uses encrypted HTTPS to communicate with the Keystone service (recommended).

cluster.auth.keystone.host

Identifies the host of the Keystone service (default is *localhost*).

cluster.auth.keystone.port

Identifies the port number of the Keystone service (default is *35357*).

cluster.auth.keystone.user_name

The user name to authenticate with Keystone when using password authentication.

cluster.auth.keystone.user_password

The password to authenticate with Keystone when using password authentication.

cluster.auth.keystone.admin_token

The administrative token to authenticate with Keystone when using token authentication.

cluster.auth.keystone.tenant_name

Specifies the name of the tenant that is used when logging into Keystone. The log-in authentication to Keystone requires the username, password, and tenant name of the user. By specifying the tenant name here, you can avoid the need for applications to supply the tenant name when logging into Keystone through the MidoNet API.

cluster.auth.keystone.domain_name

For Keystone version 3, specifies the name of the domain that is used when logging in to Keystone.

Disabling Keystone authentication

MidoNet lets you disable authentication by using a mock authentication service.

Using this service has the effect that no outside authentication service is used. Instead, MidoNet will return fake tokens to the authenticating clients. Likewise, the provider will ignore any tokens sent by the client when authorizing an API request.

To use the mock authentication service, you must configure the following keys in the MidoNet configuration.

cluster.auth.provider_class

The fully qualified path of the Java class that provides the mock authentication service.

```
org.midonet.cluster.auth.MockAuthService
```

3. Admitting resources to MidoNet

Table of Contents

When started, the MidoNet Agent (midolman) automatically connects to the ZooKeeper database and registers itself as an available host. You need to admit that host into a tunnel zone so the host can communicate within the tunnel zone to other hosts.

You can think of this as registering the host. When you admit a host, you select the physical interface the host will use for communication with other hosts. This interface must be connected to the underlay network. After you select the interface, you bind it to the tunnel zone. This allows the host to establish tunnels with all the other hosts in the tunnel zone via the selected interface.

What are tunnel zones?

A tunnel zone is an isolation zone for hosts.

Physical hosts in the same tunnel zone, communicate directly with one another, without a need to use a tunnel. MidoNet supports two types of tunnel zones for separating physical hosts in the underlay, GRE (default) and VXLAN.

A third type of a tunnel zone, VTEP, is used for connecting MidoNet hosts to a hardware VTEP. Whereas, a MidoNet host may only belong to one tunnel zone, type GRE or VXLAN, it may belong to two different tunnel zones if one of them is of GRE/VXLAN type and the other one of VTEP type. In fact, this configuration is required to successfully connect a MidoNet host to a hardware VTEP. For more information, see "VXLAN configuration" section in the "MidoNet Operations Guide".

Creating tunnel zones

This section describes how to create tunnel zones.

1. Enter the `create tunnel-zone name tz-name type tz-type` command to create a new tunnel zone, where *tz-type* is tunnel zone type, gre, vxlan, or vtep. For example:

   ```
   midonet> create tunnel-zone name new-tz type vxlan
   tzone0
   ```

   Where: new-tz = the name you want to assign to the tunnel zone; the output shows the alias ("tzone0") assigned to the tunnel zone

2. Enter the `list tunnel-zone` command to list and confirm the tunnel zone. For example:

   ```
   midonet> list tunnel-zone
   tzone tzone0 name new-tz type vxlan
   tzone tzone1 name vxlan type vxlan
   ```

Deleting tunnel zones

Use this procedure to delete a tunnel zone.

1. Enter the `list tunnel-zone` command to list the tunnel zones. For example:

```
midonet> list tunnel-zone
tzone tzone0 name new-tz type vxlan
tzone tzone1 name vxlan type vxlan
```

2. Enter the `delete tunnel-zone tz-alias` command to delete the desired tunnel zone. For example:

```
midonet> delete tunnel-zone tzone0
```

   Specify the dynamically assigned number of the alias for the tunnel zone to delete; in the above example, the assigned number is 0 (tzone0).

3. (Optional) Enter the command to list the tunnel zones to confirm the deletion:

```
midonet> list tunnel-zone
tzone tzone1 name vxlan type vxlan
```

Viewing tunnel zone information

Use this procedure to view tunnel zone information.

```
midonet> tunnel-zone tzone0 list member
zone tzone0 host host0 address 192.168.0.3
zone tzone0 host host1 address 192.168.0.5
zone tzone0 host host2 address 192.168.0.4
zone tzone0 host host3 address 192.168.0.6
```

The above output shows the:

- Aliases for the hosts in the tunnel zone (host0, host1, and so on)
- IP addresses assigned to the hosts

Working with hosts

This section shows how to view host information and admit new hosts to a tunnel zone.

Viewing host information

Use this procedure to view information about hosts.

- To list the hosts enter the command:

```
midonet> list host
host host0 name controller alive true
host host2 name compute1 alive true
host host3 name compute3 alive false
host host1 name compute2 alive false
```

- To list the interfaces on a certain host enter the command:

```
midonet> host host0 list interface
iface midonet host_id host0 status 0 addresses [] mac 12:6e:b7:d0:4f:f1
 mtu 1500 type Virtual endpoint DATAPATH
```

```
iface lo host_id host0 status 3 addresses [u'127.0.0.1',
 u'0:0:0:0:0:0:0:1'] mac 00:00:00:00:00:00 mtu 65536 type Virtual
 endpoint LOCALHOST
iface tapbf954474-ef host_id host0 status 3 addresses
 [u'fe80:0:0:0:dc40:9aff:feef:7b5e'] mac de:40:9a:ef:7b:5e mtu 1500 type
 Virtual endpoint DATAPATH
iface eth0 host_id host0 status 3 addresses [u'192.168.0.3',
 u'fe80:0:0:0:f816:3eff:febe:590'] mac fa:16:3e:be:05:90 mtu 8842 type
 Physical endpoint PHYSICAL
```

- To show the port binding for a certain host enter the command:

```
midonet> host host0 list binding
host host0 interface tapbf954474-ef port bridge0:port0
```

The above output shows that device tap tapbf954474-ef on host0 is currently connected to port0 on bridge0.

Admitting a host

You need to add new hosts to a tunnel zone. Use this procedure to admit a host into a tunnel zone.

1. To view a list of all tunnel zones, enter the `list tunnel-zone` command. For example:

```
midonet> list tunnel-zone
tzone tzone0 name gre type gre
```

2. To view the list of all hosts, enter the `list host` command. For example:

```
midonet> list host
host host0 name compute-1 alive true
host host1 name compute-2 alive true
```

3. To list all interfaces on a host, enter the `host hostX list interface` command (where: X is the dynamically assigned number for the desired host alias).

```
midonet> host host0 list interface
iface lo host_id host0 status 3 addresses [u'127.0.0.1',
 u'0:0:0:0:0:0:0:1'] mac 00:00:00:00:00:00 mtu 65536 type Virtual
 endpoint LOCALHOST
iface midonet host_id host0 status 0 addresses [] mac 8e:4d:60:c1:70:d7
 mtu 1500 type Virtual endpoint DATAPATH
iface eth1 host_id host0 status 3 addresses
 [u'fe80:0:0:0:250:56ff:fe93:7c35'] mac 00:50:56:93:7c:35 mtu 1500 type
 Physical endpoint PHYSICAL
iface eth0 host_id host0 status 3 addresses [u'10.1.2.200',
 u'fe80:0:0:0:250:56ff:fe93:c9a4'] mac 00:50:56:93:c9:a4 mtu 1500 type
 Physical endpoint PHYSICAL
```

Note the IP address of the interface that will be used for tunnel creation.

4. Enter the `tunnel-zone tzone add member host host` command to add a new member (host) to a tunnel zone. For example:

```
midonet> tunnel-zone tzone0 add member host host0
address host zone
```

5. Enter the `tunnel-zone tzone add member host host address ip-address` command to specify the IP address of the new member. For example:

```
midonet> tunnel-zone tzone0 add member host host0 address 10.1.2.200
```

In the above command example:

- tzone0 = the tunnel zone you want to add the member (host) to
- host0 = the alias of the host you want to add
- 10.1.2.200 = the IP address of the host you want to add

Removing a host from a tunnel zone

Use this procedure to remove a host from a tunnel zone.

1. Enter the `list tunnel-zone` command to list the tunnel zone. For example:

```
midonet> list tunnel-zone
tzone tzone0 name default_tz type gre
```

2. Enter the `tunnel-zone tunnel-zone list member` command to list the tunnel zone members (hosts). For example:

```
midonet> tunnel-zone tzone0 list member
zone tzone0 host host0 address 172.19.0.2
```

3. Enter the `tunnel-zone tunnel-zone member host host show` command to show information about a specific host. For example:

```
midonet> tunnel-zone tzone0 member host host0 show
tunnel-zone-host zone tzone0 host host0 address 172.19.0.2
```

4. Enter the `tunnel-zone tunnel-zone member host host delete` command to delete the desired host (identified by the host's alias). For example:

```
midonet> tunnel-zone tzone0 member host host0 delete
```

5. (Optional) You can add the host back to the tunnel zone, using the `tunnel-zone tunnel-zone member add host host address ip-address` command as shown below:

```
midonet> tunnel-zone tzone0 member add host host0 address 172.19.0.2
zone tzone0 host host0 address 172.19.0.2
```

Removing a host

Use this procedure to remove inactive hosts.

1. Enter the command to list the hosts:

```
midonet> list host
host host0 name precise64 alive true
```

2. Enter the command to delete the desired host (identified by its alias):

```
midonet> host host0 delete
```

4. Device abstractions

Table of Contents

The major resources in MidoNet's virtual network model are routers, bridges, ports, load balancers, VTEPs, chains, and rules.

Routers, bridges, and ports are devices that comprise a virtual network topology and chains and rules are policies applied to the network elements.

MidoNet lets you create routers and bridges.

Creating a router

You can create routers to provide L3 connectivity on a virtual network.

To create a router:

1. Enter the following command to list the routers on the current tenant:

```
midonet> list router
midonet>
```

   The above example shows no routers on the current tenant.

2. Enter the command to create a new router and assign it a name:

```
midonet> router create name test-router
router1
```

   The above command example shows how to create a new router named "test-router". The output shows the alias "router1" assigned to the new router.

Adding a port to a router

You need to add a port to a router to connect the router to a bridge (network) or another router.

Note

When using the MidoNet CLI, if necessary, use the sett command or some other means to access the desired tenant.

1. Enter the command to add a port to the desired router and specify the desired IP address and network for the port:

```
midonet> router router1 add port addresses 10.100.1.1/24
```

```
router1:port0
```

The above output shows the alias ("port0") assigned to the new port.

2. Enter the command to list port information for the router:

```
midonet> router router1 list port
port port0 device router1 state up mac 02:a6:81:08:ab:5d addresses 10.
100.1.1/24
```

The above output shows the:

- Alias assigned to the port ("port0")
- Device the port is attached to (router1)
- Port's state (up)
- Port's MAC address
- Port's IP and network prefix length

Adding a bridge

Use this procedure to create a bridge.

Enter the following command to create a bridge and assign it a name:

```
midonet> bridge create name test-bridge
bridge1
```

The above output shows the alias ("bridge1") assigned to the new bridge.

Adding a port to a bridge

Use this procedure to add a port to a bridge.

1. Enter the command to list the bridges on the current tenant:

```
midonet> bridge list
bridge bridge1 name test-bridge state up
```

2. Enter the command to add a port to the desired bridge:

```
midonet> bridge bridge1 add port
bridge1:port0
```

The above output shows the alias ("port0") assigned to the new port.

Binding an exterior port to a host

In order to connect a MidoNet-enabled cloud to an external network, you need to bind an exterior port to a host, such as a network interface card (NIC) with an ID of eth0, for example.

1. Enter the command to list the hosts:

```
midonet> list host
host host0 name compute-1 alive true
host host1 name compute-2 alive true
```

2. Enter the command to list the bridges on the current tenant:

```
midonet> list bridge
bridge bridge0 name External state up
bridge bridge1 name Management state up
bridge bridge2 name Internal state up
```

3. Enter the command to list the ports on the desired bridge:

```
midonet> bridge bridge0 list port
port port0 device bridge0 state up
port port1 device bridge0 state up
port port2 device bridge0 state up
```

4. Enter the command to list the interfaces on a certain host:

```
midonet> host host0 list interface
iface lo host_id host0 status 3 addresses [u'127.0.0.1',
 u'0:0:0:0:0:0:0:1'] mac 00:00:00:00:00:00 mtu 65536 type Virtual
 endpoint LOCALHOST
iface midonet host_id host0 status 0 addresses [] mac 8e:4d:60:c1:70:d7
 mtu 1500 type Virtual endpoint DATAPATH
iface eth1 host_id host0 status 3 addresses
 [u'fe80:0:0:0:250:56ff:fe93:7c35'] mac 00:50:56:93:7c:35 mtu 1500 type
 Physical endpoint PHYSICAL
iface eth0 host_id host0 status 3 addresses [u'10.1.2.200',
 u'fe80:0:0:0:250:56ff:fe93:c9a4'] mac 00:50:56:93:c9:a4 mtu 1500 type
 Physical endpoint PHYSICAL
```

5. Enter the command to bind a certain host to virtual port:

```
midonet> host host0 add binding
host interface port
```

6. Enter the command to bind a virtual port on the bridge with a physical interface on the host:

```
midonet> host host0 add binding port bridge0:port0 interface eth1
host host0 interface eth1 port bridge0:port0
```

Stateful port groups

MidoNet features stateful port groups, which are groups of virtual ports (typically two) that are logically associated, usually to perform load balancing or for link redundancy.

For such ports MidoNet keeps state local to the two endpoints of a connection. In most cases, connections that traverse MidoNet do so between a single pair of ports. Typical cases include a router with two uplink BGP ports, or an L2GW with two ports connected to a physical L2 network. In both cases, the pair of ports becomes a set of ports because packets may return through different paths. Those port pairs will share state.

You configure stateful port groups in the MidoNet CLI, using the port-group command.

Creating stateful port groups

Follow the steps of this procedure to create a stateful group of ports, using the MidoNet CLI.

Before you launch the MidoNet CLI you need to find out the OpenStack UUID of the tenant on which you want to create your port group. To this end, you can use keystone. Issue the following commands in the terminal on the MidoNet host:

```
# keystone tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| 7a4937fa604a425e867f085427cc351e |  admin   |   True  |
| 037b382a5706483a822d0f7b3b2a9555 | alt_demo |   True  |
| 0a1bf57198074c779894776a9d002146 |   demo   |   True  |
| 28c40ac757e746f08747cdb32a83c40b | services |   True  |
+----------------------------------+----------+---------+
```

The output of the command shows the full list of tenants. For this procedure we will use the 'admin' tenant, 7a4937fa604a425e867f085427cc351e.

1. In the MidoNet CLI determine the list of available routers.

   ```
   midonet> list router
   router router0 name Edge Router state up
   router router1 name TenantRouter state up
   ```

   Let’s assume that the router whose ports you are going add to the port group is Edge Router, router0.

2. Now list the ports on router0.

   ```
   midonet> router router0 list port
   port port0 device router0 state up mac 02:c2:0f:b0:f2:68 addresses 100.
   100.100.1/30
   port port1 device router0 state up mac 02:cb:3d:85:89:2a addresses 172.
   168.0.1/16
   port port2 device router0 state up mac 02:46:87:89:49:41 addresses 200.
   200.200.1/24 peer bridge0:port0
   port port3 device router0 state up mac 02:6b:9f:0d:c4:a8 addresses 169.
   254.255.1/30
   ```

   You want to add port0 and port1 on the router to load balance the BGP traffic on the Edge Router.

3. Load your tenant using the 'sett' command.

   ```
   midonet-cli> sett 7a4937fa604a425e867f085427cc351e
   tenant_id: 7a4937fa604a425e867f085427cc351e
   ```

4. Create a stateful port group using the 'port-group create' command.

   ```
   midonet-cli> port-group create name SPG stateful true
   pgroup0
   ```

5. Add the two ports on the Edge Router that you want to participate in load balancing, to the port group you just created.

   ```
   midonet> port-group pgroup0 add member port router0:port0
   port-group pgroup0 port router0:port0
   midonet> port-group pgroup0 add member port router0:port1
   port-group pgroup0 port router0:port1
   ```

You have successfully added both router ports to the stateful port group, which you can verify by issuing the following command:

```
midonet> port-group pgroup0 list member
port-group pgroup0 port router0:port1
port-group pgroup0 port router0:port0
```

5. Connecting devices

Table of Contents

You can create a virtual topology by connecting routers to bridges, to switches, and to other routers.

Connecting a bridge to a router

You can easily connect a virtual bridge to a virtual router via virtual ports on the two devices. Make sure you create a bridge and a router with an Interior port on each device.

Note

See the section called "Creating a router" [16] and the section called "Adding a bridge" [17] for information about creating routers and bridges and adding router and bridge ports.

To connect a bridge to a router:

1. Enter the command to list the bridges on the current tenant:

```
midonet> list bridge
bridge bridge1 name test-bridge state up
```

2. Enter the command to list the ports on the bridge:

```
midonet> bridge bridge1 list port
port port0 device bridge1 state up
```

3. Enter the command to list the routers on the current tenant:

```
midonet> list router
router router1 name test-router state up
```

4. Enter the command to list the ports on the router:

```
midonet> router router1 list port
port port0 device router1 state up mac 02:a6:81:08:ab:5d addresses 10.
100.1.1/24
```

5. Enter the command to bind the desired router port (for example, port0 on router1) to the desired bridge port (for example, port0 on bridge1):

```
midonet> router router1 port port0 set peer bridge1:port0
```

6. Enter the command to list the ports on the router (in this example, router1):

```
midonet> router router1 list port
port port0 device router1 state up mac 02:a6:81:08:ab:5d addresses 10.
100.1.1/24 peer bridge1:port0
```

The above output shows that port0 on router1 is connected to port0 on bridge1.

Connecting two routers

You can easily connect two virtual routers via virtual ports on each router.

Make sure you create the router ports on the two routers and assign the ports to the same subnet. See Chapter 4, “Device abstractions” [16] for information about creating routers and adding router ports.

To connect two routers:

1. Enter the command to list the routers on the current tenant:

```
midonet> list router
router router3 name test-router2 state up
router router1 name test-router state up
```

2. Enter the command to list the ports on one of the routers you want to connect:

```
midonet> router router1 list port
port port0 device router1 state up mac 02:a6:81:08:ab:5d addresses 10.
100.1.1/24 peer bridge1:port0
port port1 device router1 state up mac 02:fa:5f:87:bb:d2 addresses 10.
100.1.2/24
```

3. Enter the command to list the ports on the router you want to connect it to:

```
midonet> router router3 list port
port port0 device router3 state up mac 02:df:24:5b:19:9b addresses 10.
100.1.128/24
```

4. Enter the command to bind the port on one router (for example, port 1 on router1) to the port on another router (for example, port0 on router3):

```
midonet> router router1 port port1 set peer router3:port0
```

5. Enter the command to list the ports on one of the routers:

```
midonet> router router1 list port
port port0 device router1 state up mac 02:a6:81:08:ab:5d addresses 10.
100.1.1/24 peer bridge1:port0
port port1 device router1 state up mac 02:fa:5f:87:bb:d2 addresses 10.
100.1.2/24 peer router3:port0
```

The above output shows that port1 on router1 is connected to port0 on router3.

6. Routing

Table of Contents

Routing in a MidoNet-enabled network works the same as on physical routers. Virtual routers route packets to locally connected networks and hosts directly connected to it and forward packets to gateways for delivery to external networks.

Routing process overview

MidoNet selects a route by:

1. Find the matching routes whose destination prefix mask is longest (longest prefix match, LPM)

2. Filter for routes whose source matches the packet.

3. Filter for route(s) that has/have the highest weight.

4. Perform ECMP calculation (hash mod N) to select 1 of the routes.

For traffic that ingresses a router port, the router does the following:

- Attempts to match a given packet's destination with a route in its routing table based on the packet's source and destination, preferring the route with the lower weight value.

- For packets with more than one matching route, uses per-agent round robin selection among those routes.

- Drops packets that don't match routes with matching sources and destinations.

The router uses information in the following fields in the routing table:

- Source: defines the IP addresses and networks to filter and evaluate for delivery to directly connected networks and hosts or forward to gateways to other networks.

- Route Type: determines the action to take on the packet.

- Next Hop Port and Next Hop Gateway: determine where to route the packet.

Below are two sample routes on a tenant router (displayed using the MidoNet CLI):

```
route route2 type normal src 0.0.0.0/0 dst 172.16.3.0/24 port router0:port1
 weight 100
route route3 type normal src 172.16.3.0/24 dst 169.254.169.254 gw 172.16.3.
2 port router0:port1 weight 100
```

Source

Shows the route's source prefix for source-based routing. The algorithm to decide which route applies to the packet is briefly as follows:

1. Disregard all routes whose source prefix does not match the packet's source. The source prefix, 0.0.0.0/0, matches everything. Note that /0 denotes a bit mask of length zero, so the address 0.0.0.0 is ignored.
2. Find the route whose destination prefix matches the packet's destination and has the longest mask (a.k.a. longest-prefix matching).
3. If there's more than one route candidate (with equal destination prefix mask length and weight), use per-agent round robin to select a route.

Type

There are three types: Normal, Blackhole, and Reject.

- Normal: normal type of route for forwarding packets. Use Next Hop Gateway and Next Hop Port information to route packets.
- Blackhole: tells the router that when a packet matches this route, the packet should be dropped, without sending any notification. If no floating IP addresses exist on an external network, traffic is routed to a Blackhole, where the traffic is dropped.
- Reject: similar to Blackhole, except when a packet matches this route, the router returns an ICMP error (MidoNet only sends the error upon receiving the first packet of the flow, or if/when the flow is recomputed). The error is Type 3 (indicates the destination port is unreachable), the code is either Code 9 or Code 10 (the destination host/network is administratively prohibited), depending on whether the route's destination prefix has a mask of 32 bits (that is, a specific host = Code 10) or has a mask less than 32-bits (that is, a network = Code 9). For more information, see http://en.wikipedia.org/wiki/ICMP_Destination_Unreachable#Destination_unreachable

Destination

Shows the IP address of the interface connected to a peer device (for example, router or bridge). This is the egress port for packets sent to the destination peer.

Next Hop Gateway

The route has three options regarding what to do with a packet:

1. Drop the packet. In this case, there's no need for a next hop gateway.
2. Forward the packet directly to the destination; this only occurs if the destination is on the same L2 network as one of the router's ports. Normally this means that the packet's destination address is in the same network prefix as the router's port. Again, there's no need for a next hop gateway in this case.
3. Forward the packet towards the destination, but using an intermediary router. Such a route is known as the "next hop gateway."

The next hop gateway is the IP address of the intermediate router's port that's facing the router that owns this route. This IP address is only used to do ARP resolution (mapping the IP address to a MAC address) and to determine how to re-write the packet's destination MAC address before emitting it towards the intermediate router.

Note that routes in normal (that is, physical) routers differ from MidoNet's virtual routers in that they don't have a destination virtual port (the 'Normal' route in MidoNet has this; see "Type"). Therefore, normal routers use the Next Hop Gateway IP address to determine which port should be used to emit the packet (the port whose prefix matches the next hop gateway's IP address).

Next Hop Port

Shows the ID of the port connected to the peer device.

Weight

Can be used for load balancing for destinations with multiple paths. Lower weight values identify preferred paths (for example, higher bandwidth). The default weight value is 100. For routes learned from BGP peers, the BGP administrative distance becomes the route's weight. See also "Source".

Viewing routes

You can view the routes defined on each virtual router in MidoNet. For example, you can view information about routes to virtual bridges and to other routers, including tenant routers and Edge Routers.

To display route information about the current tenant's routers:

1. Enter the command below to list the routers for the current tenant.

```
midonet> list router
router router0 name tenant-router state up infilter chain0 outfilter
 chain1
```

2. Enter the command below to list the route list for the router, in this case, a tenant router (router0).

```
midonet> router router0 list route
route route0 type normal src 0.0.0.0/0 dst 169.254.255.2 port
 router0:port0 weight 0
route route1 type normal src 0.0.0.0/0 dst 0.0.0.0/0 port router0:port0
 weight 100
route route2 type normal src 0.0.0.0/0 dst 172.16.3.0/24 port
 router0:port1 weight 100
route route3 type normal src 172.16.3.0/24 dst 169.254.169.254 gw 172.
16.3.2 port router0:port1 weight 100
route route4 type normal src 0.0.0.0/0 dst 172.16.3.1 port router0:port1
 weight 0
```

The route list shows the following information:

- The source (src) for the traffic to match. route3 shows a specific source network to match; 0.0.0.0/0 means match traffic from every network.
- The destination (dst) for this traffic. This can be a network or a specific interface.
  - route0 shows an example of a route to a specific interface. This is the route to the link-local address, which, in this example, is peered with an Edge Router.

  - route2 shows a route to the 172.16.3.0/24 network, which is a private network.

  - route1 says: for traffic that matches every network (0.0.0.0/0) with a destination of any network (0.0.0.0/0), forward this traffic to port0, which is peered with an Edge Router. For traffic that matches the source IP address, MidoNet finds the route whose destination prefix matches the packet's destination and has the longest mask (a.k.a. longest-prefix matching). If the router cannot find a destination with a longer prefix than 0.0.0.0, the router sends the traffic to this default route.

- For destinations that are not directly connected to the router, the interface to the gateway to the destination is shown (next hop gateway).

  - route3 shows an example. This route says: traffic with a source that matches the 172.16.3.0/24 network (that is, traffic from the private network), with the destination, 169.254.169.254, which is the link-local address to the metadata service, forward this traffic to the gateway port, 172.16.3.2, which is the tenant router's interface to the metadata service. This kind of routes exist only when the network is served by Neutron DHCP Agent, typically to provide instance metadata service.

  - route4 says: for traffic from any network (0.0.0.0/0) with the destination of the tenant router's interface (172.16.3.1), which is the tenant router's interface to the private network, forward this traffic to port1. port 1 on the tenant router is peered with port0 on router1, which is the Edge Router.

Adding routes

Below are some examples of when you might want to add a route:

- You notice that a specific IP address or range is attacking your network. To prevent such attacks, you add a route to the Edge Router; this route's source IP matches the attacking IP address or range. You specify the type as a Blackhole to configure the Edge Router to drop packets from this source.

- If BGP dynamic routing is not available, you may configure static routes to forward traffic to the upstream router(s).

The attributes you specify are:

- dst = destination IP address or network to match

- src = source IP address or network to match

- type = for example, "normal"

- port = port to emit traffic over

To add a route using the MidoNet CLI:

1. Enter the command to add a route:

   ```
   midonet> router router2 add route dst 169.254.255.0/30 src 0.0.0.0/0
    type normal port router2:port2
   router2:route2
   ```

   The above command contains the following instructions:

   - For traffic with the source any network (0.0.0.0/0) with the destination network 169.254.255.0/30, forward this traffic over port2 on router2.

Note

Prior to adding the above route, a port was added to router2 using the following command:

```
midonet> router router2 add port addresses 169.254.255.3/30
router2:port2
```

2. Enter the command to list the routes on router2 to confirm the added route(s):

```
midonet> router router2 list route
route route0 type normal src 0.0.0.0/0 dst 10.100.1.1 port router2:port0
 weight 0
route route1 type normal src 0.0.0.0/0 dst 10.100.1.2 port router2:port1
 weight 0
route route2 type normal src 0.0.0.0/0 dst 169.254.255.3 port
 router2:port2 weight 0
route route3 type normal src 0.0.0.0/0 dst 169.254.255.0/30 port
 router2:port2 weight 0
```

Deleting routes

If you are using MidoNet as a standalone SDN controller, there are many situations where you might want to delete routes; all related to managing your physical network devices.

For example, if you want to reverse something you did that required manually adding routes, you can delete the routes.

Warning

It is not recommended to delete routes that were added automatically as a result of OpenStack Neutron operations.

To delete a route:

1. Enter the command to list the routes on a certain router:

```
midonet> router router2 list route
route route0 type normal src 0.0.0.0/0 dst 10.100.1.1 port router2:port0
 weight 0
route route1 type normal src 0.0.0.0/0 dst 10.100.1.2 port router2:port1
 weight 0
route route2 type normal src 0.0.0.0/0 dst 169.254.255.3 port
 router2:port2 weight 0
route route3 type normal src 0.0.0.0/0 dst 169.254.255.0/30 port
 router2:port2 weight 0
```

   The above command lists the routes on router2.

2. Enter the command(s) to delete the desired route(s) from the desired router:

```
midonet> router router2 delete route route2
midonet> router router2 delete route route3
```

   The above commands delete route2 and route3 from router2.

3. Enter the command to list the routes on the router to confirm the deletions:

```
midonet> router router2 list route
```

```
route route0 type normal src 0.0.0.0/0 dst 10.100.1.1 port router2:port0
 weight 0
route route1 type normal src 0.0.0.0/0 dst 10.100.1.2 port router2:port1
 weight 0
```

ECMP Limitations

Limitations of MidoNet ECMP for load-balancing to a set of stateful services:

- ECMP route selection is "flow-hash mod N"
- MidoNet does not test that the routes are ordered the same in every Agent so that "hash mod N" will return the same result on different L3 Gateways as long as N does not change.
- The hash algorithm does not satisfy "consistent hashing" properties, in particular, it does not minimise the impact of changes to the load-balanced set. Therefore, when the load-balanced set changes, any active flow that is recomputed (because it lasts more than 30 seconds or because upstream routing causes it to ingress a new gateway) will be balanced to a different back-end, usually leading to a broken connection.
  - Note that MidoNet's DNAT rules as well as L4 LBaaS implementation do generate per-flow state so those features are more resilient to backend set changes.
- The hash algorithm is not symmetric, so return packets are unlikely to traverse the same VNF instance.
  - This matters for stateful services that need to see traffic in both directions. Some stateful services can live without. And some stateful service instances forward traffic with their own address and therefore receive the return traffic without ECMP configuration.
- MidoNet's L3 Gateway makes use of ECMP for choosing an egress gateway for packets heading out of the logical network (to the rest of the data center or to the Internet). That use-case works fine because it is load-balancing to a set of stateless upstream routers. There is no harm to suddenly sending packets in a flow out of a different gateway node (apart from the possibility of some packets arriving out of order at the destination endpoint).

Despite the limitations discussed above, MidoNet's ECMP works pretty well for a number of use-cases including load-balancing a VIP to an Active-Active HA set of L7 load-balancers.

Remember that you can program MidoNet's virtual router's tables using either MidoNet's native API or via the Neutron ExtraRoute API extension.

7. Rule chains

Table of Contents

The routing logic for a MidoNet virtual router includes the traversal of rule chains that may modify, record state, or reject a packet.

A newly created router has two chains: one named 'pre-routing' and the other named 'post-routing'. Below are a few important facts about rule chains:

- You can put chains not only on routers, but also on bridges and ports, too.
- Pre-routing chains have access to the ingress port ID, while post-routing chains have access to both the ingress and egress port. During post-routing, the router routes the packet and both the ingress port and the egress router port are known.

A packet’s flow within a router

In order to understand rule chains, first consider how a router processes a packet:

1. Check if the packet is an ARP packet, if so handle it.
2. Look at the interface’s pre-routing chain, and call the chain that processes packets on ingress ports.
3. The rule chain processes the packet by invoking its rules in turn and potentially calling chains specified by jump rules. The chain returns two values of interest:
    - Next action:
        - DROP (the packet is silently dropped)
        - REJECT (the packet is dropped and the router sends an ICMP error message)
        - ACCEPT/CONTINUE (the router continues to process the packet for forwarding)
    - New packet:
        - the rule chain may have modified the packet’s headers, e.g. for port masquerading. Changes to packet headers, like for port masquerading and NAT, are applied to all packets in the flow.
4. If the router didn’t drop the packet, the router now invokes the routing logic that either decides to drop the packet or chooses an egress port and the next hop IP address for the packet.

5. Look at the post-routing rule chain and call the chain that processes packets on ingress and egress ports.

6. The post-routing rule chain outputs a next_action and a new_packet just like for pre-routing.

7. If the router didn't drop the packet, the router:

   - Performs an ARP lookup of the next-hop IP address.

   - Rewrites the destination hardware address and emits the packet from the egress port. if the egress port is a logical port, then the peer virtual router's logic is invoked and the flow restarts at step 1, but with a different router.

A packet's flow within a rule chain

A rule chain is invoked on a packet one rule at a time.

Every rule has a condition: a condition is a set of attributes to match on the packet – this is explained more below – and the packet is checked against this condition. If the packet does not meet the condition, control returns to the rule chain, which either invokes the next rule (if there is one), or returns to its caller. Note that the rule chain's caller is not necessarily the router – it may be another rule chain, invoked by a jump rule, described below. If the packet does match the rule's condition, then what happens to the packet depends on the rule's type. For a simple rule type, such as a DROP rule, the rule returns a next_action DROP to its chain, which then returns DROP to its caller, and so on, until the router's invocation of the initial (pre- or post-routing) rule chain returns and the router acts on the DROP next_action, as described in the previous section.

The invocation of a rule looks just like the invocation of a rule chain in the previous section:

- If the rule is invoked in pre-routing: process the rule on a packet on an ingress port.

- If the rule is invoked in post-routing: process the rule on a packet on an ingress, egress port.

These invocations return (next_action, new_packet) just as described for rule chains. However, the set of valid next_actions is different and is intended to instruct the rule chain on how to continue processing the flow. The valid next_actions are:

- ACCEPT: The rule chain should stop processing the packet and return (ACCEPT, new_packet) to its own caller.

- CONTINUE: The rule chain should invoke the next rule in the chain with the packet emitted by the rule. If there is no next rule, the rule chain should return (CONTINUE, new_packet) to its caller.

- DROP: The rule chain should stop processing the packet and return (DROP, new_packet) to its own caller.

- RETURN: The rule chain should stop processing the packet and return (CONTINUE, new_packet) to its own caller. Note that this is distinct from both the ACCEPT and CONTINUE actions because no more rules in this chain will be executed, but more rules in calling chains may be executed.

- REJECT: The rule chain should stop processing the packet and return (REJECT, new_packet) to its own caller.

The ways packets are handled by specific rules depend on their types. The following sections explain how each rule type is constructed and how each rule type affects the packets that match its condition.

Rule types

This section describes the rule types.

ACCEPT, DROP, REJECT, RETURN

These rule types do not modify packets. They simply return the action corresponding to their type/name. When constructing one of these rules, you only need to specify the type and the condition. When the rule is invoked, it checks to see whether the packet matches the rule's condition and if so, it returns the action associated with the rule's type/name, e.g. (DROP, packet) if the rule type is DROP. If the packet does not match the condition, the rule returns (CONTINUE, packet) to its caller.

There isn't a rule type CONTINUE, because such a rule would return (CONTINUE, packet) regardless of the packet's contents. Because this rule doesn't modify packets, the rule would be a useless operation.

DNAT, SNAT

These rule types modify packets. They rewrite source/destination network addresses and TCP/UDP port numbers. When constructing one of these rules, apart from the condition for matching packets, you must specify two things:

- A list of possible translation targets for matching packets' source/destination addresses.
- The next_action to return to the call chain that invoked this rule. Legal values are: ACCEPT, CONTINUE, and RETURN.

The next_action gives you more flexibility in constructing chains. After matching a packet and therefore modifying it (translating some of its addresses) the choice of what to do next is complex and is left up to you. The available options are:

- Exit all rule chains (ACCEPT).
- Invoke the next rule in the current chain (CONTINUE).
- Exit the current chain and invoke the calling chain's next rule, if the caller is a chain and if it has another (next) rule (RETURN).

Note that DNAT and SNAT rules do not distinguish between forward and return flows/packets. Packets going in the same direction as the packet that initiated the connection belong to a forward flow and packets in the opposite direction belong to a return flow. DNAT and SNAT rules simply check for a condition and if it matches, they apply the translation and then record that state so that it can be accessed during the processing of the return flow. In other words, the translation mapping is stored and used to perform the reverse translation for the return traffic flow (see "REV_DNAT, REV_SNAT"). Therefore, it important that you correctly do the following:

- Use REV_DNAT and REV_SNAT rules to reverse the address/port translations.

- Correctly order DNAT and REV_SNAT rules in pre-routing, and SNAT and REV_DNAT in post-routing.
- Avoid using DNAT rules in post-routing and SNAT rules in pre-routing.

REV_DNAT, REV_SNAT

These rule types modify packets. They rewrite source (SNAT)/destination (DNAT) network addresses and TCP/UDP port numbers. When constructing one of these rules, apart from the condition for matching packets, you must specify the next_action that should be returned to the rule's caller when packets match AND a reverse translation is found (otherwise, CONTINUE is returned). When a packet matches one of these rules, the rule looks up the reverse translation in a centralized map (soft state) and then applies it to the packet. That is why these rules don't need to specify the translation target like for DNAT and SNAT.

Jump

This rule type never modifies a packet. When constructing one of these rules, apart from the condition for matching packets, you must specify the jump_target: that is, the name of the rule chain that should be invoked for packets that match. Note that the jump_target should not be the name of the chain that contains the jump rule, as this would cause a rule-chain loop; you need to avoid looping rule-chains. To help avoid loops, the forwarding logic will detect rule-chain loops and drop any packet that visits a chain it has already visited.

When a packet matches a jump rule's condition, the action taken depends on whether the rule is invoked pre- or post-routing:

- If the rule is invoked pre-routing: the rule finds the rule chain specified by its jump_target and calls the chain to process the packet on the ingress port.
- If it's invoked post-routing, the rule calls the chain that processes the packet on the ingress, egress port.
- If the chain cannot be found, the jump rule returns the default, CONTINUE.

Rule order

Rules are stored as an ordered list in a rule chain and are evaluated in the listed order.

A specific rule is not evaluated in the event any preceding rules (or rules in chains that may have been 'jumped to') matched and resulted in an action (e.g. REJECT, ACCEPT) that terminates the processing of the packet in the chain.

The position of a rule in a rule chain is not an attribute of the rule. In the REST API, the rule-creation method specifies an integer position for the new rule in the rule chain. But that position is only meaningful relative to the rules already existing in the chain and is only used to modify the current rule list.

Note

Please read the section called "A packet's flow within a rule chain" [29] for an overview of rule-chain processing.

Rule conditions

Every rule has a single Condition object that a packet must match in order for the rule to be applied.

Taking a jump rule as an example, if a packet matches the jump's Condition object, then rule processing for that packet will continue in the jump's target chain; if the packet doesn't match, then processing continues with the rule following the jump in the jump's own rule-chain.

Condition objects specify a set or combination of attributes. Attributes are simple statements about the contents of a packet's headers. Examples of attributes are:

- 'the packet's TCP/UDP port number is between 500 and 1000'
- 'the packet's source IP address is in 10.0.0.0/16'

Note

Conditions are checked against the packet in the state the packet is in when it reaches a rule. In other words, for example, if a previous rule modified the packet's port number, then the current rule's condition will be checked against the modified, not the original, port number.

In order to form a Condition, you specify a number of attributes (optionally, you can invert most attributes using the CLI). Enter an exclamation point (!) or "bang" symbol to invert it, as shown in the "CLI Rule Chain Attributes That Match Packets" table. For example, if you invert the src attribute, this matches packets whose source does not match the specified IP address or network.

Below is the list of Condition attributes (attributes, invert-flags, and arguments) and their descriptions.

Note

The ports identified in rules are virtual ports on virtual routers or bridges. A virtual port may be bound to a specific Ethernet interface (like a tap) on a physical host OR it may be peered with another virtual port (in which case it connects two virtual devices). In either case, think of the virtual port as virtual because the rules only exist in the virtual topology AND nothing is known during rule evaluation about possible bindings of the virtual port to physical Ethernet interfaces.

Table 7.1. CLI Rule Chain Attributes

Attributes	Description
pos <INTEGER>:	The rule's position in the chain
type <TYPE>:	The rule <TYPE>; this is mostly used to distinguish between regular filtering rules and different types of NAT rules. The recognized <TYPE> values are: accept, continue, drop, jump, reject, return, dnat, snat, rev_dnat, rev_snat.
action accept	continue
return:	The rule action, meaningful for NAT rules only.
jump-to <CHAIN>:	The chain to jump to (if this is a jump rule).
target <IP_ADDRESS[-IP_ADDRESS][:INTEGER[-INTEGER]]>:	The NAT target, if this is a dnat or snat rule. At least one IP address must be given, optionally the NAT target may

Attributes	Description
	also contain a second address to form an address range and L4 port number or range of ports.

Table 7.2. CLI Rule Chain Attributes That Match Packets

Attributes That Match Packets	Description
hw-src [!]<MAC_ADDRESS>:	The source hardware address
hw-dst [!]<MAC_ADDRESS>:	The destination hardware address
ethertype [!]<STRING>:	Sets the data link layer (EtherType) of packets matched by this rule.
in-ports [!]<PORT[,PORT...]>:	Matches the virtual port through which the packet ingressed the virtual device that is currently processing the packet.
out-ports [!]<PORT[,PORT...]>:	Matches the port through which the packet egresses the virtual device that is currently processing the packet.
tos [!]<INTEGER>:	The value of the packet's Type of Service (TOS) field to match Use this field to match the differentiated services value. See TOS for information.
proto [!]<INTEGER>:	The IP protocol number to match. See Protocol Numbers for information. Examples: ICMP = 1, IGMP = 2, TCP = 6, UDP = 17
src [!]<CIDR>:	The source IP address or CIDR block
dst [!]<CIDR>:	The destination IP address or CIDR block
src-port [!]<INTEGER[-INTEGER]>:	The TCP or UDP source port or port range
dst-port [!]<INTEGER[-INTEGER]>:	The TCP or UDP destination port or port range
flow <fwd-flow|return-flow>:	Matches the connection-tracking status of the packet. If the packet is starting a new connection, fwd-flow will match. Alternatively, if the packet belongs to a connection already known to MidoNet, return-flow will match.
port-group [!]<PORT_GROUP>:	Matches a port group. Port groups allow the grouping of virtual ports to ease the creation of chain rules. See the CLI commands help for information.
ip-address-group-src [!]<IP_ADDRESS_GROUP>:	Matches a source IP address group. IP address groups allow the grouping of IP addresses to ease the creation of chain rules. See the CLI commands help for information.
ip-address-group-dst [!]<IP_ADDRESS_GROUP>:	Matches a destination IP address group. IP address groups allow the grouping of IP addresses to ease the creation of chain rules. See the CLI commands help for information.
hw-src-mask	**Source MAC address mask** - A 48-bit bitmask in the form xxxx.xxxx.xxxx, where x is any hexadecimal digit. Specifies which bits are to be considered when applying the rule's hw-src test. Default value = ffff.ffff.ffff: All bits are considered when applying the hw-src test, so a packet's source MAC address must match hw-src exactly. ffff.0000.0000: Only the first sixteen bits are considered when applying the hw-src test, the first sixteen bits of a packet's source MAC address must match the first sixteen bits of hw-src. 0000.0000.0000: No bits are considered when applying the hw-src test, so any packet will match.
hw-dst-mask	**Destination MAC address mask** - A 48-bit bitmask in the form xxxx.xxxx.xxxx, where x is any hexadecimal digit. Specifies which bits are to be considered when applying the rule's hw-dst test. Default value = ffff.ffff.ffff: All bits are considered when applying the hw-dst test, so a packet's destination MAC address must match hw-dst exactly.

Attributes That Match Packets	Description
	ffff.0000.0000: Only the first sixteen bits are considered when applying the hw-dst test, the first sixteen bits of a packet's destination MAC address must match the first sixteen bits of hw-dst. 0000.0000.0000: No bits are considered when applying the hw-dst test, so any packet will match.
fragment-policy header | nonheader | any | unfragmented	**fragment-policy** - Specifies the fragment type to match. ANY: Matches any packet. HEADER: Matches any packet that has a full header,that is, a header fragment or unfragmented packet. NONHEADER: Matches only nonheader fragments. UNFRAGMENTED: Matches only unfragmented packets. In general, ANY is the default policy. However, if a rule has a value for the src or dst field, the NONHEADER and ANY policies are disallowed and the default is HEADER. Furthermore, if the rule's type is dnat or snat and its target is not a single IP address with no ports specified, then the policy will default to UNFRAGMENTED, which is the only policy permitted for such rules. Unlike other rule properties, fragment-policy may not be inverted.

Example condition 1

Only packets whose network source is in 10.0.0.0/16 are allowed through to network 10.0.5.0/24. You can accomplish this a few different ways.

One way is to construct a DROP or REJECT rule that has a Condition and an ACCEPT rule with these attributes specify:

1. DROP when (ethertype equal 2048) AND (src NOT equal (10.0.0.0, 16))
2. ACCEPT when (dst equal (10.0.5.0, 24))
3. DROP

Note

The unconditional drop is needed to make rule 2 meaningful.

To create a rule chain with the above attributes:

1. If necessary, use the sett command or some other means to access the desired tenant.

```
midonet> sett 10a83af63f9342118433c3a43a329528
tenant_id: 10a83af63f9342118433c3a43a329528
```

2. Enter the command to create a new rule chain and assign it a name:

```
midonet> chain create name "drop_not_src_mynetwork_INBOUND"
chain5
```

3. Enter the command to drop IPv4 traffic that does not have the source 10.0.0.0/16:

```
midonet> chain chain5 add rule ethertype 2048 src !10.0.0.0/16 type drop
chain5:rule0
```

4. Enter the command to accept IPv4 traffic with the destination 10.0.5.0/24:

```
midonet> chain chain5 add rule ethertype 2048 dst 10.0.5.0/24 pos 2 type
 accept
chain5:rule2
```

5. Enter the command to list the rules added to the new rule chain:

```
midonet> chain chain5 list rule
rule rule3 ethertype 2048 src !10.0.0.0/16 proto 0 tos 0 pos 1 type drop
rule rule2 ethertype 2048 dst 10.0.5.0/24 proto 0 tos 0 pos 2 type
 accept
```

Example condition 2

Same as Example Condition 1, except here assume that you're structuring your rules differently. You want to have one DROP rule at the end of the chain that matches all packets; earlier in the chain you place ACCEPT rules that match packets/flows that are specifically allowed through.

The ACCEPT rule for the traffic allowed by Example Condition 1 would have a Condition with these attributes:

In the rule language, the chain would have:

ACCEPT when src=(10.0.0.0, 16) OR dst=(10.0.5.0, 24)

Rule at the end:

DROP all other packets

To create a rule chain with the above attributes:

1. If necessary, use the sett command or some other means to access the desired tenant.

```
midonet> sett 10a83af63f9342118433c3a43a329528
tenant_id: 10a83af63f9342118433c3a43a329528
```

2. Enter the command to create a new rule chain and assign it a name:

```
midonet> chain create name "accept_src_dst_mynetwork_INBOUND"
chain11
```

3. Enter the command to accept IPv4 traffic from the source 10.0.0.0/16:

```
midonet> chain chain11 add rule ethertype 2048 src 10.0.0.0/16 type
 accept
chain11:rule0
```

4. Enter the command to accept IPv4 traffic with the destination 10.0.5.0/24:

```
midonet> chain chain11 add rule ethertype 2048 dst 10.0.5.0/24 type
 accept
chain11:rule1
```

5. Enter the command to drop all IPv4 traffic (that didn't match the attributes in the preceding rules):

```
midonet> chain chain11 add rule ethertype 2048 pos 3 type drop
chain11:rule2
```

6. Enter the command to list the rules added to the new rule chain:

```
midonet> chain chain11 list rule
rule rule1 ethertype 2048 dst 10.0.5.0/24 proto 0 tos 0 pos 1 type
 accept
rule rule0 ethertype 2048 src 10.0.0.0/16 proto 0 tos 0 pos 2 type
 accept
rule rule3 ethertype 2048 proto 0 tos 0 pos 3 type drop
```

MidoNet rule chain examples

This example shows how you can use the MidoNet CLI to display the rule chains used to implement security groups.

MidoNet implements security groups by configuring rule chains on a port basis, as well as on bridges and routers (pre/post filtering).

Assumptions

For this example, assume the following network conditions:

- A tenant named "demo"
- A network (bridge) named demo-private-net
- A VM with a private network IP address of 172.16.3.3 and a MAC address of fa:16:3e:fb:19:07

You configured a security group that allowed:

- Ingress traffic from TCP port 5900 (Virtual Network Computing)
- Ingress traffic from TCP port 22 (SSH)
- Ingress traffic from TCP port 80 (HTTP)
- Ingress ICMP traffic

Listing the Bridges for the Tenant

Rule chains relating to OpenStack security groups are implemented on the network (bridge) ports.

To list the bridge(s) on the tenant and show the demo-private-net network (bridge), enter the command:

```
midonet> list bridge
bridge bridge0 name demo-private-net state up
```

Listing the Ports on the Bridge

To list information about the rule chains configured on the bridge ports, enter the command:

```
midonet> bridge bridge0 list port
port port0 device bridge0 state up
port port1 device bridge0 state up infilter chain2 outfilter chain3
port port2 device bridge0 state up peer router1:port1
```

Note

Ports with infilter (pre-routing) and outfilter (post-routing) chains are connected to VMs. port1 is connected to a VM.

Listing the Rules for the Inbound Chain on a Port

To list the pre-routing rule chain for port1, enter the command:

```
midonet> chain chain2 list rule
rule rule0 ethertype 2048 src !172.16.3.3 proto 0 tos 0 pos 1 type drop
rule rule1 hw-src !fa:16:3e:fb:19:07 proto 0 tos 0 pos 2 type drop
rule rule2 proto 0 tos 0 flow return-flow pos 3 type accept
rule rule3 proto 0 tos 0 pos 4 type jump jump-to chain4
rule rule4 ethertype !2054 proto 0 tos 0 pos 5 type drop
```

The pre-routing rule chain contains the following instructions:

- rule0 says: for packets that match the ethertype 2048 (IPv4) that do not match the source IP address 172.16.3.3 (the private IP address of the VM), drop these packets. This prevents the port's VM from sending packets with a forged IP address.
- rule1 says: for packets with a hardware source that does not match the listed source MAC address (the VM's MAC address), drop these packets. This prevents the VM from sending packets with a forged MAC address.
- rule2 says: for packets that match a return flow (that is, a packet that belongs to a connection already known to MidoNet), accept these packets.
- rule3 says: for packets that were not dropped or accepted as a result of matching previous rules, allow these packets to jump to the indicated chain (chain4).
- rule4 says: for packets that do not match the ethertype 2054 (ARP packets), drop these packets.

Listing the OpenStack Security Group Rule Chain

You can list all the rule chains and then look at the rule chain for the OpenStack security group.

To list all the rule chains and examine specific rule chains, enter the command:

```
midonet> list chain
chain chain5 name OS_SG_050593ed-56ad-44ef-8489-4052d02d99ff_INGRESS
chain chain0 name OS_PRE_ROUTING_5a151b0b-dea7-4918-bd17-876c1f7f5c64
chain chain1 name OS_POST_ROUTING_5a151b0b-dea7-4918-bd17-876c1f7f5c64
chain chain6 name OS_SG_01fce1b8-c277-4a37-a8cc-86732eea186d_INGRESS
chain chain4 name OS_SG_050593ed-56ad-44ef-8489-4052d02d99ff_EGRESS
chain chain7 name OS_SG_01fce1b8-c277-4a37-a8cc-86732eea186d_EGRESS
chain chain2 name OS_PORT_6f72342b-4947-432f-8d01-0cf4e4b8d049_INBOUND
chain chain3 name OS_PORT_6f72342b-4947-432f-8d01-0cf4e4b8d049_OUTBOUND
```

Note chain5, identified as a chain for an OpenStack security group (OS_SG) for INGRESS traffic.

To look at rule chain5, enter the command:

```
midonet> chain chain5 list rule
rule rule0 ethertype 2048 src 0.0.0.0/0 proto 6 tos 0 dst-port 5900 pos 1
 type accept
```

```
rule rule1 ethertype 2048 src 0.0.0.0/0 proto 1 tos 0 pos 2 type accept
rule rule2 ethertype 2048 src 0.0.0.0/0 proto 6 tos 0 dst-port 22 pos 3
 type accept
rule rule3 ethertype 2048 src 0.0.0.0/0 proto 6 tos 0 dst-port 80 pos 4
 type accept
```

The above output shows the rule chain used to implement the security group you configured in OpenStack. These rules contain the following instructions:

- All the rules match ethertype 2048 (IPv4) packets.
- All the rules match traffic from any source network (0.0.0.0/0).
- All the rules, except rule1, match packets of IP protocol 6 (TCP) and accept them. rule1 matches packets of the ICMP type and accepts them.
- In addition to the other matches already mentioned, the rules match and accept the packets according to the security group rules you defined in OpenStack, specifically, packets with the destinations:
  - TCP port 5900 (VNC)
  - TCP port 22 (SSH)
  - TCP port 80 (HTTP)

Setting and clearing Rule Chains

To set or change a rule chain on a device, use the `set` command:

```
midonet> bridge bridge0 port port1 set infilter chain8
```

To remove a rule chain from a device, use the `clear` command:

```
midonet> bridge bridge0 port port1 clear infilter
```

8. Network Address Translation

Table of Contents

To demonstrate how MidoNet uses static NAT, this chapter shows how MidoNet implements floating IP addresses by examining a floating IP address configuration.

Static NAT

This section demonstrates how MidoNet uses static NAT to implement floating IP addresses.

MidoNet implements floating IP addresses in a two-stage process:

1. Bring traffic to a floating IP address (that is, bring traffic from an external network to a tenant router).
2. Perform network address translation from the external network's public IP address to a private IP address and in the reverse direction.

Assumptions

To view this example, this section assumes you have configured:

- A project
- An Edge Router
- An external network
- A tenant router
- A private network (bridge)
- At least one VM connected to the bridge
- A floating IP address assigned to at least one VM.

Viewing NAT rule chain information

You can use the MidoNet CLI to list the routers for a tenant and list rule chains configured on the router.

The example below explains how MidoNet uses rule chains to implement source and destination NAT and shows how to use the MidoNet CLI to:

- Display information about the infilter (pre-routing) and outfilter (post-routing) chains configured on a tenant router.
- List the rules for the rule chains.

Assumptions

For the example below, assume the following network conditions exist:

- A tenant router named "tenant-router"
- A private network with a 172.16.3.0/24 network address
- A public network with a 198.51.100.0/24 network address
- A VM with a 172.16.3.3 private IP address and a 198.51.100.3 public (floating) IP address

Viewing a Pre-Routing rule

To list routers on the current tenant and the router rule-chain information on the router(s), enter the command:

```
midonet> list router
router router0 name tenant-router state up infilter chain0 outfilter chain1
```

As shown in the above output, chain0 is the router's pre-routing (infilter) rule chain and chain1 is the post-routing (outfilter) rule chain.

To list information about the router's pre-routing rule chain, enter the command:

```
midonet> chain chain0 list rule
rule rule0 dst 198.51.100.3 proto 0 tos 0 in-ports router0:port0 pos 1 type
 dnat action accept target 172.16.3.3
rule rule1 dst 198.51.100.2 proto 0 tos 0 in-ports router0:port0 pos 2 type
 rev_snat action accept
```

rule0 of the pre-routing rule chain on the tenant router contains the following instructions:

- For packets with the destination of 198.51.100.3 (the floating IP address associated with the VM):
  - Perform a destination NAT (DNAT) translation to change the destination IP address from the VM's floating IP address (198.51.100.3) to the VM's private IP address (172.16.3.3).

Viewing a Post-Routing rule

To list the post-routing rule on the tenant router, enter the command:

```
midonet> chain chain1 list rule
rule rule0 src 172.16.3.3 proto 0 tos 0 out-ports router1:port0 pos 1 type
 snat action accept target 198.51.100.3
rule rule1 proto 0 tos 0 out-ports router1:port0 pos 2 type snat action
 accept target 198.51.100.2:1--1
```

rule0 of the post-routing rule on tenant-router contains the following instructions:

- For packets from the source IP address 172.16.3.3 (the private IP address of the VM):
  - Perform a source NAT (SNAT) translation to change the source IP address from the VM's private IP address (172.16.3.3) to the VM's floating IP address (198.51.100.3).

Configuring SNAT, DNAT, and REV_DNAT

The sections below describe how to configure SNAT, DNAT, and REV_DNAT using rule-chains.

To configure SNAT and DNAT rule chains, you need to specify:

- The rule type, for example, snat or dnat
- The action, for example, accept
- The translation target(s)

Using the MidoNet CLI, you don't have to specify an address or port range if it only has one member. However, when configuring Dynamic SNAT you must explicitly set ports or port ranges with the CLI command, otherwise, it will be considered a static NAT and connection tracking will not work.

Below are examples of valid NAT targets in the CLI syntax:

```
192.168.1.1:80
192.168.1.1-192.168..1.254
192.168.1.1:80-88
192.168.1.1-192.168.1.254:80-88
```

The in-port you specify in DNAT rules matches the virtual port on a virtual router or bridge through which the packet ingressed the virtual device that is currently processing the packet. The out-port you specify in SNAT rules matches the port through which the packet egresses the virtual device that is currently processing the packet.

For REV_SNAT and REV-DNAT rules, as mentioned earlier, when a packet matches one of these rules, the rule looks up the reverse translation in a centralized map (soft state) and then applies it to the packet. That is why these rules don't need to specify the translation target like for DNAT and SNAT.

Below is a CLI example of a DNAT rule:

```
chain chain9 add rule dst 198.51.100.4 in-ports router1:port0 pos 1 type
 dnat action accept target 10.100.1.150
```

DNAT/REV_DNAT example

This example shows how to use MidoNet CLI commands to configure DNAT and REV_DNAT on a router.

Below are the assumptions regarding the rule chain for this example:

- You have two subnets: a public subnet (198.51.100.0/24) and a private subnet (10.0.0.0/24).
- You have a virtual machine connected to the tenant router with a public IP address of 198.51.100.4 and a private IP address of 10.100.1.150.

- You want to translate traffic with the VM's public IP address to the VM's private IP address.
- You want to perform a reverse destination address translation.

To create the rule chain for the above DNAT configuration:

1. Create a new rule chain:

```
midonet> chain create name "dnat-test"
chain10
```

2. Create a rule that accepts traffic on a router port with the destination 198.51.100.4 and translates the destination to 10.100.1.150:

```
midonet> chain chain10 add rule dst 198.51.100.4 in-ports router1:port0
 pos 1 type dnat action accept target 10.100.1.150
chain10:rule2
```

3. Create a rule that accepts traffic with the destination of the router's gateway to the public network and performs a reverse address translation from the public network address to the private network address.

```
midonet> chain chain10 add rule dst 198.51.100.2 in-ports router0:port0
 pos 2 type rev_snat action accept
chain10:rule3
```

4. List the rules to check them:

```
midonet> chain chain10 list rule
rule rule2 dst 198.51.100.4 proto 0 tos 0 in-ports router1:port0 pos 1
 type dnat action accept target 10.100.1.150
rule rule3 dst 198.51.100.2 proto 0 tos 0 in-ports router0:port0 pos 2
 type rev_snat action accept
```

SNAT REV_SNAT example

This example shows how to use MidoNet CLI commands to configure source NAT (known as SNAT) and reverse source nat (we call that REV_SNAT) on a router using static NAT (one private ip maps to one public ip). Please be aware that reverse source nat and DNAT are not the same thing (it depends on the connection and the associated flow state).

You can configure SNAT for traffic of any network device (like a VM or a container) that enters the virtual topology. This means you can also use a MidoNet gateway to do static source NAT for physical servers sitting in a physical network connected to a MidoNet gateway acting as a L2 gateway or L3 router.

For demonstration purposes and to keep things simple we will use a veth pair and a network namespace in a simple installation with one midonet gateway acting as a static L3 router.

Note that the router does not have to have its virtual ports configured with ip addresses that we are using for NAT. Also you do not need to modify the routing table in the router that is actually responsible for doing the NAT.

This also holds true for L4 load balancing (which is also implemented in MidoNet using NAT): the ip of the load balancer does not have to be an ip on a port of a router.

For this example to work, your physical network must be configured to route traffic for 200.200.200.0/24 to the ip address of your edge router.

The following assumptions are made for this example code: * the host for setting up the veth pair is called 'os004' * the edge router is simply called 'edge' * for brevity we have only one gw with static routing (you have to configure this yourself) in this example. * the underlay is 192.168.7.0/24 * the overlay is 192.168.11.0/24 * we want to make 192.168.11.0/24 appear as SNAT traffic coming from 200.200.200.0/24 * the reverse source NAT path should convert traffic going to 200.200.200.0/24 back to 192.168.11.0/24

On a machine with the MidoNet agent running, we create a small lab environment with a veth-pair and a separate network namespace:

```
ip link | grep veth0-snat || ip link add veth0-snat type veth peer name
 veth1-snat
ip link set dev veth0-snat up
ip netns show | grep snat-example || ip netns add snat-example
ip link set veth1-snat netns snat-example
ip netns exec snat-example ip link set dev veth1-snat up
ip netns exec snat-example ip addr add 192.168.11.11/24 dev veth1-snat
ip netns exec snat-example ip route add default via 192.168.11.1
```

We also create a new bridge, create ports on the bridge and set up the edge router port.

```
midonet-cli -e 'bridge list name snat-example-bridge' || midonet-cli -e
 'bridge create name snat-example-bridge'
export BRIDGE="$(midonet-cli -e 'bridge list name snat-example-bridge' |
 awk '{print $2;}')"

midonet-cli -e "bridge $BRIDGE port list" | wc -l | grep '2' || midonet-cli
 -e "bridge $BRIDGE port create"
midonet-cli -e "bridge $BRIDGE port list" | wc -l | grep '2' || midonet-cli
 -e "bridge $BRIDGE port create"

export VETH_PORT="$(midonet-cli -e "bridge $BRIDGE port list" | head -n1 |
 awk '{print $2;}')"
export ROUTER_PORT="$(midonet-cli -e "bridge $BRIDGE port list" | tail -n1
 | awk '{print $2;}')"

export HOST="$(midonet-cli -e "host list name os004" | awk '{print $2;}')"
midonet-cli -e "host $HOST add binding port $VETH_PORT interface veth0-
snat"

export ROUTER="$(midonet-cli -e "router list name edge" | awk '{print
 $2;}')"
midonet-cli -e "router $ROUTER port list" | grep 'address 192.168.11.1' |
 grep 'net 192.168.11.0/24' || \
midonet-cli -e "router $ROUTER port create address 192.168.11.1 net 192.
168.11.0/24"

export RPORT="$(midonet-cli -e "router $ROUTER port list" | grep 'address
 192.168.11.1' | grep 'net 192.168.11.0/24' | awk '{print $2;}')"
midonet-cli -e "router $ROUTER route list" | grep 'dst 192.168.11.0/24' ||
 \
midonet-cli -e "router $ROUTER route add type normal src 0.0.0.0/0 dst 192.
168.11.0/24 port $RPORT"

midonet-cli -e "port $RPORT set peer $ROUTER_PORT"
```

Now you should be able to ping the router from inside the network namespace.

```
root@os004:~# ip netns exec snat-example ping -c1 192.168.11.1
```

```
PING 192.168.11.1 (192.168.11.1) 56(84) bytes of data.
64 bytes from 192.168.11.1: icmp_seq=1 ttl=64 time=0.546 ms
```

Pinging the outside world from this IP should not be possible at the moment.

```
root@os002:~# clear; tcpdump -i enp0s8 -l -nnn -vvv -X -e ether host
 ac:ab:ac:ab:ac:ab and icmp
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size
 262144 bytes
13:38:16.303495 ac:ab:ac:ab:ac:ab > 80:ee:73:83:93:56, ethertype IPv4
 (0x0800), length 98: (tos 0x0, ttl 63, id 63236, offset 0, flags [DF],
 proto ICMP (1), length 84)
    192.168.11.11 > 192.168.7.1: ICMP echo request, id 4361, seq 1, length
 64
  0x0000:  4500 0054 f704 4000 3f01 b147 c0a8 0b0b  E..T..@.?..G....
  0x0010:  c0a8 0701 0800 9f7f 1109 0001 b8a9 be58  ...............X
  0x0020:  0000 0000 0da1 0400 0000 0000 1011 1213  ................
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
  0x0050:  3435 3637                                4567
```

By running tcpdump on the outgoing NIC of the gateway we can easily see that SNAT and REV_SNAT is not yet happening. We will now change this outgoing ICMP echo request to be SNATified as 200.200.200.11.

The script may seem frightening when you look at it for the first time, but rest assured it is more simple than it looks.

```
export INFILTER="snat-example-infilter"

export ROUTER="$(midonet-cli -e "router list name edge" | awk '{print
 $2;}')"
export IPORT="$(midonet-cli -e "router $ROUTER port list" | grep 'address
 192.168.11.1' | grep 'net 192.168.11.0/24' | awk '{print $2;}')"
export XPORT="$(midonet-cli -e "router $ROUTER port list" | grep 'address
 192.168.7.2' | grep 'net 192.168.7.0/24' | awk '{print $2;}')"

midonet-cli -e "router $ROUTER clear infilter"

EXISTING="$(midonet-cli -e "chain list name $INFILTER" | awk '{print
 $2;}')"
if [[ ! "x" == "x$EXISTING" ]]; then midonet-cli -e "delete chain
 $EXISTING"; fi

export CHAIN="$(midonet-cli -e "chain create name $INFILTER")"

midonet-cli -e "chain $CHAIN add rule \
  src 192.168.11.11/32 \
  in-ports $IPORT \
  fragment-policy unfragmented \
  no-vlan false \
  pos 1 \
  type snat \
  action accept \
  target 200.200.200.11:10000-60000"

midonet-cli -e "chain $CHAIN add rule \
  dst 200.200.200.11 \
  in-ports $XPORT \
  fragment-policy any \
  no-vlan false \
  pos 1 \
  type rev_snat \
  action accept"
```

```
midonet-cli -e "router $ROUTER set infilter $CHAIN"
```

By setting the infilter on the router, this SNAT setup automatically becomes a firewall, this means that only this machine can now use the router and its traffic will always be SNATed when going over the router.

Here you can see the final results, tcpdump from the gateway (arp traffic omitted):

```
root@os002:~# clear; tcpdump -i enp0s8 -l -nnn -vvv -X -e ether host
 ac:ab:ac:ab:ac:ab and icmp
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size
 262144 bytes
14:36:41.512662 ac:ab:ac:ab:ac:ab > 80:ee:73:83:93:56, ethertype IPv4
 (0x0800), length 98: (tos 0x0, ttl 63, id 13058, offset 0, flags [DF],
 proto ICMP (1), length 84)
    200.200.200.11 > 192.168.7.1: ICMP echo request, id 4927, seq 1, length
 64
  0x0000:  4500 0054 3302 4000 3f01 b029 c8c8 c80b  E..T3.@.?..)....
  0x0010:  c0a8 0701 0800 138c 133f 0001 69b7 be58  .........?..i..X
  0x0020:  0000 0000 e350 0700 0000 0000 1011 1213  .....P..........
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
  0x0050:  3435 3637                                4567
14:36:41.513747 80:ee:73:83:93:56 > ac:ab:ac:ab:ac:ab, ethertype IPv4
 (0x0800), length 98: (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto
 ICMP (1), length 84)
    192.168.7.1 > 200.200.200.11: ICMP echo reply, id 4927, seq 1, length
 64
  0x0000:  4500 0054 0000 4000 4001 e22b c0a8 0701  E..T..@.@..+....
  0x0010:  c8c8 c80b 0000 1b8c 133f 0001 69b7 be58  .........?..i..X
  0x0020:  0000 0000 e350 0700 0000 0000 1011 1213  .....P..........
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
  0x0050:  3435 3637                                4567
```

This is the traffic in the VXLAN tunnels going between the node with the veth pair and the MidoNet gateway:

```
root@os002:~# clear; tcpdump -i enp0s3 -l -nnn -vvv -X -e -Tvxlan port 6677
tcpdump: listening on enp0s3, link-type EN10MB (Ethernet), capture size
 262144 bytes
14:37:26.709018 08:00:27:97:e4:37 > 08:00:27:c5:6b:60, ethertype IPv4
 (0x0800), length 148: (tos 0x0, ttl 255, id 24228, offset 0, flags [none],
 proto UDP (17), length 134)
    192.168.7.189.49719 > 192.168.7.190.6677: VXLAN, flags [I] (0x08), vni
 8072578
ac:ab:ac:ab:ac:ab > 80:ee:73:83:93:56, ethertype IPv4 (0x0800), length 98:
 (tos 0x0, ttl 63, id 17418, offset 0, flags [DF], proto ICMP (1), length
 84)
    200.200.200.11 > 192.168.7.1: ICMP echo request, id 4930, seq 1, length
 64
  0x0000:  4500 0086 5ea4 0000 ff11 cbf6 c0a8 07bd  E...^...........
  0x0010:  c0a8 07be c237 1a15 0072 0000 0800 0000  .....7...r......
  0x0020:  7b2d 8200 80ee 7383 9356 acab acab acab  {-....s..V......
  0x0030:  0800 4500 0054 440a 4000 3f01 9f21 c8c8  ..E..TD.@.?..!..
  0x0040:  c80b c0a8 0701 0800 a70e 1342 0001 96b7  ...........B....
  0x0050:  be58 0000 0000 1fcb 0a00 0000 0000 1011  .X..............
  0x0060:  1213 1415 1617 1819 1a1b 1c1d 1e1f 2021  ...............!
  0x0070:  2223 2425 2627 2829 2a2b 2c2d 2e2f 3031  "#$%&'()*+,-./01
  0x0080:  3233 3435 3637                           234567
14:37:26.714107 08:00:27:c5:6b:60 > 08:00:27:97:e4:37, ethertype IPv4
 (0x0800), length 148: (tos 0x0, ttl 255, id 45283, offset 0, flags [none],
 proto UDP (17), length 134)
```

```
   192.168.7.190.56640 > 192.168.7.189.6677: VXLAN, flags [I] (0x08), vni
 8204919
ac:ca:ba:33:65:40 > 2a:b5:3b:78:86:b0, ethertype IPv4 (0x0800), length 98:
 (tos 0x0, ttl 63, id 0, offset 0, flags [DF], proto ICMP (1), length 84)
   192.168.7.1 > 192.168.11.11: ICMP echo reply, id 4930, seq 1, length 64
  0x0000:  4500 0086 b0e3 0000 ff11 79b7 c0a8 07be  E.........y.....
  0x0010:  c0a8 07bd dd40 1a15 0072 0000 0800 0000  .....@...r......
  0x0020:  7d32 7700 2ab5 3b78 86b0 acca ba33 6540  }2w.*.;x.....3e@
  0x0030:  0800 4500 0054 0000 4000 3f01 a84c c0a8  ..E..T..@.?..L..
  0x0040:  0701 c0a8 0b0b 0000 af0e 1342 0001 96b7  ...........B....
  0x0050:  be58 0000 0000 1fcb 0a00 0000 0000 1011  .X..............
  0x0060:  1213 1415 1617 1819 1a1b 1c1d 1e1f 2021  ...............!
  0x0070:  2223 2425 2627 2829 2a2b 2c2d 2e2f 3031  "#$%&'()*+,-./01
  0x0080:  3233 3435 3637                           234567
```

For reference, this is a tcpdump on the veth pair before the packet enters the overlay and after the return packet leaves the overlay:

```
root@os004:~# clear; ip netns exec snat-example tcpdump -i veth1-snat -l -
nnn -vvv -X -e
tcpdump: listening on veth1-snat, link-type EN10MB (Ethernet), capture size
 262144 bytes
14:39:46.635497 2a:b5:3b:78:86:b0 > ac:ca:ba:33:65:40, ethertype IPv4
 (0x0800), length 98: (tos 0x0, ttl 64, id 24548, offset 0, flags [DF],
 proto ICMP (1), length 84)
   192.168.11.11 > 192.168.7.1: ICMP echo request, id 5036, seq 1, length
 64
  0x0000:  4500 0054 5fe4 4000 4001 4768 c0a8 0b0b  E..T_.@.@.Gh....
  0x0010:  c0a8 0701 0800 d8bc 13ac 0001 22b8 be58  ............"..X
  0x0020:  0000 0000 62b2 0900 0000 0000 1011 1213  ....b...........
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
  0x0050:  3435 3637                                4567
14:39:46.637896 ac:ca:ba:33:65:40 > 2a:b5:3b:78:86:b0, ethertype IPv4
 (0x0800), length 98: (tos 0x0, ttl 63, id 0, offset 0, flags [DF], proto
 ICMP (1), length 84)
   192.168.7.1 > 192.168.11.11: ICMP echo reply, id 5036, seq 1, length 64
  0x0000:  4500 0054 0000 4000 3f01 a84c c0a8 0701  E..T..@.?..L....
  0x0010:  c0a8 0b0b 0000 e0bc 13ac 0001 22b8 be58  ............"..X
  0x0020:  0000 0000 62b2 0900 0000 0000 1011 1213  ....b...........
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
  0x0050:  3435 3637                                4567
```

As you can see, the MidoNet agent on the originating node already does the necessary SNAT transformations, the gateway is just used to send the packet into the physical network. This is another reason why using a NIC with vxlan offloading is fast, there is no userland trip involved on the egress.

Vice versa (and here we are touching on the subject of the problem of how to scale flow state using a distributed system) the REV_SNAT is happening on the gateway ingress when receiving the ICMP reply, the transformations are done before the packet enters the VXLAN tunnel.

Both the gateway and the compute node running the veth pair lab have kernel flow table rules to make unpacking the VXLAN tunnel data fast and without userland intervention.

9. Layer 4 Load Balancing

Table of Contents

The load balancer service in MidoNet provides Layer 4 (L4) load balancing. A typical use case involves a tenant wishing to balance traffic (load) from floating IP addresses to private IP addresses.

Generally, the traffic comes in from publicly routable addresses (for example, from users of the service all over the world) to a virtual IP address (VIP) (an IP assigned to the load balancer instead of a VM), and is then load-balanced to numerous private IP addresses within the cloud. The load balancer sends the traffic to the private IP addresses of the back-end servers – in MidoNet's case, these back-end servers are VMs. The load balancer does not terminate the connection, nor is it really transparent because it's translating the destination IP.

Configuration overview

As part of the configuration, you define a pool of back-end servers (VMs) to which traffic is load-balanced. The pool members typically have private IP addresses. The placement of the VMs is flexible but must take into account where the load balancer is configured. A general rule is that the VM private addresses **must be** reachable from the router to which the load balancer is attached. Therefore, pool members may

- all belong to a single subnet of the router, or
- they may be placed in two or more subnets of the router

Finally, note that because the load balancer leaves the request source address unmodified, the load balancer must be placed in the path of the return traffic – otherwise return traffic will go directly to the request source without giving the load balancer a chance to reverse the VIP#back-end-IP translation that was applied on the forward packets.

Health monitoring

In addition, you can configure a health monitor to perform checks on the back-end servers. The health monitor automatically removes unhealthy nodes from the pool and adds them back when they return to health.

Load balancer configuration

This procedure outlines the steps to configure a load-balancer for an existing tenant in MidoNet. In the example, we will use:

- VIP: 203.0.113.2
- Pool Members: 192.168.100.100, 192.168.100.101

Let's start by determining the list of routers we have available in our MidoNet deployment:

```
midonet> list router
router router0 name Edge Router state up
router router1 name Tenant Router state up infilter chain0 outfilter chain1
```

As you can see, your Tenant Router where you are going to create a load balancer is `router1`.

Important

In MidoNet, routers have inbound and outbound filters. If the load balancer on a router balances traffic, these filters will be skipped. When using MidoNet with OpenStack, these filters usually only contain NAT rules that are irrelevant to load-balanced traffic, but this is worth taking into account if you are adding custom rules to the router's filters.

1. Create a load balancer and assign it to the Tenant Router.

   ```
   midonet> load-balancer create
   lb0
   midonet> router router1 set load-balancer lb0
   ```

   The load balancer assigned to the router will act on traffic flowing through that router.

2. Create a pool to which target back-end servers will be assigned.

   ```
   midonet> load-balancer lb0 create pool lb-method ROUND_ROBIN
   lb0:pool0
   midonet> load-balancer lb0 pool pool0 show
   pool pool0 load-balancer lb0 lb-method ROUND_ROBIN state up
   ```

3. Next, add target back-end servers to the pool you just created.

   ```
   midonet> load-balancer lb0 pool pool0 create member address 192.168.100.
   100 protocol-port 80
   lb0:pool0:pm0
   midonet> load-balancer lb0 pool pool0 create member address 192.168.100.
   101 protocol-port 80
   lb0:pool0:pm1
   midonet> load-balancer lb0 pool pool0 member pm0 show
   pm pm0 address 192.168.100.1 protocol-port 80 weight 0 state up
   ```

   For each back-end server you must add its IP address and port to the pool.

4. Create a virtual IP address (VIP) and port, then assign it to the pool against which load balancing will be performed (lb0:pool0). Typically, a VIP is an IP address from the public IP space.

   ```
   midonet> load-balancer lb0 pool pool0 list vip
   midonet> load-balancer lb0 pool pool0 create vip address 203.0.113.2
    persistence SOURCE_IP protocol-port 8080
   lb0:pool0:vip0
   midonet> load-balancer lb0 pool pool0 vip vip0 show
   vip vip0 load-balancer lb0 address 203.0.113.2 protocol-port 8080
    persistence SOURCE_IP state up
   ```

   ### Note

   Above example uses sticky source IP address persistence, read more about it in the section called "Sticky Source IP" [49].

5. Lastly, you must add a routing rule on the Edge Router (`router0`) so that a packet sent to the VIP is able to find its way to the Tenant Router where the LB is defined.

   a. First, identify the ports on the Edge Router, using the `router router0 list port` command, like so:

```
midonet> router router0 list port
port port0 device router0 state up mac 02:c2:0f:b0:f2:68 addresses
 100.100.100.1/30
port port1 device router0 state up mac 02:cb:3d:85:89:2a addresses
 172.168.0.1/16
port port2 device router0 state up mac 02:46:87:89:49:41 addresses
 200.200.200.1/24 peer bridge0:port0
port port3 device router0 state up mac 02:6b:9f:0d:c4:a8 addresses
 169.254.255.1/30 peer router1:port0
...
```

   Identify the port on the Edge Router that is used to route traffic to the Tenant Router (`router1`). In this example, we see `port3` on device `router0` is peered to the Tenant Router port (`router1:port0`).

   b. Next, add the route to the Edge Router to send traffic to `router0:port3` (local port on Edge Router).

```
midonet> router router0 add route dst 203.0.113.2/32 src 0.0.0.0/0
 type normal port router0:port3
router0:route11
```

   This rule matches any incoming traffic (`src 0.0.0.0/0`) to the Edge Router that is sent to the VIP (`dst 203.0.113.2/32`) and sends it out the desired Edge Router port (`router0:port3`).

To delete an existing load balancer, run the following command:

```
midonet> load-balancer list
lb lb0 state up
midonet> load-balancer lb0 delete
```

Sticky Source IP

On many occasions, you want a load balancer to keep track of the sessions. To accomplish this, the MidoNet load balancer provides sticky source IP address persistence.

When configured on a virtual IP (VIP), the source IP address of the packet is used to determine the destination back-end server, and all the subsequent traffic from the same source IP address gets sent to the same server.

Session persistence example

The example below shows how to use the MidoNet CLI to configure a VIP with sticky source IP address persistence (`persistence SOURCE_IP`).

```
midonet> load-balancer lb0 pool pool0 create vip address 203.0.113.2
 persistence SOURCE_IP protocol-port 8080
lb0:pool0:vip0
midonet> load-balancer lb0 pool pool0 vip vip0 show
vip vip0 load-balancer lb0 address 203.0.113.2 protocol-port 8080
 persistence SOURCE_IP state up
```

To disable sticky source IP address persistence, set it to `NONE`:

```
midonet> load-balancer lb0 pool pool0 vip vip0 set persistence NONE
```

```
midonet> load-balancer lb0 pool pool0 vip vip0 show
vip vip0 load-balancer lb0 address 203.0.113.2 protocol-port 8080 state up
```

Important

- If you toggle on/off sticky source IP address mode on a VIP, existing connections using that VIP will be dropped.
- If you disable a pool member while in sticky source IP address mode, existing connections that are balanced to that member will be dropped.
- If you disable a pool member while NOT in sticky source IP address mode, existing connections that are balanced to that member will be allowed to finish, but no new connections will be sent to that member.
- Stickiness remains active for one day. If a session is inactive for more than a day, the stickiness goes away and the subsequent traffic will be load balanced normally.

Health monitor

Health monitoring is the act of checking a set of pool members for "aliveness". This usually means HTTP, TCP, UDP, or ICMP connectivity is possible to the node.

In MidoNet's case, only TCP connectivity is checked. Health monitors work by sending packets to the pool members and checking whether or not they receive a reply. The node is considered ACTIVE if the pool member responds to the packet within a certain amount of time, and after a certain amount of retries. Therefore, health monitors act on the following three variables:

- max_retries: How many times the health monitor sends a packet to the pool member without receiving a response before the health monitor considers the node to be INACTIVE
- delay: Amount of time between each transmission of a packet from the health monitor to the pool member
- timeout: Additional timeout after a connection has been established

The health monitor keeps track of the current state of all pool members it is assigned to. Load balancing decisions can then be made based on the "aliveness" of a pool member.

HAProxy configuration

When using a Layer 4 load balancer, you can configure a health monitor to perform checks on the back-end servers.

Only a single host runs all health monitor instances at a given time. If that host goes down for any reason, a different host will be elected and spawn the HAProxy instances. These instances are managed by the MidoNet Agent and do not require any special setup. However, hosts that can potentially hold the HAProxy instances have to be chosen.

To enable a MidoNet Agent host for health monitoring, its `health_monitor_enable` property has to be set to `true`.

Run the following command to check if a host is enabled for health monitoring:

```
$ mn-conf get agent.haproxy_health_monitor.health_monitor_enable
```

To toggle health monitoring for a certain host, use the following commands on that host:

```
$ echo "agent.haproxy_health_monitor.health_monitor_enable : true" | mn-
conf set -h local
```

```
$ echo "agent.haproxy_health_monitor.health_monitor_enable : false" | mn-
conf set -h local
```

Additionally, the hosts running the HAProxy instances must have a group called "nogroup" and a user called "nobody", otherwise HAProxy will not be able to start. While this is a default configuration on Ubuntu, on Red Hat you must create this user and group.

How the Health Monitor works in the MidoNet Agent

- The MidoNet Agent does not implement its own health monitor. Instead, it leverages the health checker that is a part of HAProxy.
  - Whenever a user attaches a health monitor to a pool, the MidoNet Agent starts up an HAProxy instance associated with that pool.
  - The HAProxy process then receives information about all of the pool members it must watch.
  - The MidoNet Agent periodically polls HAProxy about the status of its nodes. If it detects a change in status, the MidoNet Agent will update its own database with the status information.
- The following configuration settings (inside mn-conf(1)) tell the MidoNet Agent how to act with respect to health monitoring:
  - health_monitor_enable: true indicates the MidoNet Agent host is eligible to set up HAProxy for health monitoring.
  - namespace_cleanup: true indicates the MidoNet Agent host needs to clean up any left over HAProxy namespaces on the host that are still present after the MidoNet Agent has gone down and is no longer the designated HAProxy host. By default, this is set to false.
  - namespace_suffix: String appended to the end of the namespace names that hold the HAProxy instances. This is to allow you to easily identify the namespaces being created for HAProxy. _MN is the default value.
  - haproxy_file_loc: identifies the location in the file system where the config files for HAProxy will be created. The default value is: /etc/midolman/l4lb/.
- At any given time, there is only one host containing all of the HAProxy instances. This host will be one of the MidoNet Agent hosts that has health_monitor_enable=true defined in its configuration. If this host goes down for any reason, one of the other hosts with the health_monitor_enable=true setting will take over and spawn any required HAProxy instances.

Enabling Health Monitoring in a Pool

To enable health monitoring on a pool you can do perform one of the following:

- Create a health monitor object using the CLI or API server, and set the relevant delay, timeout, and max_retries values (see "Health Monitor" for information).
- Attach the health monitor object to the pool that you want to be monitored. A single health monitor can be attached to any number of pools, but pools may only have a single health monitor.
- Set the admin_state_up on the health monitor object to true.

CLI Example

The example below shows how to use the MidoNet CLI to configure health monitoring.

```
midonet> health-monitor list
midonet> health-monitor create type TCP delay 100 max-retries 50 timeout
 500
hm0
midonet> load-balancer lb0 pool pool0 set health-monitor hm0
midonet> load-balancer lb0 pool pool0 health-monitor show
hm hm0 delay 100 timeout 500 max-retries 50 state down
midonet> health-monitor hm0 pool list
pool pool0 load-balancer lb0 health-monitor hm0 lb-method ROUND_ROBIN state
 up
```

Disabling Health Monitoring

To disable health monitoring on a pool you can do perform one of the following:

- Set the admin_state_up on the health monitor to false. Note that all pools that are using this health monitor will have health monitoring disabled.
- Set the health_monitor_id on the pool to null.
- Delete the health monitor object.

10. Load Balancing as a Service (LBaaS)

Table of Contents

Load Balancers are used to distribute traffic to a set of members based on a configured algorithm. It can be used for fault tolerance, high availability, performance improvements, as well as in many other situations. Load balancers can be configured in a cloud environment (Load Balancer as a Service, or LBaaS).

Neutron LBaaS Support

Neutron and MidoNet has supported LBaaS for some time, but the first implementation of LBaaS was suboptimal, especially for scaling and performance purposes. Thus, Neutron released a second version of the LBaaS standard, and MidoNet now also supports the newer LBaaS V2 model as well as the older, deprecated LBaaS V1 model.

There have been many changes from LBaaS V1 to V2, so this guide will both cover LBaaS V2 from a fresh-starter perspective, as well as offering guides on using V2 for those familiar with those familiar with V1.

Regarding LBaaS configuration, in order for load balancing to work in terms of a virtually defined network, there must be a route from the network on which the load balancer resides to each individual pool member. This also means no network address translation should be performed in between the load balancer and the pool members. Note that this is a general configuration requirement for load balancing, and not only limited to MidoNet.

Neutron LBaaS V1 Specification Support

MidoNet supports the Neutron LBaaS V1 specification (as documented in https://wiki.openstack.org/wiki/Neutron/LBaaS/API_1.0). To set up and use load balancers, users can set up and manipulate load balancers, pools, members, health monitors, etc. via the standard Neutron commands.

MidoNet load balancer limitations

Although MidoNet supports the Neutron LBaaS V1 API, not all Neutron LBaaS features are supported in MidoNet:

- L7 load balancing is not supported.
- There are no pool statistics.
- Only round robin is supported for the load balancer method.
- The Neutron provider model is not supported (for example, assigning a specific provider for each pool is not supported).
- Only one health monitor per pool (the first one in the list).
- Only TCP health check (no UDP, HTTP, also no ICMP/pings).

- Only Source IP session persistence (no cookie or URL).
- You cannot associate a floating IP to a virtual IP address (VIP).
- A VIP must not be on the same subnet as the pool member IF health monitoring is enabled.
- There is no connection limit.
- Only TCP load balancing is supported.
- When using sticky-source IP, directly connecting to the selected back-end host on the same port that is being used by the load balancer is not possible.
- To use an external network for VIPs, it has to be "shared".

Neutron LBaaS V2 Specification Support

MidoNet also supports the Neutron LBaaS V2 specification (as documented in https://wiki.openstack.org/wiki/Neutron/LBaaS/API_2.0). To set up and use load balancers, users can set up and manipulate load balancers, pools, members, health monitors, etc. via the standard Neutron commands as outlined in http://docs.openstack.org/mitaka/networking-guide/config-lbaas.html.

MidoNet load balancer limitations

Although MidoNet supports the Neutron LBaaS V2 API, not all Neutron LBaaS features are supported in MidoNet:

- L7 load balancing is not supported, which means several parameters for Layer 7 options, such as `sni-container-refs` and `default-tls-container-ref`, are not supported. Also, the protocol field for listeners, pools, and health monitor type can only be set as **TCP**.
- Only round robin is supported for the load balancer method.
- Only Source IP session persistence is supported.

11. L2 address masking

Table of Contents

You can use L2 address masking to filter out specific L2 frames, such as multicast frames. This allows you to add a single rule with a specific mask, which reduces the number of rules needed.

Below a couple of important facts about L2 address masking in MidoNet:

- OpenStack security group rules don't match L2 fields, so this feature is provided for customers who directly access/use MidoNet's API and CLI.
- Old rules that do not include values for the hw-dst-mask attribute in the CLI should be interpreted as having the values for this field/attribute set to the default: ffff.ffff.ffff

See Chapter 7, "Rule chains" [28] for information about rule chains, including information about the hw-src-mask and hw-dst-mask attributes used to implement this feature.

L2 address mask rule chain example

This is an example of L2 Address Mask Rule Chain creation.

1. Create a chain (this creates chain alias, "chain0" in the example, pointing to the chain created):

   ```
   create chain name "l2-mask"
   ```

2. Add a rule to the chain that drops traffic with a src (source) MAC not starting with 12:34:56:

   ```
   chain chain0 create rule hw-src !12:34:56:78:90:ab hw-src-mask ffff.
   ff00.0000 type drop
   ```

Example

Here's an example for how you might use hw-dst-mask.

You decide to block (DROP) all multicast traffic on a specific L2 virtual network (bridge). The 8th (least significant) bit of the first (most significant) octet of a MAC address is 1 if the address is multicast, 0 if it's unicast.

However, a MAC broadcast address has all octets set to FF. It's best not to drop broadcast packets,for example, ARP requests. Therefore, you add the following two rules to the pre-bridging rule-chain of the virtual bridge:

- at position 1: ACCEPT if hw-dst is "ff:ff:ff:ff:ff:ff"

  ```
  midonet> chain chain0 add rule hw-dst ff:ff:ff:ff:ff:ff type accept
  ```

- at position 2: DROP if hw-dst has bit 0100.0000.0000 set

  ```
  chain chain0 add rule hw-dst 01:00:00:00:00:00 hw-dst-mask 0100.0000.0000
   pos 2 type drop
  ```

12. Handling fragmented packets

Table of Contents

If you observe that MidoNet is encountering fragmented packets, this section describes how you can handle this situation.

You can configure IP fragment matching in rules to configure how fragmented packets are handled when L4 rules are applied in the virtual topology. This feature allows IP fragments to pass through the virtual topology instead of being dropped. With this feature implementation, you have the following options:

- If you are writing an L2/L3 firewall, you can be IP fragment agnostic: no special handling of IP fragments is required. (L3 NAT handles IP fragments correctly.)
- If you are writing an L4 firewall, you can specify special handling for IP fragments.

Definitions and allowed values

- header = the Condition matches non-fragmented packets AND header fragments. header refers to a non-fragmented packet or to the first fragment (the header fragment) of a fragmented packet. The header fragment is the only one whose IPv4 "fragment offset" field is set to zero. header is the only allowed value if the rule's Condition matches L4 fields OR the rule is dynamic (that is, port modifying) DNAT or SNAT
- nonheader = the Condition matches only non-header fragments. nonheader refers to the second and subsequent fragments.
- any = the Condition matches any fragment AND non-fragmented packets.
- unfragmented = the Condition matches any non-fragmented packet.

Note

If the fragment-policy argument is not set, then it is treated like the setting is any, unless header is the only allowed value.

Note

When you run OpenStack Icehouse against MidoNet with the Condition semantics described above, the handling of fragments will change as follows:

Instead of dropping fragments when they pass through L4 rules, these fragments will be ignored by L4 rules, and therefore, may potentially allow the packets through filters.

A single L4 flow may generate up to two different flows: one to handle non-header fragments, another to handle all other packets.

Fragmented packets rule chain creation example

Create a chain (this creates a rule chain with an alias, "chain0" in the example, pointing to the chain created):

```
create chain name chain0
```

Add a rule to the chain that drops header fragments:

```
chain chain0 add rule fragment-policy header pos 2 header type drop
```

Example 1 Firewall, Does Not Account for Fragmented Packets

The example below only handles non-fragmented packets. These are the firewall rules you start with before you decide to handle fragmented packets.

Initially, you design your firewall to:

- Only allow incoming TCP port 80 (HTTP) traffic
- Drop all other packets

Without addressing fragmented packets, you create a rule chain with the following two rules:

- Rule at position 1
  - By default, this rule matches only non-fragmented packets and header fragments.
  - ACCEPTs packets with protocol=TCP and destination=80.

    ```
    midonet> chain chain0 add rule ethertype 2048 src 0.0.0.0/0 proto 6 in-
    ports router2:port0 dst-port 80 pos 1 type accept
    ```
- Rule at position 2
  - DROPs all packets.

    ```
    midonet> chain chain0 add rule ethertype 2048 src 0.0.0.0/0 dst 0.0.0.
    0/0 in-ports router2:port0 pos 2 type drop
    ```

```
midonet> chain chain0 list rule
rule rule0 ethertype 2048 src 0.0.0.0/0 proto 6 tos 0 dst-port 80 in-ports
 router2:port0 pos 1 type accept
rule rule1 ethertype 2048 src 0.0.0.0/0 dst 0.0.0.0/0 proto 0 tos 0 in-
ports router2:port0 pos 2 type drop
```

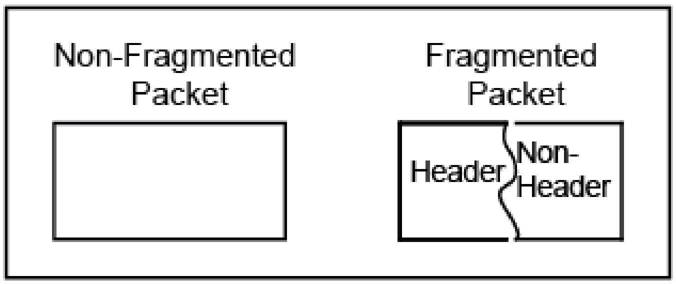
With the above rule chain, MidoNet handles fragmented packet with the destination TCP port 80 as follows:

- The first half of the packet, which contains the TCP header, reaches the rule at position 1, and is accepted.

- However, the second half of the fragmented, which does not have the destination port, reaches the rule at position 1, does not match the rule's condition, and is dropped. This means the fragmented packets do not reach the Web server.

Example 2 Firewall, Addresses Fragmented Packets

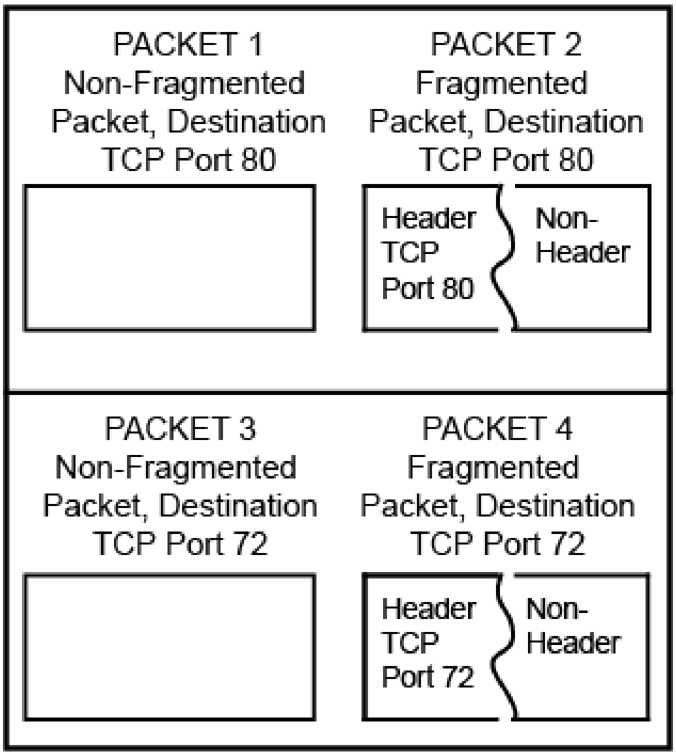
To address this problem, MidoNet provides a mechanism to handle the fragmented packets. This mechanism allows the fragmented packets to reach their destination, as shown in the following example. The drawing below simply depicts a whole, non-fragmented packet and a fragmented packet that consists of two parts, a header and non-header.



Non-Fragmented and Fragmented Packets

For this example, consider the following packets:

- Non-fragmented packet with the destination, TCP port 80
- Fragmented packet with the destination, TCP port 80
- Non-fragmented packet with the destination, TCP port 72
- Fragmented packet with the destination, TCP port 72



Fragmented and Non-Fragmented Packets with Different Destinations

Given the above packets and the rule in example 1, MidoNet processes the packets as follows:

- Packet 1 matches the rule in position 1 and is accepted.
- The header part of packet 2 matches the rule in position 1 and is accepted; the non-header fragment, which doesn't contain the destination, does not match the rule and is dropped.
- Packet 3's destination does not match the rule in position 1 and is dropped, same thing for the header part of packet 4. The non-header part of packet 4 does not contain destination information and is dropped.

The first objective is to accept the part of the packet fragments that contains the headers. To do this, you create the same rule at position 1. The change is to add a new rule at position 2, to drop all packets that contain TCP/UDP headers.

- Rule at position 1
  - By default, this rule matches only non-fragmented packets and header fragments.
  - ACCEPTs packets from in-ports=router2:port0 with protocol=TCP and destination=80.

    ```
    midonet> chain chain18 add rule ethertype 2048 src 0.0.0.0/0 proto 6
     in-ports router2:port0 dst-port 80 pos 1 type accept
    ```
- Rule at position 2
  - Drop packets that contain TCP/UDP headers

    ```
    midonet> chain chain18 add rule ethertype 2048 src 0.0.0.0/0 in-ports
     router2:port0 fragment-policy header pos 2 type drop
    ```
- Rule at position 3
  - Accept all other packets

    ```
    midonet> chain chain18 add rule ethertype 2048 src 0.0.0.0/0 in-ports
     router2:port0 dst 0.0.0.0/0 pos 3 type accept
    ```

Look at the packets in the above figure, starting with the packets destined for port 72, and how they the progress through this new rule chain:

- Packet 3's destination is port 72, not port 80, does not match the rule in position number 1, and continues to the rule at position number 2.
- Packet 3 contains a TCP header and therefore matches the rule at position number 2 and isdropped.
- The header fragment of packet 4 contains a destination of port 72, does not match the rule at position 1 and continues to the rule at position number 2.
- This fragment contains a TCP header, matches the rule at position number 2, and is dropped.

- The non-header fragment of packet 4 does not contain a header (and therefore, has no destination information), does not match the rule at position 1, and continues to the rule at position 2.
- This non-header packet fragment does not contain a TCP/UDP header, does not match the rule at position 2, and continues to the rule at position 3.
- The rule at position 3 accepts all packets that reach it and accepts this packet fragment. Because this packet does not have any associated header information it will not be reassembled and sent to an application and will eventually be dropped.

Looking at packets 1 and 2:

- Packet 1 is destined for TCP port 80, matches the rule at position 1, and is accepted.
- For packet 2, the packet fragment with the header contains a destination of TCP port 80, matches the rule at position 1, and is accepted.
- The non-header packet fragment of packet number 2 does not contain a header, does not match the rule at position number 1, and continues to the rule at position number 2.
- This non-header packet fragment does not contain a TCP/UDP header, therefore does not match the rule at position number 2, is not dropped, and continues to the rule at position number 3.
- The rule at position number 3 accepts all packets, so this packet fragment is accepted.

This change allows non-header fragments to get past both the rules at positions 1 and 2, and exit the chain with an ACCEPT. With this change, the firewall now lets all non-header fragments through, but you decided that the level of risk is acceptable and are just trying to fix the broken HTTP flows. This is not a problem, as an unwanted non-header fragment will be discarded if the corresponding header fragment is never received.

13. MidoNet resource protection

Table of Contents

This section describes how to protect the MidoNet Agent against DOS attacks carried out by potentially rogue VMs.

Introduction

MidoNet provides resource protection/isolation among tenants running VMs on the same hypervisor. Specifically, MidoNet provides protection against a VM initiating a Denial of Service (DoS) attack against the MidoNet Agent by emitting packets that miss the kernel flow table as fast as possible. Without resource protection, new flows to/from other VMs are prevented from being handled in a timely manner or at all because the rogue VM would capture most of the Agent's capacity to process packets. In a public cloud setting, where tenants are not necessarily trusted, this may be considered a serious problem.

Expected Behavior

There are two major requirements that this solution addresses:

- VMs are guaranteed to have their fair share of the MidoNet Agent's processing capacity. Even if another VM sends packets at a rate that exceeds the total capacity of the Agent, other VMs will still be able to set up new flows at the expected rate and latency.

- The Agent fairly redistributes the processing capacity allocated to VMs that are not fully using it.

The MidoNet Agent applies a hierarchical token bucket (HTB) to all packets that miss the kernel flow table and go up to userspace, so as to fairly share processing capacity among all ports.

The HTB is set up in such a way that tunnel traffic between MidoNet hosts and through a VTEP is guaranteed 50% of the resources while the remaining 50% is distributed among VM ports.

At the top of the hierarchy is a bucket whose size defines the total burst capacity of the system. Below this bucket lie three other buckets, one for tunnel traffic, another for VTEP traffic and another for VM traffic. They evenly share processing capacity, with the tunnel traffic and VTEP buckets having a non-zero size. Below the VM-shared bucket there is a bucket for every VM. The VM-shared bucket has zero capacity, meaning that it accumulates no tokens: if all the VM leaf buckets are full, excess capacity goes to either the tunnel traffic or VTEP buckets, or to the top-level bucket.

Configuration

You can specify resource protection configuration parameters using mn-conf(1), in the agent.datapath section. The available parameters are:

- global_incoming_burst_capacity - this sets the size of the top-level bucket and also defines the total number of tokens in the system (corresponding to in-flight packets) that will be divided among the different levels in the HTB; the rate at which tokens are placed back in the bucket is a function of the rate at which they are processed.
- tunnel_incoming_burst_capacity - this sets the capacity of the bucket associated with tunnel traffic, enforcing the rate at which a MidoNet Agent can communicate with the other Agents.
- vm_incoming_burst_capacity - this sets the capacity of each VM leaf bucket, which is below the shared VM bucket. This parameter enforces the rate at which individual VMs can sendtraffic.
- vtep_incoming_burst_capacity - this sets the capacity of the bucket associated with the VxLAN VTEP functionality, which enforces the rate at which the MidoNet Agent can communicate with the VxLAN domain.

See the mn-conf(1) schemas for more information about the above parameters.

Recommended values for these properties depend on the role of the MidoNet host (Gateway vs. Compute Node) and interaction with other resource-related properties, like JVM-memory and flow-table size. Midolman RPM and Debian packages include versions of each configuration tuned for Compute/Gateway hosts respectively. You can find these configurations in /etc/midolman, alongside the default configuration files. See "Recommended Values" for a table of recommended values.

Disabling Resource Protection

You can disable the resource protection feature.

To disable resource protection:

1. Specify a size value of "0" for all the parameters described in the Configuration section, except for the global_incoming_burst_capacity parameter.

This will cause all tokens to accumulate in the global bucket and all the traffic will be distributed from this single bucket.

14. MidoNet monitoring

Table of Contents

MidoNet is composed of various services; each service exposes a variety of metrics that can be fetched from typical monitoring services.

This chapter describes the main available metrics for each service.

Metering

Note: This feature is in **experimental** status.

Overview

The goal of metering is to provide packets and bytes traffic counters for arbitrary slices of the traffic that travels through MidoNet.

A meter is a counter of bytes and packets, associated with a name. In order to be incremented, the meter needs to have flows associated with it. MidoNet agents will automatically associate flows with certain meters, and users can create their own custom meters setting the `meterName` attribute in chain rules.

For example, all traffic going through bridge with uuid F00 in MidoNet be counted under meter `meters:device:F00`. All traffic egressing port BAR will also be reflected in meter `meters:port:tx:BAR`.

MidoNet agents offer these counters for their partial view of overlay traffic. In other words, each agent will provide meters that only account for the traffic that agent has simulated. For a given meter, the MidoNet-wide real value is the sum of the value of this meter across all agents.

Note: Metering data is meant to be polled and stored by a monitoring layer onto a time series database. Thus agents don't persist the metering data they gather, and meter values will reset to zero when an agent reboots. **Any metering data collection layer should account for this effect and detect counter resets.**

Querying meters

Agents publish meters over JMX and a command line tool, `mm-meter`, uses their JMX interface to list, fetch and monitor meter values.

For example code on the JMX interface, the best source is the code of `mm-meter` itself.

Querying meters with `mm-meter` is very simple:

```
$ mm-meter --help
```

```
  -h, --host  <arg>   Host (default = localhost)
  -p, --port  <arg>   JMX port (default = 7200)
      --help          Show help message

Subcommand: list - list all active meters
      --help   Show help message
Subcommand: get
  -n, --meter-name  <arg>   name of the meter
      --help                Show help message

 trailing arguments:
  delay (not required)   delay between updates, in seconds. If no delay is
                       specified, only one report is printed. (default = 0)
  count (not required)   number of updates, defaults to infinity
                       (default = 2147483647)
```

The `list` command will print a list of all meters known to this agent:

```
$ mm-meter list
meters:user:port0-on-the-bridge
meters:port:rx:cf453c9d-94c4-4c27-ba32-529b7cbacf1d
meters:device:845a54bf-b702-4dc2-8958-bbe7156bc4ef
meters:port:tx:cf453c9d-94c4-4c27-ba32-529b7cbacf1d
meters:port:tx:9182485b-8f86-462d-a8be-62586060eeb9
meters:device:f0d1f093-2de7-49a1-a5ec-898f94769e34
meters:device:9182485b-8f86-462d-a8be-62586060eeb9
meters:port:rx:9182485b-8f86-462d-a8be-62586060eeb9
meters:device:cf453c9d-94c4-4c27-ba32-529b7cbacf1d
```

And the `get` command will print the *current*, *local* counters for a meter. It takes a delay, in which case it will poll the meter and print deltas periodically:

```
$ mm-meter get -n meters:port:rx:cf453c9d-94c4-4c27-ba32-529b7cbacf1d 10
    packets         bytes
     568935    4215888475
          0             0
          0             0
         23          5834
          0             0
```

Creating a custom meter

Operators may want to meter a custom slice of their virtual network traffic. This is possible by matching on that slice using one or several chain rules in the virtual topology. The `meterName` property in chain rules will assign matching flows to the meter referred to by its value, in addition to the meters that flow would naturally feed.

Besides using the REST API, operators can use `midonet-cli` to set up such rules. The following rule will assign to meter `my-meter` all traffic that hits the rule after having traversed device `9182485b-8f86-462d-a8be-62586060eeb9`:

```
midonet> chain chain0 list rule
rule rule0 proto 0 tos 0 traversed-device 9182485b-8f86-462d-
a8be-62586060eeb9 fragment-policy any pos 1 type accept meter my-meter
```

Note that, when inspecting meters `my-meter` will turn into `meters:user:my-meter`, to avoid naming conflicts with built-in meters.

Monitoring Network State Database

The Network State Database is deployed with ZooKeeper and optionally, Cassandra, instances. Both offer JMX bindings.

The configuration provided with MidoNet uses only a subset of the most relevant for our use cases. Details in the sections below provide additional information about the metrics configured by MidoNet's deployment scripts, as well as an explanation about what to watch.

JXM Metrics

In addition to monitoring ZooKeeper and Cassandra directly, both the MidoNet Agent and the MidoNet Cluster expose a set of internal metrics that you can use to monitor the performance and health of the connection to the ZooKeeper Network State Database, as well as gaining visibility into various statistics such as the number of reads and writes.

These metrics are available using JMX in the `metrics` domain. The name of each JMX MBean is prefixed by `org.midonet.cluster.monitoring.metrics`. To access a particular metric, prepend the above prefix to the metric names listed below. For example, the full name of a JMX metric is:

```
metrics:name=org.midonet.cluster.monitoring.metrics.StorageGauge.
connectionState
```

General Metrics

These metrics are gauges for the current ZooKeeper connection state and open observables to the topology objects.

Name	Description
`StorageGauge.connectionState`	The state of the connection to ZooKeeper used for normal reading and writing.
`StorageGauge.failFastConnectionState`	The state of the fail fast connection to ZooKeeper, which is used by some services to detect whether a certain node has lost connectivity to the NSDB with a low latency.
`StorageGauge.objectObservables`	The number of open observables for topology objects.
`StorageGauge.classObservables`	The number of open observables for topology classes.
`StorageGauge.objectObservables.*`	The number of open observables for topology objects of a particular class.

Error Metrics

These metrics expose counters for errors that occurred while writing or reading the topology or state data in ZooKeeper.

Name	Description
`StorageCounter.concurrentModificationException`	The number of exceptions thrown because a topology object was modified concurrently by two or more clients.
`StorageCounter.conflictException`	The number of exceptions thrown because an attempt was made to modify a topology object that conflicted with the current references to/from that object.
`StorageCounter.objectReferencedException`	The number of exceptions thrown because an attempt was made to delete a topology object that was still referenced by another object.
`StorageCounter.objectExistsException`	The number of exceptions thrown because an attempt was made to create a topology object with an identifier that already exists.
`StorageCounter.objectNotFoundException`	The number of exceptions thrown because an attempt was made to access (update or delete) a topology object that does not exist.
`StorageCounter.nodeExistsException`	The number of exceptions thrown because an attempt was made to create a ZooKeeper node that already exists.
`StorageCounter.nodeNotFoundException`	The number of exceptions thrown because an attempt was made to access (update or delete) a ZooKeeper node that does not exist.

Name	Description
`StorageCounter.objectObservableClosed`	The number of occurrences where at attempt was made to read a topology object via an observable stream that was already closed. This measures the degree of concurrent access to the same objects and it is automatically recovered by opening a new observable.
`StorageCounter.classObservableClosed`	The number of occurrences where an attempt was made to read all the objects in a topology class via an observable stream that was already closed. This measures the degree of concurrent access to the same objects and it is automatically recovered by opening a new observable.
`StorageCounter.stateObservableClosed`	The number of occurrences where an attempt was made to read the state of a topology object via an observable stream that was already closed. This measures the degree of concurrent access to the same object state and it is automatically recovered by opening a new observable.
`StorageCounter.objectObservableError`	The number of unrecoverable errors that were emitted via the observable stream of a topology object.
`StorageCounter.classObservableError`	The number of unrecoverable errors that were emitted via the observable stream of a topology class.
`StorageCounter.stateObservableError`	The number of unrecoverable errors that were emitted via the observable stream of a topology object state.

Performance Metrics

These metrics expose information about the ZooKeeper connection changes and data operations.

Name	Description
`StorageMeter.connectionsCreated`	A meter for the number of created ZooKeeper connections.
`StorageMeter.connectionsLost`	A meter for the number of lost ZooKeeper connections.
`StorageTimer.read`	A timer for the number of ZooKeeper node reads.
`StorageTimer.readChildren`	A timer for the number of ZooKeeper node children reads.
`StorageTimer.write`	A timer for the number of ZooKeeper node writes.
`StorageTimer.multi`	A timer for the number of ZooKeeper transaction writes.
`StorageHistogram.stateTableReadLatency`	A histogram for the latency of reading the entire list of entries of a state table from ZooKeeper.
`StorageHistogram.stateTableAddLatency`	A histogram for the latency of adding a new entry to a state table by writing directly to ZooKeeper.
`StorageHistogram.stateTableRoundTripLatency`	A histogram for the latency between adding a new entry to a state table and receiving a notification the entry was accepted by the NSDB. When the *state proxy* service is disabled, this latency measures the round-trip refresh of a state table via ZooKeeper. When the *state proxy* service is enabled, this latency measures the delay of writing the entry to ZooKeeper and reading the updated table via the state table proxy.

Session Metrics

These metrics expose statistics for the ZooKeeper connection.

Name	Description
`StorageHistogram.timeConnected`	A histogram for the duration of the ZooKeeper connection.
`StorageHistogram.timeDisconnected`	A histogram for the duration of the ZooKeeper connection loss.

Watcher Metrics

These metrics expose statistics for the watcher triggered by changes to the topology or state data from ZooKeeper.

Name	Description
`StorageCounter.nodeTriggeredWatchers`	The number of triggered watchers for changes to ZooKeeper node data.

Name	Description
StorageCounter.childrenTriggeredWatchers	The number of triggered watchers for changes to ZooKeeper node children.

Cassandra

By default, Cassandra uses port 7199 for JMX service from all its nodes and you can connect using jconsole for a comprehensive view.

Additionally, Cassandra's own nodetool utility offers commands like cfstats and tpstats that allow access to valuable stats into keyspaces, tables, column families, and so on on a given node.

For a rich reference into Cassandra monitoring, visit the official documentation (go to http://www.datastax.com/, and search for "monitoring a Cassandra cluster").

Cache Reqs vs. Hits

This is self-descriptive, ideally you want the cache hits to be as close to the requests as possible. Note that by default MidoNet Cassandra nodes only enable the Partition Key Cache, but not Row Cache, so it's normal that these stay at 0. For MidoNet the Partition Key Cache should effectively be very similar to the Row Key Cache because our column families (CF) have only one column and therefore rows are not spread across several SSTables.

Compactions

This indicates the number of bytes being compacted. Typical workloads will present regular small spikes when the minor compaction jobs are run, and infrequent large spikes when major compactions are run. A large number of compactions indicates the need to add capacity to the cluster.

Internal Tasks

These are internal Cassandra tasks. The most important are:

- Gossip: MidoNet's Cassandra nodes are expected to spend a fair amount of their time busy in Gossip (wherein state information transfers among peers).
- MemTable Post Flusher: memtable flushes that are waiting to be written to the commit log. These should be as low as possible, and definitely not sustained.
- Hinted Handoff tasks: the appearance of these tasks indicates cases where replicas are detected as unavailable, so non-replica nodes need to temporarily store data until the replicas become available. Frequent Hinted Handoff spikes may hint at nodes being partitioned from the cluster.
- Anti-Entropy spikes: indicate data inconsistencies detected and being resolved.
- Stream activity: involves transferring or requesting data from other nodes. Ideally these should be infrequent and short-spaced.

Messaging Service Tasks

These are tasks received and responded to each of the peer nodes. Expect an even distribution with all peers.

NAT Column Family Latency

This is a key metric that informs about the read and write latency to the NAT mapping cache. High latency, especially in reads, is problematic because it causes high latency in traffic traversing those virtual routers that apply the NAT rules. Due to Cassandra's own guarantees, write latency can be expected to be lower. Note that higher replication levels have a significant impact in latency (nodes have to retrieve/receive ACK from n replicas). Spikes in latency can be correlated to events like compactions, especially in cache misses, because Cassandra needs to go to disk to fetch data during high I/O load due to compactions.

NAT Column Family Memtable

Shows data size and column count. This is in-memory data. Expect a see-saw pattern because most of the data expires after the mapping time to lives (TTLs) expire.

NAT Column Family Disk Usage

Shows overall disk usage, including that used for the bloom filters used to save trips to cache when keys are not present.

NAT Column Family Ops

Shows read and writes on each node. The aggregated views are probably more valuable to help you spot bad distribution of load across the cluster.

Node Load

Shows the disk space used in the node.

Number of Nodes in Cluster

This is the view each node has of the rest of the cluster, and can help you spot partitions.

Request Task Completed by Stage

Shows tasks completed by the node. Mutations are changes in data, request responses are data sent to requesting peers. The Read-repair task appears as a result of nodes detecting inconsistent data and asking to perform a read to update the data; obviously, this task should be as infrequent as possible.

StorageProxy Operation Count

Shows overall read/write operations in the node.

StorageProxy Recent and Total Latency

Shows overall read/write latency in the cluster. Watch for big discrepancies between NAT Column Family (CF) Latency and this metric, because that would help determine whether problems are related to a single CF or the entire storage. More importantly: it indicates what features may be impacted in Midolman agents.

- JVM garbage-collection (GC) times
- JVM Heap Summary
- JVM Non-Heap Summary

Descriptions of these graphs are beyond the scope of this guide, but high JVM GC times are the best indication that Cassandra may be consuming too much time on garbage collection. This will correlate with high latency accessing Midolman's column families, which will propagate to Midolman. The Midolman agent will increase simulation latency and also degrade the utilization of CPU resources (experiencing more idle time while waiting for responses from Cassandra).

ZooKeeper

You can find ZooKeeper stats in the "zookeeper" category of the MidoStorage group. ZooKeeper classifies metrics in separate MBeans for leader/follower, so each node will report some values twice, one in the "Follower" role, another in the "Leader" role. Take into account that a given node may change roles (for example, if the leader shuts down, a follower node may be promoted to leader). You can easily spot these events. For example, the line in the "Connection Count as Follower" will suddenly blank out, and another will appear in the "Connection Count as Leader" graph.

Connection Count (as Follower/Leader)

These two graphs display the number of live connections to this node in its role at a givenpoint in time.

In Memory Data Tree (as Follower/Leader)

Exposes the size of the in-memory znode database, both data nodes and watch count.

Latency (as Follower/Leader)

Exposes the average and maximum latency experienced in connections.

Packet Count (as Follower/Leader)

Exposes the count of packets sent/received by the node in its role at a given point in time.

Quorum Size

Exposes each node's view of the number of nodes agreeing on the leader's election.

ZooKeeper also exposes some information about each specific connection, this may be useful to watch for troubleshooting. Using jconsole (go to http://www.oracle.com/technetwork/java/index.html and search on "jconsole" for information), you can:

1. Connect to any ZooKeeper node at port 9199.
2. Navigate to org.apache.ZooKeeperService, ReplicatedServer_idX.
3. Choose the desired replica.
4. Go into Leader or Follower, Connections to see a list of IP addresses of the connected clients. Information shown here includes:
   - latency
   - packet sent/received count
   - session ID, etc. for that specific client.

Some of the MBeans whose values are exposed in our graphs also contain (computationally intensive) operations that also offer interesting information. Using jconsole, expand org.apache.ZooKeeperService, and then the appropriate replica, and either Leader or Follower, according to its role:

- InMemoryDataTree.approximateDataSize: tells the size of the in-memory data store.
- InMemoryDataTree.countEphemerals: tells the count of ephemeral nodes.

The installation script also provides graphs to monitor the state of ZooKeeper's JVM:

- JVM GC times
- JVM Heap Summary
- JVM Non-Heap Summary

Descriptions of these graphs are beyond the scope of this document, but high JVM GC times are the best indication that ZooKeeper may be the source of problems in Midolman, which will be exhibited in high latency and under-utilization of CPU resources. ZooKeeper uses the Parallel collector, and the JVM GC times track the time of the last collection in all spaces from the java.lang:type=GarbageCollector,name=PS Scavenge MBean, which deals with all generations.

Monitoring Midolman Agents

MidoNet Agents expose a set of internal metrics that you can use to monitor the performance and health of agent nodes.

These metrics are available using JMX in the `metrics` domain. The name of each JMX MBean is prefixed by `org.midonet.midolman.monitoring.metrics`. To access a particular metric, prepend the above prefix to the metric names listed below. For example, the full name of a JMX metric is:

```
metrics:name=org.midonet.midolman.monitoring.metrics.DatapathMeter.flows.
created
```

Additionally, some graphs are provided to monitor the state of the JVM running the MidoNet Agent.

Datapath Metrics

These metrics include meters for the created and deleted datapath flows, and errors that occur while changing these flows.

Name	Description
`DatapathMeter.flows.created`	A meter for the created datapath flows.
`DatapathMeter.flows.createErrors`	A meter for the errors occurring during the creation of a datapath flow.
`DatapathMeter.flows.createDupeErrors`	A meter for the errors occurring during the creation of a duplicate datapath flow.
`DatapathMeter.flows.deleted`	A meter for the deleted datapath flows.
`DatapathMeter.flows.deleteErrors`	A meter for the errors occurring during the deletion of a datapath flow.

Netlink Metrics

These metrics include meters for Netlink messages.

Name	Description
`NetlinkMeter.notifications`	A meter for the notifications received from the OVS datapath via Netlink.

Name	Description
`NetlinkMeter.errors`	A meter for the errors received from the OVS datapath via Netlink.
`NetlinkMeter.htbDrops`	A meter for the packets dropped because of the MidoNet Agent rate limiting mechanism.

Packet Pipeline Metrics

These metrics expose counters and meters for the simulation packet pipeline. The MidoNet Agent exposes certain metrics for each packet worker, where the worker index is included in the metric name. For example, when configuring MidoNet to use two (2) packet workers, the following distinct metrics are available: `PacketPipelineHistogram.worker-0.packetsProcessed` and `PacketPipelineHistogram.worker-1.packetsProcessed`.

Global Metrics

Name	Description
`PacketPipelineCounter.contextsAllocated`	The number of allocated packet context. A packet context is an internal variable that tracks the simulation progress for a packet that is being processed. Since each packet context requires a certain amount of memory, the counter reflects the memory consumption of the MidoNet agent when processing the user-space packets.
`PacketPipelineCounter.contextsPooled`	The number of packet contexts that have been pooled to be reused on subsequent packets.
`PacketPipelineCounter.contextsBeingProcessed`	The number of packet contexts corresponding to packets that are currently simulated in the pipeline.

Packet Worker Metrics

Name	Description
`PacketPipelineCounter.worker-{id}.packetsOnHold`	The number of packets on hold, that are waiting for the agent to read the virtual topology data from the Network State Database.
`PacketPipelineMeter.worker-{id}.packetsPostponed`	A meter for the packets that were postponed because their corresponding virtual topology devices was not yet available at the agent.
`PacketPipelineMeter.worker-{id}.packetsDropped`	A meter for the packets dropped during simulation. These packets are dropped because the current virtual topology does not allow them to reach their intended destination.
`PacketPipelineHistogram.worker-{id}.packetsProcessed`	A histogram for the simulation latency of the processed packets. This value is fundamental for network latency.
`PacketPipelineHistogram.executor-{id}.packetsExecuted`	A histogram for the packet execution latency, which includes sending back a processed packet to the OVS datapath.
`PacketPipelineMeter.worker-{id}.statePacketsProcessed`	A meter for the processed packets that carried flow state data.
`PacketPipelineMeter.worker-{id}.packetQueue.overflow`	A meter for the packet queue overflows.
`FlowTablesGauge.worker-{id}.currentDatapathFlows`	A gauge for the current datapath flows.
`FlowTablesMeter.worker-{id}.datapathFlowsCreated`	A meter for created datapath flows.
`FlowTablesMeter.worker-{id}.datapathFlowsRemoved`	A meter for removed datapath flows.

Topology Metrics

The metrics measure the virtual topology cached by a given MidoNet Agent. The virtual topology consists of virtual devices, such as bridges, routers, ports, chains, rules, etc. When a device is requested in order to process a packet that must traverse or otherwise requires that device, the MidoNet Agent loads the device from the Network State Database, and caches it for later usage. In addition, the agent keeps a notification stream open to the NSDB for that devices, such that future changes of the device are notified to the agent, updating its internal cache and invalidating any flows that have previously been established using that device's older configuration.

Some virtual topology metrics are global, whereas others are reported for each device class, which can be one of the following: `Bridge`, `Chain`, `Host`, `IPAddrGroup`, `LoadBalancer`, `Mirror`, `Pool`, `PoolHealthMonitorMap`, `Port`, `PortGroup`, `Router`, `RuleLogger` and `TunnelZone`.

Global Metrics

Name	Description
`VirtualTopologyGauge.devices`	A gauge for the number of devices stored in the virtual topology cache.
`VirtualTopologyGauge.observables`	A gauge for the number of open observable streams that report changes for the devices stored in the virtual topology cache.
`VirtualTopologyGauge.cacheHit`	A gauge for the cache hits when a device was requested by the packet pipeline when simulating a packet. A cache hit means that the device was available immediately, and the packet simulation was not interrupted.
`VirtualTopologyGauge.cacheMiss`	A gauge for the cache misses when a device was requested by the packet pipeline when simulating a packet. A cache miss means that the device needed to be loaded from the NSDB, requiring the packet simulation to be postponed until the device became available. Cache misses lead to higher packet processing latencies.
`VirtualTopologyCounter.deviceUpdate`	The number of updates the virtual topology caches receives for all cached devices.
`VirtualTopologyCounter.deviceError`	The number of errors the virtual topology caches receives for all cached devices.
`VirtualTopologyCounter.deviceComplete`	A counter for the cached devices that were deleted.
`VirtualTopologyMeter.deviceUpdate`	A meter for the device updates received by the virtual topology for the cached devices.
`VirtualTopologyMeter.deviceError`	A meter for the device errors received by the virtual topology for the cached devices.
`VirtualTopologyMeter.deviceComplete`	A meter for the devices deleted from the virtual topology.
`VirtualTopologyHistogram.deviceLatency`	A histogram with the latency of loading a device from the NSDB.
`VirtualTopologyHistogram.deviceLifetime`	A histogram with the lifetime of a device in the virtual topology cache.

JVM Non-Heap Summary

Shows off-heap memory usage, which consists of mainly buffer pools used for messages to/from the Netlink layer.

JVM Heap Summary

Shows per-generation stats. The MidoNet Agent has very specific memory-usage constraints because it aims for a low memory and CPU footprint. At the same time, simulations generate a significant amount of short-lived garbage.

- The Eden is configured as the largest generation trying to hold as much garbage as possible. However, it is likely that it fills up frequently under high traffic, which may imply that some short-lived objects get promoted to the old generation and garbage is collected soon afterward.
- The Old Generation is expected to contain a baseline of long-lived objects that get reused during simulations. An amount of short-lived objects may also be pushed from the young generation, eventually also filling the old generation and triggering a GC event that will collect them. This will show as a see-saw pattern in the "Old used". The

see-saw should converge to oscillating between stable maximum/minimum values on top of the long-lived objects baseline.

  - Large spikes indicate higher CPU consumption in GC and are usually associated with a certain throughput degradation. You may slightly alleviate this by increasing the size of the Eden.

- JVM GC times: shows the duration of the last garbage collection performed by the JVM G1 collector. It is closely associated with the see-saw pattern described above.

  - Note that these times do not stop the application completely, because part of the work is done concurrently. The main impact is in CPU "stolen" from the agent.

CPU Usage

Under high traffic, the MidoNet Agent should tend to saturate all CPUs, reflecting in the graph as high "user" utilization and little or no "idle". Note that "user" may include other processes, so especially in Gateway Nodes, you should verify that only the agent is consuming most of CPU time dedicated to user processes. High "system", "iowait" indicators are clear indication of high load, excessive context switching, and contention or other problems on the host.

Monitoring events

This section describes how MidoNet's event system works so you can monitor the system's day-to-day operations.

Overview

This section provides a high-level overview of the information covered regarding event monitoring.

Categories of Event Messages

Below are the following types of events defined in the MidoNet system:

- Changes in the virtual topology
- Events concerning the MidoNet API server, including:
  - Changes to the connection status to the Network State Database
- Events concerning MidoNet Agents, including:
  - Changes to the connection status to the Network State Database
  - Changes affecting the status of network interfaces (for example, physical network interfaces and taps)
  - Daemons starting and exiting

Configuration

Each event message is generated with logback (http://logback.qos.ch/).

The configuration files are located at the following locations, depending on the type of the node.

Table 14.1. Configuration Files/Locations

Type of Node	Location of the Configuration File
MidoNet Network Agent	/etc/midolman/logback.xml
MidoNet Cluster server	/etc/midonet-cluster/logback.xml

Below are the behaviors with the default configuration shipped with the MidoNet release, but you can configure the behaviors as you like. See http://logback.qos.ch/manual/index.html for instructions on how to configure the logback.xml file.

Event log files locations

Event messages are stored locally on the filesystem in a separate file, in addition to the ordinary log file.

Table 14.2. Event Message Files/Locations

Type of Node	Location
MidoNet Network Agent	/var/log/midolman/midolman.event.log
MidoNet API server	/var/log/tomcat6/midonet-api.event.log (on Red Hat) /var/log/tomcat7/midonet-api.event.log (on Ubuntu)

Tip

In addition to midolman.event.log, /var/log/midolman/midolman.log contains additional debug information. You do not normally need to use it, but it may contain useful troubleshooting information.

Message format

By default, event messages have the following format.

```
<pattern>%d{yyyy.MM.dd HH:mm:ss.SSS} ${HOSTNAME} %-5level %logger - %m%n
%rEx </pattern>
```

See http://logback.qos.ch/manual/layouts.html for details about the above placeholders.

List of event messages

This section lists the event messages.

The event messages are organized in the following major categories:

- Virtual topology events
- API server events
- MidoNet Agent events

Virtual topology events

This section describes the messages associated with virtual topology events.

Router

Logger	org.midonet.event.topology.Router.CREATE

Message	CREATE routerId={0}, data={1}.
Level	INFO
Explanation	Router with routerId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Router.UPDATE
Message	UPDATE routerId={0}, data={1}.
Level	INFO
Explanation	Router with routerId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Router.DELETE
Message	DELETE routerId={0}.
Level	INFO
Explanation	Router with routerId={0} was deleted.
Corrective Action	N/A

Logger	org.midonet.event.topology.Router.ROUTE_CREATE
Message	ROUTE_CREATE routerId={0}, data={1}.
Level	INFO
Explanation	Route={1} was created in routerId={0}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Router.ROUTE_DELETE
Message	ROUTE_DELETE routerId={0}, routeId={1}.
Level	INFO
Explanation	routeId={1} was deleted in routerId={0}.
Corrective Action	N/A

Bridge

Logger	org.midonet.event.topology.Bridge.CREATE
Message	CREATE bridgeId={0}, data={1}.
Level	INFO
Explanation	Bridge with bridgeId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bridge.UPDATE
Message	UPDATE bridgeId={0}, data={1}.
Level	INFO
Explanation	Bridge with bridgeId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bridge.DELETE
Message	DELETE bridgeId={0}.
Level	INFO
Explanation	Bridge with bridgeId={0} was deleted.
Corrective Action	N/A

Port

Logger	org.midonet.event.topology.Port.CREATE

Message	CREATE portId={0}, data={1}.
Level	INFO
Explanation	Port with portId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.UPDATE
Message	UPDATE portId={0}, data={1}.
Level	INFO
Explanation	Port with portId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.DELETE
Message	DELETE portId={0}.
Level	INFO
Explanation	Port with portId={0} was deleted.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.LINK
Message	LINK portId={0}, data={1}.
Level	INFO
Explanation	Port with portId={0} was linked.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.UNLINK
Message	UNLINK portId={0}, data={1}.
Level	INFO
Explanation	Port with portId={0} was unlinked.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.BIND
Message	BIND portId={0}, data={1}.
Level	INFO
Explanation	Port with portId={0} was bound.
Corrective Action	N/A

Logger	org.midonet.event.topology.Port.UNBIND
Message	UNBIND portId={0}.
Level	INFO
Explanation	Port with portId={0} was unbound.
Corrective Action	N/A

Chain

Logger	org.midonet.event.topology.Chain.CREATE
Message	CREATE chainId={0}, data={1}.
Level	INFO
Explanation	Chain with chainId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Chain.DELETE

Message	DELETE chainId={0}.
Level	INFO
Explanation	Chain with chainId={0} was deleted.
Corrective Action	N/A

Rule

Logger	org.midonet.event.topology.Rule.CREATE
Message	CREATE ruleId={0}, data={1}.
Level	INFO
Explanation	Rule with ruleId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Rule.DELETE
Message	DELETE ruleId={0}.
Level	INFO
Explanation	Rule with ruleId={0} was deleted.
Corrective Action	N/A

Tunnel Zone

Logger	org.midonet.event.topology.TunnelZone.CREATE
Message	CREATE tunnelZoneId={0}, data={1}.
Level	INFO
Explanation	TunnelZone with tunnelZoneId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.TunnelZone.UPDATE
Message	UPDATE tunnelZoneId={0}, data={1}.
Level	INFO
Explanation	TunnelZone with tunnelZoneId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.TunnelZone.DELETE
Message	DELETE tunnelZoneId={0}.
Level	INFO
Explanation	TunnelZone with tunnelZoneId={0} was deleted.
Corrective Action	N/A

Logger	org.midonet.event.topology.TunnelZone.MEMBER_CREATE
Message	MEMBER_CREATE tunnelZoneId={0}, data={1}.
Level	INFO
Explanation	TunnelZone member={1} was added to tunnel-ZoneId={0}.
Corrective Action	N/A

Logger	org.midonet.event.topology.TunnelZone.MEMBER_DELETE
Message	MEMBER_DELETE tunnelZoneId={0}, data={1}.
Level	INFO
Explanation	TunnelZone member={1} was deleted from tunnel-ZoneId={0}.

Corrective Action	N/A

BGP

Logger	org.midonet.event.topology.Bgp.CREATE
Message	CREATE bgpId={0}, data={1}.
Level	INFO
Explanation	Bgp with bgpId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bgp.UPDATE
Message	UPDATE bgpId={0}, data={1}.
Level	INFO
Explanation	Bgp with bgpId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bgp.DELETE
Message	DELETE bgpId={0}.
Level	INFO
Explanation	Bgp with bgpId={0} was deleted.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bgp.ROUTE_CREATE
Message	ROUTE_CREATE bgpId={0}, data={1}.
Level	INFO
Explanation	Route={1} was added to bgpId={0}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Bgp.ROUTE_DELETE
Message	ROUTE_DELETE bgpId={0}, data={1}.
Level	INFO
Explanation	Route={1} was deleted from bgpId={0}.
Corrective Action	N/A

LoadBalancer

Logger	org.midonet.event.topology.LoadBalancer.CREATE
Message	CREATE loadBalancerId={0}, data={1}.
Level	INFO
Explanation	LoadBalancer with loadBalancerId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.LoadBalancer.UPDATE
Message	UPDATE loadBalancerId={0}, data={1}.
Level	INFO
Explanation	LoadBalancer with loadBalancerId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.LoadBalancer.DELETE
Message	DELETE loadBalancerId={0}.

Level	INFO
Explanation	LoadBalancer with loadBalancerId={0} was deleted.
Corrective Action	N/A

VIP

Logger	org.midonet.event.topology.VIP.CREATE
Message	CREATE vipId={0}, data={1}.
Level	INFO
Explanation	VIP with vipId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.VIP.UPDATE
Message	UPDATE vipId={0}, data={1}.
Level	INFO
Explanation	VIP with vipId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.VIP.DELETE
Message	DELETE vipId={0}.
Level	INFO
Explanation	VIP with vipId={0} was deleted.
Corrective Action	N/A

Pool

Logger	org.midonet.event.topology.Pool.CREATE
Message	CREATE poolId={0}, data={1}.
Level	INFO
Explanation	Pool with poolId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.Pool.UPDATE
Message	UPDATE poolId={0}, data={1}.
Level	INFO
Explanation	Pool with poolId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.Pool.DELETE
Message	DELETE poolId={0}.
Level	INFO
Explanation	Pool with poolId={0} was deleted.
Corrective Action	N/A

PoolMember

Logger	org.midonet.event.topology.PoolMember.CREATE
Message	CREATE poolMemberId={0}, data={1}.
Level	INFO
Explanation	PoolMember with poolMemberId={0} was created.

Corrective Action	N/A

Logger	org.midonet.event.topology.PoolMember.UPDATE
Message	UPDATE poolMemberId={0}, data={1}.
Level	INFO
Explanation	PoolMember with poolMemberId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.PoolMember.DELETE
Message	DELETE poolMemberId={0}.
Level	INFO
Explanation	PoolMember with poolMemberId={0} was deleted.
Corrective Action	N/A

HealthMonitor

Logger	org.midonet.event.topology.HealthMonitor.CREATE
Message	CREATE healthMonitorId={0}, data={1}.
Level	INFO
Explanation	HealthMonitor with healthMonitorId={0} was created.
Corrective Action	N/A

Logger	org.midonet.event.topology.HealthMonitor.UPDATE
Message	UPDATE healthMonitorId={0}, data={1}.
Level	INFO
Explanation	HealthMonitor with healthMonitorId={0} was updated to {1}.
Corrective Action	N/A

Logger	org.midonet.event.topology.HealthMonitor.DELETE
Message	DELETE healthMonitorId={0}.
Level	INFO
Explanation	HealthMonitor with healthMonitorId={0} was deleted.
Corrective Action	N/A

API server events

This section describes the messages associated with API server events.

NSDB (Network State Database)

Logger	org.midonet.event.api.Nsdb.CONNECT
Message	CONNECT Connected to the NSDB cluster.
Level	INFO
Explanation	API server was connected to the NSDB cluster.
Corrective Action	N/A

Logger	org.midonet.event.api.Nsdb.DISCONNECT
Message	DISCONNECT Disconnected from the NSDB cluster.
Level	WARNING
Explanation	API server was disconnected from the NSDB cluster.

Corrective Action	If the connection is restored after this event, no corrective action is required. If this event continues, check the network connection between the API server and the NSDB cluster.

Logger	org.midonet.event.api.Nsdb.CONN_EXPIRE
Message	CONN_EXPIRE Connection to the NSDB cluster expired.
Level	ERROR
Explanation	The connection from the API server to the NSDB cluster expired.
Corrective Action	Check the network connection between the API server and the NSDB cluster and restart the MidoNet API server so it reconnects to the NSDB cluster.

MidoNet Agent events

This section describes the messages associated with MidoNet Agent events.

NSDB

Logger	org.midonet.event.agent.Nsdb.CONNECT
Message	CONNECT Connected to the NSDB cluster.
Level	INFO
Explanation	MidoNet Agent was connected to the NSDB cluster.
Corrective Action	N/A

Logger	org.midonet.event.agent.Nsdb.DISCONNECT
Message	DISCONNECT Disconnected from the NSDB cluster.
Level	WARNING
Explanation	MidoNet Agent was disconnected from the NSDB cluster.
Corrective Action	If the connection is restored after this event, no corrective action is required. If this event continues, check the network connection between the MidoNet Agent and the NSDB cluster.

Logger	org.midonet.event.agent.Nsdb.CONN_EXPIRE
Message	CONN_EXPIRE Connection to the NSDB cluster expired. Shutting down the MidoNet Agent.
Level	ERROR
Explanation	The connection from the MidoNet Agent to the NSDB cluster expired. Shutting down the MidoNet Agent.
Corrective Action	Check the network connection between the MidoNet Agent node and the NSDB cluster and restart the MidoNet Agent service on the node so it reconnects to the NSDB cluster.

Interface

Logger	org.midonet.event.agent.Interface.DETECT
Message	NEW interface={0}
Level	INFO
Explanation	MidoNet Agent detected a new interface={0}.
Corrective Action	N/A

Logger	org.midonet.event.agent.Interface.UPDATE

Message	UPDATE interface={0} was updated.
Level	INFO
Explanation	MidoNet Agent detected an update in interface={0}.
Corrective Action	N/A

Logger	org.midonet.event.agent.Interface.DELETE
Message	DELETE interface={0} was deleted.
Level	INFO
Explanation	MidoNet Agent detected that interface={0} was deleted.
Corrective Action	N/A

Service

Logger	org.midonet.event.agent.Service.START
Message	START Service started.
Level	INFO
Explanation	Service started.
Corrective Action	N/A

Logger	org.midonet.event.agent.Service.EXIT
Message	EXIT Service exited.
Level	WARNING
Explanation	Service exited.
Corrective Action	Restart the MidoNet Agent service if this event happened unintentionally. If this event recurs, file a ticket in the bug tracker for further investigation by developers.

Packet Tracing

To configure packet tracing (via logging) in a MidoNet Agent (Midolman), the 'mm-trace' command can be used.

A MidoNet Agent can hold a set of filters that, when matching on an incoming packet, will cause it to log everything about its simulation to the agent’s log file, regardless of the configured log level.

All trace messages have a "cookie:" prefix to identify its packet, and that can be used as a grep expression to filter out any non-tracing messages.

Important

The filters are not persistent, they are lost every time the agent is rebooted.

However, mm-trace prints the filters in exactly the same syntax that it will accept to re-add them again, allowing operators to easily replay the commands.

Usage

All available options can be displayed with the '–help' option:

```
$ mm-trace --help
  -h, --host  <arg>   Host (default = localhost)
```

```
  -p, --port  <arg>   JMX port (default = 7200)
      --help          Show help message

Subcommand: add - add a packet tracing match
  -d, --debug                logs at debug level
      --dst-port  <arg>      match on TCP/UDP destination port
      --ethertype  <arg>     match on ethertype
      --ip-dst  <arg>        match on ip destination address
      --ip-protocol  <arg>   match on ip protocol field
      --ip-src  <arg>        match on ip source address
  -l, --limit  <arg>         number of packets to match before disabling
 this trace
      --mac-dst  <arg>       match on destination MAC address
      --mac-src  <arg>       match on source MAC address
      --src-port  <arg>      match on TCP/UDP source port
  -t, --trace                logs at trace level
      --help                 Show help message
Subcommand: remove - remove a packet tracing match
  -d, --debug                logs at debug level
      --dst-port  <arg>      match on TCP/UDP destination port
      --ethertype  <arg>     match on ethertype
      --ip-dst  <arg>        match on ip destination address
      --ip-protocol  <arg>   match on ip protocol field
      --ip-src  <arg>        match on ip source address
  -l, --limit  <arg>         number of packets to match before disabling
 this trace
      --mac-dst  <arg>       match on destination MAC address
      --mac-src  <arg>       match on source MAC address
      --src-port  <arg>      match on TCP/UDP source port
  -t, --trace                logs at trace level
      --help                 Show help message
Subcommand: flush - clear the list of tracing matches
  -D, --dead-only   flush expired tracers only
      --help        Show help message
Subcommand: list - list all active tracing matches
  -L, --live-only   list active tracers only
      --help        Show help message
```

Example

```
$ mm-trace list
$ mm-trace add --debug --ip-dst 192.0.2.1
$ mm-trace add --trace --ip-src 192.0.2.1 --dst-port 80
$ mm-trace list
tracer: --debug --ip-dst 192.0.2.1
tracer: --trace --ip-src 192.0.2.1 --dst-port 80
$ mm-trace remove --trace --ip-src 192.0.2.1 --dst-port 80
Removed 1 tracer(s)
$ mm-trace flush
Removed 1 tracer(s)
```

Port mirroring

Port mirroring lets operators monitor arbitrary subsets of traffic in the overlay in specified vports. This can be useful for passive monitoring or for active troubleshooting.

MidoNet v5.0 introduces port mirroring based on these concepts:

1. A new type of virtual device: **mirror**.
2. Each mirror is associated with a destination virtual port, through its **to-port** attribute. This is where mirror traffic will be copied to.

3. Each mirror has a list of **matches**. Matches are conditions that match traffic exactly like conditions in rule chains do, they have the same attributes. These **matches** select which traffic will be captured by the mirror.

4. Ports, bridges and routers contain two lists of mirrors: inbound and outbound. These are the points at which mirrors may capture traffic.

Operators can create mirrors, configure them to match the desired traffic and apply them at one or several points in the virtual topology.

Mirroring example

Let's assume a simple overlay topology:

1. A virtual bridge with three virtual ports
2. A virtual router with:
    a. One virtual port connected to an upstream physical router
    b. One virtual port connected to the bridge
3. Two VMs connected to the remaining two ports in the bridge and addresses 192.168.1.10 and 192.168.1.11.

If we inspect it with the CLI, it looks like this:

```
midonet> bridge list
bridge bridge0 name a-tenant state up
midonet> router list
router router0 name gateway state up asn 0
midonet> bridge bridge0 list port
port port0 device bridge0 state up plugged no vlan 0 peer router0:port0
port port1 device bridge0 state up plugged no vlan 0
port port2 device bridge0 state up plugged no vlan 0
midonet> router router0 list port
port port0 device router0 state up plugged no mac ac:ca:ba:73:9c:05
 addresses 192.168.1.1/24 peer bridge0:port0
port port1 device router0 state up plugged no mac ac:ca:ba:a0:6b:43
 addresses 10.0.0.1/24
midonet>
```

An operator wants to see/monitor some of the traffic in this overlay. Logging into the appropriate hypervisor where a VM may be running and executing tcpdump on the tap device where that VM is connected could work. However it's a cumbersome and error prone: one needs to find the particular hypervisor and tap. And it's not very flexible: one may want to monitor traffic that belongs to several VMs, or traffic as it looks like when it traverses a virtual router in the middle of the topology.

Preparing a monitoring namespace. To get started with port mirroring, we need a port to mirror to. For this purpose, lets create an isolated monitoring bridge, add a port to it and hook up a Linux network namespace to the port, where we can run tcpdump or any other passive network monitoring tool.

First, let's create the bridge and port:

```
midonet> bridge create name "Monitoring bridge"
bridge1
midonet> list bridge
bridge bridge0 name a-tenant state up
```

```
bridge bridge1 name Monitoring bridge state up
midonet> bridge bridge1 add port
bridge1:port0
midonet>
```

Now let's log into a hypervisor, which will be the monitoring machine, its hostname is **hypervisor01**. Let's create a network namespace in it and a veth pair we can bind to a MidoNet vport:

```
$ sudo ip netns add mon
$ sudo ip link add name mondp type veth peer name monns
$ sudo ip link set mondp up
$ sudo ip link set monns netns mon
$ sudo ip netns exec mon ip link set monns up
$ sudo ip netns exec mon ip link set dev lo up
```

Now we have a namespace named **mon**, which contains a network interface named **monns**. On the other side of the veth pair, **mondp** is ready to be bound to a midonet vport.

Let's go on with that step, in **midonet-cli**. First, let's identify the monitoring host, we can use its host name, **hypervisor01** to filter the list of hosts:

```
midonet> host list name hypervisor01
host host0 name hypervisor01 alive true addresses 10.1.0.1,127.0.0.1
midonet>
```

Now we can bind the "physical" monitoring namespace in the monitoring port in the virtual topology:

```
midonet> host host0 add binding port bridge1:port0 interface mondp
host host0 interface mondp port bridge1:port0
```

At this point, the monitoring network namespace is connected to MidoNet, albeit to a virtual bridge that has no other ports. However this is not a problem, because we are going to use it to mirror packets from other parts of the overlay.

Using port mirroring. Once the monitoring namespace is ready, mirrors can be used to see copies of any subset of overlay traffic in the monitoring port. The operator just needs to create the mirrors with the appropriate matches and apply them to any point or points in the overlay topology.

Let's see how an operator would monitor ip traffic within the tenant's bridge. In other words, the operator wants to see traffic that is local to the bridge and not traffic going towards or coming from the router.

To achieve that, create a mirror that matches traffic in the 192.168.1.0/24 network:

```
midonet> create mirror to-port bridge1:port0
mirror0
midonet> mirror mirror0 matches add dst 192.168.0.0/24 src 192.168.0.0/24
src 192.168.0.0/24 dst 192.168.0.0/24
midonet> mirror mirror0 list matches
src 192.168.0.0/24 dst 192.168.0.0/24 fragment-policy any no-vlan false
midonet>
```

...and apply it to one of the two mirroring hooks in the bridge:

```
midonet> bridge bridge0 set in-mirrors mirror0
midonet> bridge bridge0 show
bridge bridge0 name a-tenant state up in-mirrors mirror0
midonet>
```

Now the operator can see all local traffic in that bridge by tcpdump'ing on the monitoring port:

```
hypervisor01$ sudo ip netns exec mon tcpdump -nei monns
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on monns, link-type EN10MB (Ethernet), capture size 65535 bytes
```

By the same means, the operator could mirror any other slice of traffic and do so from any point in the virtual overlay. If a mirror is applied to the upstream facing port of the router, the mirror will see the MAC and IP addresses as that port sees them.

Each mirror can be applied at any number of devices, and can hold several match conditions to capture different slices of traffic. Similarly, each mirroring hook in a device, can have several mirrors applied. Thus the operator has total freedom in selecting which traffic to monitor in his monitoring port, or, by creating different network interfaces and adding more vports to the monitoring bridge, he could also send different kinds of traffic to different monitoring ports.

Removing port mirrors. To remove port mirrors from a bridge, use the `clear` command:

```
midonet> bridge bridge0 clear in-mirrors
midonet> bridge bridge0 show
bridge bridge0 name a-tenant state up
```

15. VXLAN configuration

Table of Contents

MidoNet supports the Virtual Extensible LAN (VXLAN) technology.

What is VXLAN?

VXLAN is a network virtualization technology that uses a VLAN-like encapsulation technique to encapsulate MAC-based OSI layer 2 Ethernet frames within layer 3 UDP packets.

This type of encapsulation (Ethernet-in-IP) is much better suited to Software Defined Networks than either VLANs (802.1q) or even stacked VLANs (Q-in-Q).

Another important advantage of VXLAN over traditional VLAN is its 24-bit VXLAN ID thanks to which VXLAN can scale up to over 16 million logical networks. By comparison - the maximum number of VLANs is 4096.

How is VXLAN supported in MidoNet?

MidoNet provides VXLAN implementation through:

- VXLAN Gateway, to bridge the overlay with physical L3 hosts in the underlay.
- VXLAN tunneling between MidoNet hosts.

VXLAN Gateway

The VXLAN Gateway (VXGW) allows a virtual bridge to be extended to a physical L2 segment that is reachable via an L3 network and a VXLAN-capable physical switch.

A VXLAN-capable physical switch is also referred to as a *hardware VTEP* (VXLAN Tunnel End Point). The VXGW allows creating one or many VXLAN-based Logical Switches that span any number of hardware VTEPs and a single MidoNet-ODP cloud.

The VXGW has the following advantages:

- Provides L2 connectivity between VMs in an overlay and servers in a physical L2 segment.

- Provides L2 connectivity across an L3 transport network. This is useful when the L2 fabric doesn't reach all the way from the racks hosting the VMs to the physical L2 segment of interest.
- Compared to a pure L2 gateway, the VXGW scales better for overlay solutions:
  - in a pure L2 solution, traffic between VMs and the physical segment must be routed through a few gateway nodes that are physically connected to the L2 segment. The physical connections are inherently limited; additionally, their use is limited by protocols like STP.
  - with the VXGW, traffic between VMs and the physical segment can be routed directly between any of the compute hosts and the hardware VTEP.

VXGW Management

A VXGW will be formed by binding a Neutron network to any number of port-vlan pairs on one or multiple VTEPs.

VTEPs implement an abstraction called Logical Switch, independent from MidoNet. A Logical Switch represents a virtual L2 segment that connects VLANs on some of the VTEP's ports. For example, in a given VTEP "A", with ports p1, p2, a Logical Switch can be formed by binding (p1, VLAN 40) and (p2, VLAN 30). A Logical Switch can also extend the L2 segment to ports on a different VTEP, whereby both devices would tunnelling traffic on the Logical Switch.

A port-VLAN pair can only be bound to a single Logical Switch. However, a given port may be configured multiple Logical Switches as long as the binding combines different VLANs.

The MidoNet Coordinator (the section called “VXLAN Coordinator” [89]) is able to connect to the Management Service of VTEPs, create and configure Logical Switches in its OVSDB instance according to the settings applied through the MidoNet API. Additionally, the Coordinator is able to extend the Logical Switch functions described above to a Neutron network.

MidoNet simplifies and automates these configuration details which remain invisible to the user. It uses some conventions that may be useful for two reasons: troubleshooting purposes, and making MidoNet usage of the VTEP compatible with non-MidoNet related ones.

- MidoNet will create a single Logical Switch in the VTEP to group all port-VLAN pairs bound to each Neutron network.
- The Logical Switch name will be formed by prepending `mn-` to the Neutron network ID, being thus unique in a single MidoNet deployment. Operators are free to create Logical Switches with any name format, but should never create VTEPs with names prefixed with `mn-`.
- The Logical Switch tunnel key (VNI) is automatically generated by MidoNet, monotonically increasing from 10000. Operators are free to use VNIs from 1 to 9999 for their own purposes.
- When a Neutron network is bound to port-VLAN pairs on multiple VTEPs, a Logical Switch will be created on each VTEP's database. However, all will share the same name and VNID, following the conventions above.

The MidoNet controller will handle the exchange of learned MACs among all VTEPs and MidoNet's Network State Database (NSDB) automatically.

VXLAN Coordinator

The Coordinator is the component of the MidoNet architecture responsible for VXLAN support.

The Coordinator has the following responsibilities:

- Exposing VTEP state through the MidoNet REST API.
- Configuring the VTEP switch in order to implement the bindings configured through the MidoNet REST API.
- Acting as an L2 control plane for traffic flowing between MN and the VTEP.

VXLAN Flooding Proxy

The VXLAN Gateway controller running in the MidoNet Cluster nodes will try to populate the MAC Remote tables in VTEPs so that the switch can tunnel traffic directly to the exact hypervisor that hosts the destination VM.

Depending on the virtual topology, it may not always be possible to instruct the VTEP to tunnel to a specific physical location. This will typically happen on BUM (Broadcast, Unknown and Multicast) traffic. In these cases MidoNet will instruct the VTEP to tunnel the packet to a service node in order for it to be simulated and delivered to the right destination. This node is called the "Flooding Proxy", and it has the same properties:

- The Flooding Proxy (FP) is one single node elected among all the MidoNet hosts that belong to the same tunnel zone as the VTEP.
- The FP will be in charge of simulating BUM traffic, and tunnelling the packet to their destination (typically a hypervisor).
- Upon failure of the currently elected Flooding Proxy, the MidoNet cluster will use a weighted algorithm to elect a new "Flooding Proxy" role, and instruct the VTEP to tunnel all BUM traffic to it for simulation.

The weight assigned to a MidoNet Agent defaults to 1, and can be altered issuing the following command on the MidoNet CLI:

```
host <host-alias> set flooding-proxy-weight <new-weight>
```

Higher weights will imply a higher probability of the host being chosen as Flooding Proxy.

To exclude an Agent from the candidate set for Flooding Proxy, assign a weight of 0.

Note that the Flooding Proxy may potentially process a large volume of traffic. In these circumstances it is recommended to assign a much higher weight to a dedicated host.

Connecting to the VTEP

Use this procedure to connect MidoNet to a hardware VTEP. This step is required before any Neutron networks can be bound to port-VLAN pairs on that VTEP.

1. Refer to the documentation of your switch to enable VXLAN on it, then configure it as a VTEP with all the required parameters.

   MidoNet will expect that the Physical_Switch table on the VTEP contains a record with the management IP, management port and tunnel IP of this VTEP. Keep these details at hand as they will be needed to configure the VTEP and any bindings to Neutron networks. Use the following command to dump the contents of this table:

   ```
   vtep-ctl list Physical_switch
   ```

   Make sure that your VTEP also registers all the physical ports. You can verify this by examining the Physical_Ports table in the VTEP. Only ports present in this table will be available for binding to a Neutron network, Use the following command would display all physical ports, replacing *<vtep-name>* with the name assigned to the Physical_Switch (you can check this using the previous `vtep-ctl list Physical_Switch` command).

   ```
   vtep-ctl list-ports <vtep-name>
   ```

2. After setting up the VTEP you might need to test connectivity to both the tunnel and management interfaces. Both should be *up*.

   To test the connection to the management database, you can run:

   ```
   $ telnet <management-ip> <management-port>
   Trying <management-ip>...
   Connected to <management-ip>
   Escape character is '^]'.
   ```

   At this point, paste this on the console:

   ```
   {"method":"list_dbs","id":"list_dbs","params":[]}
   ```

   And you should see this reply:

   ```
   {"id":"list_dbs","error":null,"result":["hardware_vtep"]}
   ```

   This just verified that the OVSDB server that holds the configuration of this VTEP is active and handling connections. If you did not get a similar reply, please review the configuration of the VTEP.

3. The VXLAN service in Midonet is enabled by default. However, if the service has been previously disabled, you can enable it by typing the following command and restarting at least one cluster instance in order to become the VXLAN coordinator:

   ```
   $ echo "cluster.vxgw.enabled=true" | mn-conf set -t default
   ```

4. Having ensured that the VTEP has been properly configured, now you're ready to add the VTEP to the MidoNet configuration.

   For information, see the section called "Adding a VTEP" [100].

   ### Important

   Apart from the information on the VLAN-port assignment, and VTEP management interface IP and port, you will also need the identifier of a TunnelZone of type *vtep*. All the hosts running MidoNet Agent daemons that you want to create VXLAN tunnels with the VTEP should be members of this tunnel zone, using the local IP that each host uses as a VXLAN tunnel endpoint.

After you successfully add a VTEP to the MidoNet configuration, the API Server connects to its (VTEP's) management interface and collects all the required information for creating a Logical Bridge. For more information, see the "Logical Bridge" section.

Setting up a connection between a VTEP and a Neutron network

Use this procedure to set up a connection between a VTEP and a Neutron network in MidoNet.

For this procedure you will need to know the VTEP's management IP and port, the physical port on the VTEP and the VLAN ID at this port to which you are connecting, the UUID of the Neutron network which you want to connect to the VTEP, and the IP addresses of all the hosts on the Neutron network that you want to communicate with the VTEP.

1. Create a tunnel zone of type *vtep*.

   All hosts that want to communicate with the VTEP using VXLAN tunnels are required to belong to a tunnel zone of type VTEP, using the IP that each of them will use as VXLAN tunnel endpoint.

   To create a tunnel zone issue the following command in the MidoNet CLI:

   ```
   midonet> tunnel-zone create name vtep_zone type vtep
   tzone0
   ```

   As you can see you just created a *vtep* type tunnel zone, tzone1.

2. Add a VTEP to MidoNet and assign it to the *vtep* tunnel zone that you created, using the local IP that this host it meant to use in VXLAN tunnels to the VTEP. Note that this IP may be the same that the host uses to communicate with other MidoNet hosts.

   ```
   midonet> vtep add management-ip 192.168.2.10 management-port 6632
    tunnel-zone tzone0
   vtep0
   ```

   You can determine the state of the VTEP connection by listing the current VTEPs. If your VTEP had been added successfully you should see a similar message, saying `connection-state connected`.

   ```
   vtep vtep0 name VTEP-NAME description OVS VTEP Emulator management-ip
    192.168.2.10 management-port 6632 tunnel-zone tzone0 connection-state
    connected
   ```

3. Create a binding between the VTEP and a Neutron network behind a MidoNet bridge. For that, you will need the UUID of the Neutron network behind that bridge. To find out the UUID use these commands:

   ```
   midonet> list bridge
   bridge bridge0 name public state up
   midonet> show bridge bridge0 id
   765cf657-3cf4-4d79-9621-7d71af38a298
   ```

   The Neutron network you are binding the VTEP to is behind bridge0, and it has the UUID of 765cf657-3cf4-4d79-9621-7d71af38a298 as you can see in the output of the command.

4. For the hosts on the Neutron network to be able to communicate with the VTEP, their IP addresses have to be in the same tunnel zone as the VTEP.

   a. To find out their addresses, use these commands:

   ```
   midonet> host list
   host host0 name rhos5-allinone-jenkins.novalocal alive true
   midonet> host host0 list interface
   iface veth1 host_id host0 status 3 addresses [u'172.16.0.2',
    u'fe80:0:0:0:fc2a:9eff:fef2:aa6c'] mac fe:2a:9e:f2:aa:6c mtu 1500
    type Virtual endpoint DATAPATH
   ...
   ```

   b. Add the host's IP address to the same tunnel zone as the VTEP:

   ```
   midonet> tunnel-zone tzone0 add member host host0 address 172.16.0.2
   ```

   Repeat these steps for every host in the Neutron network that you want to communicate with the VTEP.

   Important

   Normally, a host may only be assigned to one tunnel zone, *gre* or *vxlan* type. A host connecting to a VTEP is an exception because you may assign it to two tunnel zones, a *gre*/*vxlan* one and a *vtep* one. Note that a host can still use the same IP for both tunnel zones.

5. Create a binding between the VTEP's VLAN 10 and the Neutron network behind the bridge0.

   In this example you are connecting the hosts on VLAN 10 with the Neutron network 765cf657-3cf4-4d79-9621-7d71af38a298 behind the bridge0:

   ```
   midonet> vtep vtep0 add binding network-id
    765cf657-3cf4-4d79-9621-7d71af38a298 physical-port swp1 vlan 10
   ```

Congratulations, you have just created a binding between the network behind the VTEP's VLAN `10` physical port `swp1` and the Neutron network with the UUID `765cf657-3cf4-4d79-9621-7d71af38a298` in MidoNet.

Tip

To be able to test the connection between the VTEP and MidoNet (i.e. to *ping* MidoNet from a host on the VTEP) you have to modify the default ingress security rule, by adding to it the IP address of the host (pinging the host from MidoNet should work without any additional configuration). For more information, see the section called "Enabling connection between VTEP and MidoNet hosts" [92].

Enabling connection between VTEP and MidoNet hosts

By default Neutron includes a security rule on all networks that restricts forwarding only to traffic addressed to IP/MAC of VMs on that network.

By binding a network to physical ports in a VTEP, we're effectively adding hosts to the L2 segment of this Neutron network that Neutron itself does not know about, and thus traffic addressed to these physical hosts will be dropped.

The following procedure describes changes ton the default ingress security rule in order to allow traffic to hosts on the VTEP.

1. In the MidoNet CLI find out what the ingress default security rule is by issuing this command:

   ```
   midonet> list chain
   chain chain0 name OS_SG_64d9f3df-9875-4896-ad0c-ffc5bba84c5e_INGRESS
   chain chain1 name OS_SG_64d9f3df-9875-4896-ad0c-ffc5bba84c5e_EGRESS
   ...
   ```

   Locate the ingress security rule that is assigned to the neutron network. In this case, we'll use chain0 (OS_SG_64d9f3df-9875-4896-ad0c-ffc5bba84c5e_INGRESS) rule chain, the ingress chain.

2. List the rules that implement this security rule by issuing this command:

   ```
   midonet> chain chain0 list rule
   rule rule0 ethertype 2048 proto 0 tos 0 ip-address-group-src ip-address-
   group0 fragment-policy unfragmented pos 1 type accept
   rule rule1 ethertype -31011 proto 0 tos 0 ip-address-group-src ip-
   address-group0 fragment-policy unfragmented pos 2 type accept
   ```

   The security group that is responsible for controlling ICMP packets (ethertype 2048=IP) is ip-address-group0.

3. Now, go ahead and add the IP address of the host on the VTEP to the security group ip-address-group0.

   For example, if the IP address of the host is 172.16.0.3, issue this command:

   ```
   midonet> ip-address-group ip-address-group0 add ip address 172.16.0.3
   address 172.16.0.3
   ```

You should now be able to ping a host in MidoNet from host 172.16.0.3 on the VTEP (providing they are in the same tunnel zone).

VXLAN Gateway high availability (VTEP side, active-passive mode)

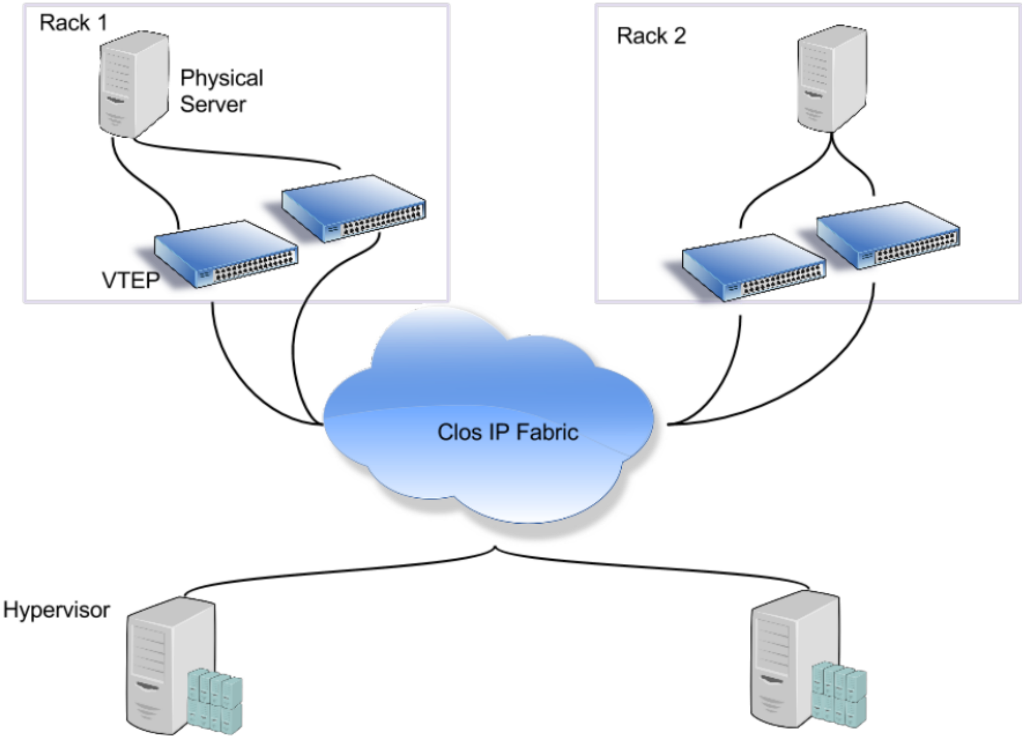
MidoNet supports VTEP HA by using two ToR VTEPs, both connected to each host in the rack with an active-passive configuration.

There are two possible failure scenarios:

- The active link from a host is lost. The passive link will take over, so traffic from the host will now flow through the other VTEP and its MACs will be learned there.
- A VTEP dies. Any host whose active link connects to this VTEP will now move to the passive link, with the same result: MACs and IPs migrate to a different VTEP.

MAC migrations will be applied from the VxLAN Gateway Service as soon as the OVSDB notifies about them, the new MacLocations will be propagated to VTEPs and NSDB, and traffic from agents and other VTEPs will now flow through the new VTEP.

Figure 15.1. VTEP HA



Troubleshooting VTEP/VXGW configuration

VTEP deployments have a relatively large number of moving pieces and potential failure points. This guide will focus on troubleshooting MidoNet and the integration with the VTEP. For specifics on the configuration of the logical switch please refer to your vendor's documentation.

Is the MidoNet API able to connect to the VTEP

After following the procedure to add a VTEP as described in the section called "Adding a VTEP" [100], the expected output should be as follows:

```
midonet> vtep add management-ip 192.168.2.10 management-port 6633 tunnel-
zone tzone0
vtep0
midonet> vtep list
vtep vtep0 name VTEP-NAME management-ip 192.168.2.10 management-port 6633
 tunnel-zone tzone0 connection-state connected
```

The same output should appear for VTEPs already added to MidoNet.

Note that the state is `connected`. An `error` state will indicate that the VTEP's management IP is unreachable from the MidoNet API.

Is the VTEP well configured?

A typical reason for the VTEP being on `error` state is a misconfiguration of the VTEP OVDSB instance. You can verify this by executing the following command on the console:

```
ovsdb-client dump hardware_vtep
```

Scroll down to the Physical_Switch table, which will look like this:

```
Physical_Switch table
_uuid                                description        management_ips
  name        ports                                   switch_fault_status
 tunnel_ips
------------------------------------ ------------------ ----------------
 ----------- -------------------------------------- -------------------
 ------------
3647f020-9ecf-4854-8f75-9011b8c9996a "VTEP-DESCRIPTION"  ["192.168.2.10"]
 "VTEP-NAME" [698ede89-31f8-4797-a885-1b2dd4c585e3] []
 ["10.0.0.1"]
```

Verify that an entry exists, and the `management_ips` and `tunnel_ips` fields correspond to the physical configuration. The management IP is the one you'll be using on the `vtep add` command. The tunnel IP is not relevant at this point, however, MidoNet expects that a value is present in this field.

Is the OVSDB instance running and accessible?

If the MidoNet API shows an ERROR on the VTEP list, and your configuration is correct, you should verify that the OVSDB instance is listening on the same management-port that you're specifying in the vtep add call.

From the host running the MidoNet REST API (and Coordinator) try to establish a Telnet connection to the VTEP management interface IP and port, assuming these are 192.168.2.10 and 6632:

```
telnet 192.168.2.10 6632
```

If the connection is successful, you should see the following output:

```
Trying 192.168.2.10..
Connected to localhost.
Escape character is '^]'.
```

This means that we have a TCP socket listening on the right port. We can now verify that the OVSDB is responsive. If this is not the case, then check your switch manual to listen for connections at the selected TCP port.

If the output was correct, then enter the following input into the console:

```
{"method":"list_dbs","id":"list_dbs","params":[]}
```

The desired output is:

```
{"id":"list_dbs","result":["hardware_vtep"],"error":null}
```

The content of the brackets after `result` may vary, but we must see a `hardware_vtep`, indicating that there is a VTEP schema on this instance of the OVSDB. If you fail to see this output, the VTEP likely doesn't contain a `hardware_vtep` schema in its OVSDB instance. Refer to your switch documentation for instructions to configure it.

VTEP and bindings are added but no traffic goes through

Verify first that you enabled the VXLAN Gateway service in the MidoNet API. This service is enabled by default and it is required to configure the VTEP and synchronize its state. If the service has been previously disabled, you can enable it using the `mn-conf` utility tool to set the `cluster.vxgw.enabled` to `true`:

```
$ echo "cluster.vxgw.enabled=true" | mn-conf set -t default
```

After enabling the service you must restart one or more MidoNet cluster instances.

Verify that the VXLAN Gateway service is started

The VXLAN Gateway service may be enabled in several MidoNet Cluster instances. All of them will coordinate through the Network State Database (NSDB) to elect a leader that will perform all coordination tasks for a particular VTEP. When a MidoNet Cluster instance takes over leadership the following DEBUG message is displayed in the cluster log (`/var/log/midonet-cluster/midonet-cluster.log`):

```
DEBUG snatcher-99a629d1-c729-4973-b7e0-44183a5589ff I own Vtep
 ba2739df-87cf-458f-9ad2-39885cab217d
```

If another instance is already a leader, all other instances will display the following DEBUG message:

```
DEBUG snatcher-99a629d1-c729-4973-b7e0-44183a5589ff Vtep
 ba2739df-87cf-458f-9ad2-39885cab217d already taken by node
 45926c1a-1264-4e19-87c9-b6aca33169f2
```

At least one instance of the MidoNet API should display the positive message indicating that it became the VXLAN Gateway leader for a particular VTEP. This is the instance that should be watched for further log messages.

When deploying more than one MidoNet Cluster instance, different instances may be responsible of different hardware VTEPs at the same time. In each case, the log indicates the VTEPs of which a particular instance has assumed ownership.

If a MidoNet cluster instance is shut-down or fails, the remaining MidoNet Cluster instances will assume the ownership and coordinate any orphaned VTEPs.

Verify that the VXLAN Gateway leader picks up VTEPs and Networks

The VXLAN Gateway service will scan all the Neutron networks in MidoNet's NSDB and proceed to monitor those that are bound to any VTEPs.

Whenever a Neutron network is bound to a VTEP, the following message will appear in the DEBUG logs. Note that all log messages relevant for a given Neutron network will be tagged with the appropriate UUID:

```
DEBUG vxgw-vtep-ba2739df-87cf-458f-9ad2-39885cab217d Network 4659a6ab-
fcd2-4744-bfbb-6a331164881e updated its bindings
```

You can filter updates relevant to specific networks by editing:

```
vi /etc/midonet-cluster/logback.xml
```

Follow the instructions detailed in this file to enable different processes in the coordinator. For brevity, log messages mentioned below will omit the Network UUID tag.

As mentioned above, you should be seeing a message like the following for each Neutron network:

```
DEBUG vxgw-vtep-ba2739df-87cf-458f-9ad2-39885cab217d Network 4659a6ab-
fcd2-4744-bfbb-6a331164881e is now watched
```

Failures during this phase typically indicate errors accessing the NSDB, for example:

```
ERROR vxgw-vtep-ba2739df-87cf-458f-9ad2-39885cab217d VTEP error
```

The MidoNet controller will WARN in the logs whenever a recoverable error is found, and try to restore connectivity to the NSDB. Non recoverable errors will be marked as ERROR.

If the logs show problems connecting to the NSDB verify that the NSDB is active, and MidoNet Cluster is successfully able to access it.

Verify that the MidoNet coordinator synchronizes MAC with the VTEPs

After fetching Neutron network configuration from the NDSB, the MidoNet Cluster logs should display the following message:

```
MAC-port table of network 4659a6ab-fcd2-4744-bfbb-6a331164881e is now
 watched
ARP table of network 4659a6ab-fcd2-4744-bfbb-6a331164881e is now watched
Network 4659a6ab-fcd2-4744-bfbb-6a331164881e is now watched
```

These indicate that the MidoNet coordinator is monitoring the network's state, which will be synchronized to the VTEP.

Verify that the MidoNet coordinator connects to the VTEP(s)

The MidoNet coordinator will also bootstrap a process to exchange state among the network, and all VTEPs with port-VLAN pairs bound to it. When the controller detects any port-VLAN pair in a new VTEP, it will show the following message (assuming management IP and management port are 192.168.2.10 and 6632):

```
New bindings from network 4659a6ab-fcd2-4744-bfbb-6a331164881e:
 ArrayBuffer((swp1,0))
```

At this point it will ensure that a connection is stablished to this VTEP's management IP and that the bindings configured through the MidoNet REST API are correctly reflected in the VTEP. Normal output will look like this (note that they may appear mixed with other messages):

```
Publishing remote MAC to VTEP: MacLocation{logicalSwitch='mn-4659a6ab-
fcd2-4744-bfbb-6a331164881e', mac=unknown-dst, ipAddr='null',
 tunnelEndpoint=null}
```

If the coordinator reports any errors connecting to the VTEP it will automatically try to connect, but you should verify that the VTEP is up and accessible.

Following a successful consolidation of state, MidoNet will start the synchronization of MACs and ARP entries:

```
VTEP table entry added: PhysicalLocatorSet{uuid=115da3cb-8926-42bd-
a416-7e006a353b73, locatorIds='Set(be104a88-5b9c-4b61-bf94-6ea3c797f612)'}
VTEP table entry added: PhysicalLocator{uuid=be104a88-5b9c-4b61-
bf94-6ea3c797f612, dstIp=172.17.2.10, encapsulation='vxlan_over_ipv4'}
VTEP table entry added: McastMac{uuid=57e5f212-72b6-4cd7-ba24-2eb3791a90c5,
 logicalSwitch=a8817c74-708f-438d-9a17-02377cbd01ce, mac=unknown-dst, ip=
null, locatorSet=115da3cb-8926-42bd-a416-7e006a353b73}
VTEP table entry added: PhysicalPort{uuid=5b90109a-fafc-4d53-
a2ae-f79736a5ae9b, name=swp1, description=null bindings=Map(0 ->
 a8817c74-708f-438d-9a17-02377cbd01ce) stats=Map(0 -> ac00d040-deca-45b1-
a391-468d45503238) faultStatus=Set()
```

Connection errors to the VTEP are possible at this point, but should be handled gracefully by the coordinator.

If MidoNet finds a non recoverable error, the following ERROR will be displayed (assuming same management port and id as above):

```
Error on OVSDB connection status stream
```

The MidoNet coordinator will ignore this Neutron network until it's updated again. It should however be able to continue operating with all other configured networks.

Verify that the MidoNet coordinator synchronizes state

If no errors are displayed up to here, edit the logback.xml file mentioned above and enable DEBUG logs in the vxgw processes:

```
<!-- <logger name="org.midonet.cluster.vxgw" level="DEBUG" /> -->
```

Remove the `<!--` and `-#` tags to enable this configuration and restart the MidoNet Cluster logs to begin showing DEBUG messages. Choose TRACE instead of DEBUG for more exhaustive information (none will be too verbose to have a significant performance impact).

Messages like the following show that the MidoNet coordinator is successfully exchanging MACs among Midonet and VTEPs.

```
TRACE vtep-172_17_2_10:6632 MAC remote tables updated successfully
```

Verify that VXLAN tunnels are being established

If the coordinator is working normally, but traffic is still not flowing, you should verify that the VTEPs and MidoNet hosts are able to stablish VXLAN tunnels successfully.

While keeping a ping active from the VM to the server you're trying to communicate with, log in to the MidoNet compute hosting the VM that you're trying to communicate with a server on the VTEP. Run the following command:

```
tcpdump -leni any port 4789
```

Assuming that the MidoNet compute is 192.168.2.14, and the VTEP's tunnel IP is 192.168.2.17, the output should be similar to this (depending on your tcpdump version):

```
15:51:28.183233 Out fa:16:3e:df:b7:53 ethertype IPv4 (0x0800), length 94:
 192.168.2.14.39547 > 192.168.2.17.4789: VXLAN, flags [I] (0x08), vni 10012
aa:aa:aa:aa:aa:aa > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42:
 Request who-has 10.0.0.1 tell 10.0.0.10, length 28
15:51:28.186891  In fa:16:3e:52:d8:f3 ethertype IPv4 (0x0800), length 94:
 192.168.2.17.59630 > 192.168.2.13.4789: VXLAN, flags [I] (0x08), vni 10012
cc:dd:ee:ee:ee:ff > aa:aa:aa:aa:aa:aa, ethertype ARP (0x0806), length 42:
 Reply 10.0.0.10 is-at cc:dd:ee:ee:ee:ff
```

The first line shows that the MidoNet Agent (192.168.2.14) is emitting a tunnelled packet towards the VTEP (192.168.2.17:4789), using 10012 as VNID. The encapsulated packet is shown on the second line, and corresponds to an ARP REQUEST from a VM with ip 10.0.0.10 regarding server 10.0.0.1.

In this example, the VTEP is responding correctly on the third line, showing a return packet with the same VNID.

The same example can be applied in reverse on the VTEP. A ping from the physical server connected to the VTEP should generate a tunnelled packet towards a MidoNet Agent, and receive similar return packets.

The MidoNet agent is not emitting traffic

Verify the VXLAN-related options in mn-conf(1). Examine the MidoNet Agent logs in debug mode and look for simulations on the Neutron network that might be dropping packets, or throwing errors on the simulation.

The VTEP is not emitting traffic on the tunnel

Ensure that the VTEP configuration reflects the bindings configured throught the MidoNet REST API. Use the following command to list the VTEPs present in the switch:

```
vtep-ctl list-ls
```

This will display all Logical Switches present in the switch. If you bound a Neutron network with UUID c68fa502-62e5-4b33-9f2f-d5d0257deb4f, then you should see the following item in the list:

```
mn-c68fa502-62e5-4b33-9f2f-d5d0257deb4f
```

Now list the bindings on the port that you used to create the port-vlan binding in the midonet-cli. Let's assume we have `swp1`, and created a binding with `swp1` and VLAN 93. The expected output would be:

```
vtep-ctl list-bindings <vtep-name> swp1
0093 mn-c68fa502-62e5-4b33-9f2f-d5d0257deb4f
```

You can find out the <*vtep-name*> using the `vtep-ctl list-ps` command.

If any of these outputs is not as expected, the MidoNet coordinator is most likely not being able to consolidate the configuration from the NSDB. Verify the MidoNet API logs and locate the relevant errors in order to correct them.

Verify that MACs are being synchronized correctly to the VTEP

Finally, you can list the local and remote MACs present in the VTEP's database:

```
vtep-ctl list-local-macs mn-c68fa502-62e5-4b33-9f2f-d5d0257deb4f
```

This should show all the MACs learned by the VTEP from traffic observed on local ports. If the local server is correctly configured, you will typically see the server's MAC here.

The following command will display the remote MACs:

```
vtep-ctl list-remote-macs mn-c68fa502-62e5-4b33-9f2f-d5d0257deb4f
```

The list here will show MACs in MidoNet VMs or other VTEPs, which are injected by the MidoNet coordinator.

If any of these steps don't show an expected output, the synchronization processes may be failing. Inspect the MidoNet API logs for more details.

CLI commands used for working with the VXGW

This section describes the CLI commands that you can use to work with the VXGW and VTEPs.

Obtaining a list of VTEPs

Use this command to obtain the list of hardware VTEPs of which MidoNet is aware.

Syntax

```
list vtep
```

Result

For each VTEP, the command returns this information:

- name
- description
- management IP address
- management port
- tunnel IP addresses
- connection state, which indicates the state of the connection between the MidoNet cluster node managing the VTEP and the hardware VTEP, and it can be one of:
  - `connected` when the cluster node established a connection
  - `disconnected` when all MidoNet cluster nodes are disconnected from the VTEP, such as when no cluster node has started managing the VTEP or when the VXLAN Gateway service is disabled across all cluster nodes.
  - `error` when there is a MidoNet cluster node owning and managing the VTEP but establishing a connection to the VTEPs OVSDB database failed.

Example

```
midonet> list vtep
vtep vtep0 name VTEP-NAME management-ip 192.168.2.10 management-port 6633
 connection-state connected
```

Adding a VTEP

Use this command to add a hardware VTEP to MidoNet.

Syntax

```
vtep add management-ip <vtep-ip-address> management-port <vtep-port>
 tunnel-zone <tunnel-zone-alias>
```

where *vtep-ip-address* and *vtep-port* are the VTEP's management IP address and port, and *tunnel-zone-alias* is used to determine the interface that will be used as the other end point of the VXLAN tunnel (in the MidoNet Agent).

Result

If the command runs successfully it writes the information you provided with it to Zookeeper. The command returns an error message, if a VTEP with these parameters already exists.

Examples

An example of a successful command:

```
midonet> vtep add management-ip 192.168.2.10 management-port 6633 tunnel-
zone tzone0
vtep vtep0 name VTEP-NAME management-ip 192.168.2.10 management-port 6633
 tunnel-zone tzone0 connection-state connected
```

An example of an unsuccessful command:

```
midonet> vtep add management-ip 192.168.2.10 management-port 6633
Internal error: {"message":"Validation error(s) found","code":400,
"violations":[{"message":"Tunnel zone ID is not valid.",
"property":"tunnelZoneId"}]}
```

Obtaining information about a VTEP

Use this command to obtain information about a selected VTEP.

Syntax

```
vtep <vtep-alias> show <property>
```

where *property* is one of the following VTEP's attributes:

- name
- description
- management-ip
- management-port
- connection-state
- tunnel-zone

Result

The command returns the following information about the VTEP:

- name
- description
- management IP address (the same as the IP used with the command)
- mgmt_port (the same as the port values used by the command)
- tunnel IP addresses
- connection state (one of: connected, disconnected, error. The state is error if the end-point is not a VXLAN End Point)
- the tunnel-zone to which this VTEP belongs.

Example

Successful command:

```
midonet> vtep vtep0 show id
ba2739df-87cf-458f-9ad2-39885cab217d
```

```
midonet> vtep vtep0 show management-ip
192.168.2.10
```

Adding a VTEP binding

Use this command to bridge the port-VLAN pair on a VTEP to a specified Neutron network.

Syntax

```
vtep <vtep-alias> add binding physical-port <port-name> vlan <vlan-id>
 network-id <neutron-network-id>
```

where *neutron-network-id* is the UUID of the network to which the port-VLAN assignment is to be made.

Result

If the command is successful, the result is a description of the Neutron network, VTEP physical port and VLAN ID tuple:

```
physical-port swp1 vlan 0 network-id 4659a6ab-fcd2-4744-bfbb-6a331164881e
```

When creating the first binding to a network, MidoNet will create a new port on the corresponding virtual bridge called a 'VXLAN port'. This port represents the Neutron network’s wiring to the VTEP. The same VXLAN port is used for subsequent port-VLAN pairs bound to the same Neutron network.

The command will fail in these conditions:

- the hardware VTEP does not have a physical port with the specified name (code *401*)
- the hardware VTEP is not configured (code *503*).
- the Neutron network does not exist (code *400*).

Example

Successful command:

```
midonet> vtep vtep0 add binding physical-port swp1 vlan 0 network-id
 4659a6ab-fcd2-4744-bfbb-6a331164881e
physical-port swp1 vlan 0 network-id 4659a6ab-fcd2-4744-bfbb-6a331164881e
```

Unsuccessful command:

```
midonet> vtep vtep0 add binding physical-port swp10 vlan 0 network-id
 4659a6ab-fcd2-4744-bfbb-6a331164881e
Internal error: {"message":"The VTEP at 192.168.2.10:6632 does not have a
 port named swp10.","code":401}
```

```
midonet> vtep vtep0 add binding physical-port swp10 vlan 0 network-id
 4659a6ab-fcd2-4744-bfbb-6a331164881e
Internal error: {"message":"Cannot add binding to VTEP 192.168.2.10:6632
 because the physical switch is not configured.","code":503}
```

```
midonet> vtep vtep0 add binding physical-port swp10 vlan 0 network-id
 f00f00f0-0f00-f00f-00f0-0f00f00f00f0
Internal error: {"message":"There is no Network with ID f00f00f0-0f00-
f00f-00f0-0f00f00f00f0.","code":400}
```

Obtaining descriptions of VTEP bindings

The MidoNet CLI offers a command to obtain the descriptions of all bindings on a given VTEP, as well as all the VTEPs to which a specific Neutron network is bound to.

All bindings in a VTEP

First, list all the VTEPs in order to identify the desired management IP:

```
midonet> vtep list
vtep vtep0 name VTEP-NAME description VTEP-DESCRIPTION management-ip 192.
168.2.10 management-port 6632 tunnel-zone tzone0 connection-state connected
```

Result

The following command returns the descriptions of all VXLAN ports and their Neutron network bindings on the selected VTEP:

```
vtep vtep0 list binding
physical-port swp1 vlan 0 network-id 4659a6ab-fcd2-4744-bfbb-6a331164881e
physical-port swp2 vlan 10 network-id 1d475afc-d892-4dc7-af72-9bd88e565dde
physical-port swp3 vlan 20 network-id 4659a6ab-fcd2-4744-bfbb-6a331164881e
```

From the output you can see a list of all the port-VLAN pairs applied on a given VTEP. The following information is displayed (the first line will be used as example):

- The physical port name (e.g. `swp2`) and VLAN (e.g. `10`).
- The UUID of the Neutron network to which the port-VLAN pair is bound (e.g., `1d475afc-d892-4dc7-af72-9bd88e565dde`)

VTEPs bound in a Neutron network

First chose the MidoNet bridge corresponding to the desired Neutron network:

```
midonet> bridge list
bridge bridge0 name my_network state up
```

We can verify the id of this bridge, which will be the same as the Neutron network:

```
midonet> bridge bridge0 show id
4659a6ab-fcd2-4744-bfbb-6a331164881e
```

List the ports on the bridge:

```
midonet> bridge bridge0 port list
port port0 device bridge0 state up
port port1 device bridge0 state up
port port2 device bridge0 state up vtep vtep0
```

Result

The bridge will complete the list of ports with one entry for each of the VTEPs that contain at least one binding. In this case we see that the Neutron network is bound to port-VLAN pairs at VTEPs 192.168.2.10 (as shown in the `list binding` example above).

Removing a VTEP binding

Use this command to disassociate a port-VLAN pair from the Neutron network's LogicalSwitch.

Syntax

```
vtep <vtep-alias> delete binding physical-port <port-name>
```

Result

You can delete a single VTEP binding to a Neutron network. If that was the VTEP's last remaining port-VLAN pair bound to the network, then the Neutron network's VXLAN port is also deleted.

Example

Examples of successfully run commands:

```
midonet> vtep vtep0 delete binding physical-port swp1
```

An example of a unsuccessful command:

```
midonet> vtep vtep0 delete binding
Syntax error at: ...binding
```

Deleting a VTEP

Use this command to delete a VTEP.

Syntax

```
vtep <vtep-alias> delete
```

Result

Issuing this command completely deletes a VTEP from MidoNet's list of known VTEPs.

The command will fail if any of the VTEP's port-VLAN pairs are bound to any Neutron networks. For more information, see the section called "Removing a VTEP binding" [103]

Example

```
midonet> vtep vtep0 delete
```

16. Setting up an L2 gateway

Table of Contents

This section describes how to set up an L2 gateway between MidoNet's virtual bridges and physical switches.

Important

This feature is not currently implemented in OpenStack Neutron. Therefore, you cannot use an L2 gateway to extend a MidoNet network (or bridge) to a Neutron bridge.

Topology

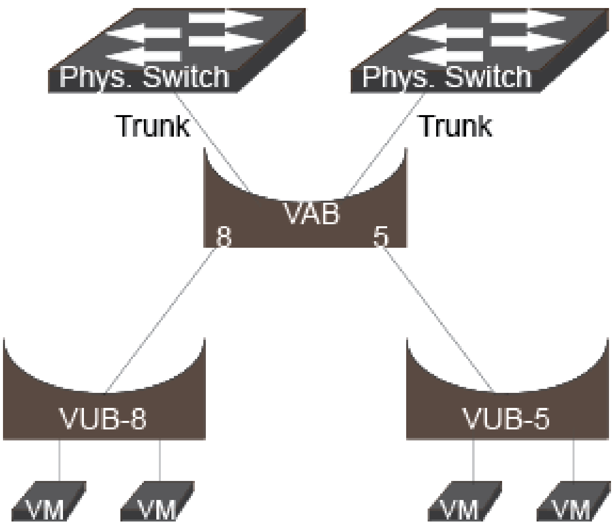
You can configure MidoNet's bridge virtual ports with a single VLAN ID, thereby introducing a change in the behavior of the processing of VLAN-tagged frames.

This guide refers to a bridge that owns one or more VLAN-configured ports as a "VLAN Aware Bridge" or VAB. A VAB may contain multiple virtual ports configured with a VLAN ID, but these tags must be unique in the VAB. A VAB may contain any number of trunk ports (that is, ports configured to support VLAN trunk links), but only one of the ports is expected to be active, according to the high-availability mechanism.

This guide refers to any bridge that has no virtual ports configured with a VLAN ID as a "VLAN Unaware Bridge" or VUB. VUBs may only have one virtual port linked to a VAB.

The diagram below depicts a typical L2 gateway topology.

Figure 16.1. Topology with VLANs and L2 Gateway



The VAB in this example has two trunk ports that are bound to physical interfaces (either on the same or different physical hosts). Each of these ports has L2 connectivity to physical switches and may carry VLAN-tagged traffic.

The VAB also has two other virtual ports, both configured with different VLAN-IDs. These virtual ports are linked to peers at two VUBs (VUB-8 and VUB-5), which in turn are linked to two VMs through two virtual ports each.

The VAB examines the VLAN tags of all traffic ingressing from the two trunk ports. For frames tagged with the same VLAN-ID as any of its other virtual ports (in the example, 5 or 8), the VAB removes the VLAN-ID and sends the frames to the appropriate port. For frames ingressing the VAB from a virtual port configured with a VLAN-ID, the VAB adds the corresponding VLAN-ID to the frames, and then sends the frames to the appropriate trunk port according to the bridge's MAC-port table.

Note

Neutron networks can only be mapped to the VUBs, so the VMs linked to the VABs are managed outside of OpenStack. This means that IP address management in Neutron cannot be used for the VMs on VABs.

Configuring an L2 gateway

Use this procedure to configure an L2 gateway.

Below is an example of a configuration showing how to use the MidoNet CLI to replicate the topology shown in Figure 16.1, "Topology with VLANs and L2 Gateway" [105].

1. Create the VAB and two ports configured with the appropriate VLAN IDs:

```
midonet> bridge create name vab
midonet> bridge bridge0 port add vlan 8
bridge0:port0
midonet> bridge bridge0 port add vlan 5
bridge0:port1
```

2. Create the two VUBs and their virtual ports:

```
midonet> bridge add name vub-8
bridge1
midonet> bridge add name vub-5
bridge2
midonet> bridge bridge1 port add
bridge1:port0
midonet> bridge bridge2 port add
bridge2:port0
```

3. Link the ports:

```
midonet> bridge bridge0 port port0 set peer bridge1:port0
midonet> bridge bridge0 port port1 set peer bridge2:port0
```

4. Add the VAB's trunk ports:

```
midonet> bridge bridge0 port add
bridge0:port2
midonet> bridge bridge0 port add
bridge0:port3
```

5. Assuming that there are two interfaces, host0:eth0 and host1:eth1, connected to the physical switches' trunks, bind them to the VAB's trunk ports:

```
midonet> host host0 binding add interface eth0 port bridge0:port2
```

```
midonet> host host1 binding add interface eth1 port bridge0:port3
```

Fail-over/Fail-back

In combination with the Spanning Tree Protocol (STP) enabled on the physical bridges, MidoNet VABs are able to provide fail-over capabilities by forwarding Bridge Protocol Data Unit (BPDU) frames across their trunk ports.

Assuming that both physical switches belong to the same bridged network, as a result of the STP, both devices detect a loop through MidoNet's VAB and one switch chooses to block its trunk. For example, let's assume the left switch blocks. The VAB only sees ingress traffic from the right trunk, and thus associates all source MAC addresses seen in those frames to the right trunk.

A variety of events, including failures in the network, may result in the switches deciding to invert the state of the trunks. An example could be MidoNet losing connection to the left switch, and thus stop forwarding BPDUs to/from the right bridge and undoing the loop.

In such a fail-over scenario, traffic would start flowing from the other switch. With this change, MidoNet now detects ingress traffic on a new port, and thus updates its internal MAC-port associations. If the former state of the topology is restored (that is, MidoNet recovers connectivity to the left switch), MidoNet will again react and update its MAC-port associations.

The fail-over/fail-back times depend on the STP configuration on the switches, mainly the "forward delay," and the nature of the traffic. With standard values, and continuous traffic ingressing from the trunks, fail-over and fail-back cycles should be completed in 50 seconds, plus MAC learning time.

17. Service Containers

Table of Contents

MidoNet supports the network functions virtualization, such as L4LB, VPN or BGP, using third party software that executes in namespaces connected to the MidoNet virtual topology. To make the integration of these components easier and to control the namespace scheduling, MidoNet 5.1 introduces the concept of service containers. Currently, they are used only for VPNaaS.

Service containers are Linux namespaces with a port connected to a MidoNet virtual device, and they run a set of programs that are configured according to the type of the service container.

Service containers are launched and destroyed automatically by the MidoNet Cluster with the topology object to which they correspond. For example, an IPSec container, used to encrypt traffic for VPN connections and handle the security associations, launches when creating the first Neutron `VpnService` on a router, and it is destroyed when deleting the last.

Container scheduling on the available physical computes is also managed by the MidoNet Cluster, and it takes into account the computes' availability, one of the several configurable scheduling policies, and a per-compute host container weight parameter allowing the cloud operator to differentiate the container load between different computes.

Architecture

A service container is an object that exists in the MidoNet topology, like a virtual device or port. When created, it is linked to an existing virtual port, which becomes the *container port* through which the container namespace exchanges traffic with the other devices inside the MidoNet virtual topology.

A *service type* describes the container namespace and the processes that are executed inside it, while a *configuration identifier* keeps a reference to an object in the MidoNet topology that indicates how the namespaces and the container processes are configured.

The container *scheduling policy* is specified using a *service container group* object. Therefore, several containers can share the same scheduling policy by referencing the same container group. The MidoNet Cluster uses the policy as defined by the container group to determine the physical compute where a container is launched. This scheduling is controlled via the binding of the container port on a given host. For more information see the section called “Scheduling Policies” [115].

In addition, a service container group specifies an *allocation policy*, which specifies how existing compute hosts are allocated to new containers given the host pool determined from the *scheduling policy*. Whereas the scheduling policy considers configurable properties of the hosts, such as membership to a particular host or port group, the allocation policy uses live attributes of the hosts, such as their *container weight* or number of launched containers. For more information see the section called “Allocation Policies” [116].

When a MidoNet Agent launches a scheduled container, it runs a set of routines that are customized for a particular container type, and which configures and starts the container processes. These can be loaded from external libraries, allowing the cloud operator to deploy additional containerized services as needed, without necessarily installing a new version of the agent.

MidoNet agents that run service containers use the same container library to monitor and report the container status to the NSDB. This allows the MidoNet Cluster to report it via the REST API, and to drive the container scheduling upon a container failure.

General Fields

The following table lists the fields that describe a service container in the MidoNet topology.

Field Name	Type	Description
`id`	UUID	The service container identifier.
`serviceType`	String	A unique name that identifies the type of the service container.
`groupId`	UUID	The identifier of the service container group that describes the scheduling policy.
`portId`	UUID	The identifier of the container virtual port. This is an exterior port connected to a device from the virtual topology. The port’s binding information describes the compute host where the container is launched.
`configurationId`	UUID	The identifier of an object from the MidoNet topology that provides the container configuration. The type of this object is specific for a particular service type, and the agent library handling that service type is responsible for giving it the proper interpretation.

Status Fields

The following table lists the fields that describe the status of a service container.

Field Name	Type	Description
`statusCode`	String	Indicates the running state of a container, it can be one of the following: `starting`, `running`, `stopping`, `stopped` and `error`.
`statusMessage`	String	A custom status message, usually containing information about the processes running inside the container.
`hostId`	UUID	The identifier of the agent host where the container is running. This field is present only for starting, running or stopping, containers.
`namespaceName`	String	The name of the Linux namespace hosting the container.
`interfaceName`	String	The name of the virtual Ethernet interface connected to the container.

Container Group Fields

The following table lists the fields that describe the service container group. For more information on the service container scheduling and scheduling policies see the section called “Scheduling” [112].

Field Name	Type	Description
`id`	UUID	The service container group identifier.

Field Name	Type	Description
hostGroupId	UUID	The identifier of the host group when using host group scheduling policy.
portGroupId	UUID	The identifier of the port group when using port group scheduling policy.
policy	String	The hosts allocation policy for the containers belonging to this group.

Configuration

The containers are configured using mn-conf. Both the MidoNet Agent and Cluster run a Container Service, which is responsible for launching the containers at the agent, and scheduling the containers across different computes at the cluster.

Agent Configuration

The container configuration at the agent is found under the agent.containers namespace. The following table lists the configuration keys, and modifying the value of a configuration key requires restarting the agent in order to take effect.

Field Name	Type	Description
enabled	Boolean	Whether an agent instance runs the containers service. This allows the operator to disable containers across all compute hosts or at specific compute hosts. The default is true.
timeout	Duration	The timeout for a container operation to complete. The default is 30 seconds.
shutdown_grace_time	Duration	The timeout for all containers to stop gracefully when the containers service is stopped. The default is 30 seconds.
thread_count	Integer	The number of threads that manage the containers executing at this agent. This determines the number of containers that can be changed in parallel. Once created a container is bound to the same thread on which it was created.
log_directory	String	The name of the log directory where the agent writes a record of all container operations. The log is used to cleanup containers following an agent failure. The default value is containers. The directory is created in the same directory as the agent log, which is configurable using the midolman.log.dir JVM system property in the agent startup script, and which defaults to /var/log/midolman.

Specific container types may include additional configuration keys, as follows.

IPSec Containers Configuration

The following table lists the configuration keys for IPSec containers, which are used to implement VPNaaS. All keys are found under the agent.containers.ipsec namespace.

Field Name	Type	Description
logging_enabled	Boolean	Indicates whether the IPSec container writes to the agent log the log messages reported by the IPSec processes running inside the container. The default is true. When enabled, these messages are logged using the logger name org.midonet.containers.ipsec.ipsec-pluto.
logging_poll_interval	Duration	The polling interval for the IPSec container log. The default is 250 milliseconds.
logging_timeout	Duration	The timeout when closing the log file. The default is 3 seconds.

Field Name	Type	Description
status_update_interval	Duration	The interval for updating the container status and health. The default is 5 seconds.

Cluster Configuration

The containers configuration at the cluster is found under the `cluster.containers` namespace. The following table lists the configuration keys, and modifying the value of a configuration key requires restarting the cluster in order to take effect.

Field Name	Type	Description
enabled	Boolean	Whether a cluster node runs the containers service. This allows the operator to disable containers across all cluster nodes or at specific cluster nodes. The default is `true`.
scheduler_timeout	Duration	The timeout for a container to reach `running` state when scheduled at a compute host. The default is 10 seconds. For more information on container scheduling see the section called “Scheduling” [112].
scheduler_retry	Duration	Indicates the time interval after which the scheduler attempts to reschedule a down container.
scheduler_max_retries	Integer	The maximum number of attempts the scheduler will retry to schedule a down container. Once this number is reached, the scheduler will schedule a new container only as a response to an external event, such as the available hosts have changed or the operator triggered a manual scheduling.
scheduler_bad_host_lifetime	Duration	When an agent fails to start a container within the expected timeout interval, that host is considered bad and not used for subsequent scheduling operations of the same container. This key controls for how long a host is considered bad after failing to launch a container. The default is 5 minutes. Set this value to zero (0) to disable bad hosts for scheduling. For more information on container scheduling see the section called “Scheduling” [112].

Note

If the containers service is enabled at multiple cluster nodes, only one cluster node determined by via distributed leader election will actively perform the container scheduling at a time. If the leader node fails, another node will take over the scheduling role automatically. The failover latency equals the configured ZooKeeper session timeout at `zookeeper.session_timeout`.

Management

Normally, the service containers are launched and destroyed automatically by the MidoNet Cluster. Therefore, the operator would typically use the MidoNet CLI only to inspect the containers configuration and running state.

Listing the Service Containers

To list information about all service containers, enter the command:

```
midonet> list container
container container0 container-group cgroup0 configuration 63c6fa43-
abcd-43b0-870b-c72ab6f2c4dc port port0 type IPSEC status running host host0
 namespace vpn-e9a728b2 interface vpn-e9a728b2
container container1 container-group cgroup1 configuration
 401994b8-4fec-428d-9133-40c51988578a port port1 type IPSEC status running
 host host0 namespace vpn-a55d15fe interface vpn-a55d15fe
```

The example above shows two service containers of `IPSEC` type that have been scheduled at host `host0` and are currently in the `running` state. If the containers are not running, then the output of the command would be as follows:

```
midonet> list container
container container0 container-group cgroup0 configuration 63c6fa43-
abcd-43b0-870b-c72ab6f2c4dc port port0 type IPSEC status stopped
container container1 container-group cgroup1 configuration
 401994b8-4fec-428d-9133-40c51988578a port port1 type IPSEC status stopped
```

Listing the Service Container Group

To list information about all service container groups, enter the command:

```
midonet> list container-group
cgroup cgroup0 policy least
cgroup cgroup1 policy least
cgroup cgroup2 policy least
```

Viewing the Container Status Message

The container status message is usually a text with multiple lines, and therefore it is not included in the container listing. Therefore, to view the message, enter the command:

```
midonet> container container0 show status-message
000 using kernel interface: netkey
000 interface lo/lo ::1@500
000 interface lo/lo 127.0.0.1@4500
000 interface lo/lo 127.0.0.1@500
000 interface vpn-e9a728b2-ns/vpn-e9a728b2-ns 1.0.0.2@4500
000 interface vpn-e9a728b2-ns/vpn-e9a728b2-ns 1.0.0.2@500
000 interface vpn-e9a728b2-ns/vpn-e9a728b2-ns 169.254.0.2@4500
000 interface vpn-e9a728b2-ns/vpn-e9a728b2-ns 169.254.0.2@500
...
```

Note

You can also use the `show` command for any other fields of a service container or service container group object.

Warning

MidoNet also provides `create` and `delete` commands for the service containers. However, these should not be used under normal circumstances and should be reserved only for troubleshooting purposes.

Scheduling

The MidoNet Cluster uses the Containers Service to schedule existing service containers at different compute hosts. Container scheduling considers the following input data:

- The MidoNet agent must be running at the compute host, and it must have the Containers Service enabled. See the section called “Agent Configuration” [110] for how to enable or disable the service at the agent.
- The configuration of the agent host in the NSDB should have a positive value for the *service container weight*. See the section called “Host Container Weight” [114].
- The configuration of the agent host in the NSDB should have a positive value for the *service container limit*. See the section called “Host Container Limit” [114].

- The container service scheduling policy as defined by the service container group. See the section called "Scheduling Policies" [115].
- The container service host allocation policy as defined by the service container group. See the section called "Allocation Policies" [116].
- The container status reported by the last host where the container has been scheduled.

Scheduling is performed at the container level, and it considers the following possible states for a container:

State	Description
`DOWN`	The container is not scheduled at a host.
`SCHEDULED`	The container is scheduled at a host, but that host has not yet reported the container status.
`UP`	The container is scheduled at a host, and that host has reported the container status as `running`.

Only `DOWN` containers are being scheduled, and scheduling is performed when the container is created and subsequently whenever the available hosts and their *state* changes. By host state, we understand whether the agent at that host is running the container service and its corresponding *container weight*.

A host is selected to run a container if it is *eligible*, that is it meets the first three criteria from above about running the container service, having a positive *container weight*, does not currently run more containers than its *container limit*, and if the host has not been previously marked as *bad* for the same container. A *bad* host is a host that has previously failed to launch a container. This may happen when the host is overloaded, or it does not have installed the necessary software that is needed to launch the container. Marking the hosts as *bad* is a temporary action, and it prevents the rescheduling logic for retrying indefinitely with the same agent. The interval during which a host is marked as bad after failing to launch a container is configurable in the cluster containers configuration. See the section called "Cluster Configuration" [111].

Important

The values of the `scheduler_timeout` and `scheduler_bad_host_lifetime` configuration keys are paramount to ensure that containers are scheduled properly, and their values should consider both the acceptable downtime during a container or agent failure and the time required to start the container at the host. Some containers, such as the `IPSEC` containers, requires a non-negligible time interval to start, and therefore setting the container timeout to a very low value coupled with a long bad host lifetime will lead to containers not being scheduled. This situation may occur during the bulk creation of containers with a small number of compute hosts. In such cases, we recommend increasing the `scheduler_timeout` or setting the `scheduler_bad_host_lifetime` to zero.

A host is selected from the eligible hosts using the current allocation policy, specified by the service container group. See the section called "Allocation Policies" [116].

Note

Running containers are not affected by changes to a host's *container weight*, except when it is set to zero.

After a container has been scheduled, if an agent fails to report the container as `running` within the allowed scheduler timeout, the scheduling logic is repeated. The service also monitors continuously the host and container state, such that the container is re-scheduled when any of the following situations occurs.

- The host container service is not running, because the agent has been shut down or has failed.
- The host *container weight* is set to zero.
- The container is in `DOWN` state and the list of eligible hosts has changed.
- The container is in `SCHEDULED` state and the scheduling timeout has expired.
- The container is in UP state and the container state is `stopping`, `stopped` or `error`.

Host Container Weight

You can use the container weight of a host to determine whether is is eligible to run service containers, and the weight it should be given when selecting it during container scheduling.

To view the container weight assigned to all hosts, use the command:

```
midonet> host list
host host1 name host1 alive true addresses 10.0.0.1 flooding-proxy-weight 1
 container-weight 0 container-limit no-limit
host host2 name host2 alive true addresses 10.0.0.2 flooding-proxy-weight 1
 container-weight 5 container-limit no-limit
host host3 name host3 alive true addresses 10.0.0.3 flooding-proxy-weight 1
 container-weight 10 container-limit no-limit
```

To set the container weight to a different value, use the command:

```
midonet> host host1 set container-weight <new-value>
```

Note

Modifying the container weight on hosts takes an immediate effect on the container scheduling. If the weight is set to zero, all containers that are running or have been scheduled to run at that host will be migrated to other eligible hosts.

Host Container Limit

Compute hosts can be configured to run up to a configurable number of service containers. By default, a host does not have an upper limit for how many container can be launched.

To view the container limit assigned to each host, use the command:

```
midonet> host list
host host1 name host1 alive true addresses 10.0.0.1 flooding-proxy-weight 1
 container-weight 0 container-limit no-limit
host host2 name host2 alive true addresses 10.0.0.2 flooding-proxy-weight 1
 container-weight 5 container-limit no-limit
host host3 name host3 alive true addresses 10.0.0.3 flooding-proxy-weight 1
 container-weight 10 container-limit 100
```

In the host listing, the `container-limit` property indicates the container limit for each host. For the hosts that do not have a container limit, the value of the property is `no-limit`, whereas for the hosts with a limit the property is an integer representing the maximum number of service containers that can be launched by that host.

To set the container limit to a different value, use the command:

```
midonet> host host1 set container-limit <new-value>
```

Note

In this version of MidoNet the container limit is enforced only for new containers launched by a host, and existing containers are not affected. This is unlike that container weight, which takes immediate effect when set to zero, forcing the migration of all containers running at that host.

Important

If you want to gracefully migrate all container from a host, for example in order to take down a compute host for maintenance, use the `container-weight` property of a host rather than the `container-limit`. The latter is intended to established a container capacity for the hosts and it does not trigger a container migration.

To unset the container limit for a host use the command:

```
midonet> host host1 clear container-limit
```

Scheduling Policies

MidoNet supports three scheduling policies for service containers. Each policy is configured by assigning the container to an appropriate service container group.

- *Anywhere Policy*: This policy corresponds to service container groups that set neither `hostGroupId` nor `portGroupId`. With this policy containers are scheduled at any of the MidoNet hosts that run the container service and have a positive container weight.
- *Host Group Policy*: Containers with this policy are scheduled only at the hosts that are member of the host group set in the service container group. Hosts must still meet the eligibility requirements to be selected.
- *Port Group Policy*: Containers with this policy are scheduled only at the hosts bound to the exterior ports that are members of the port group set in the service container group. Hosts must still meet the eligibility requirements. When using the port group policy, the set of hosts will change when ports migrate between hosts. Therefore, this policy ensures that containers will be scheduled at the same compute hosts where the ports in the port group are located. For more information on port groups, see the section called “Stateful port groups” [18].

Important

To ensure host-to-host traffic, all eligible hosts for a particular scheduling policy must be configured in the same tunnel zone. The Containers Service does not use the tunnel zone membership as an eligibility requirement during the host selection.

Allocation Policies

MidoNet supports two host allocation policies for service containers. These policies specify how a host is selected to launch a container from the pool of hosts determined by the scheduling policy. For a host to be selected with a given allocation policy, it otherwise must meet the eligibility requirements for launching a new container: it has a positive container weight, it does not exceed its container limit, it is not marked as a bad host, it is active and it currently runs the containers service.

- *Least Scheduler*: The `least` policy selects the host from the pool that has the smallest number of containers.
- *Weighted Scheduler*: The `weighted` policy selects a host from the pool randomly, with each host having a probability of being selected proportional to its container weight relative to all other hosts. For example, three hosts with the container weights 1, 3 and 4 will have 12.5%, 37.5% and 50% probability in the statistical sense, respectively, of being selected to run a container.

You can view the current allocation policy by typing:

```
midonet> list container-group
cgroup cgroup0 policy least
cgroup cgroup1 policy least
cgroup cgroup2 policy weighted
```

To change the current allocation policy, use:

```
midonet> container-group cgroup0 set policy <policy-name>
```

Setting the allocation policy will apply to scheduling new containers belonging to this service container group, and whenever existing containers are being rescheduled. However, running containers are not affected.

Manual Scheduling

By default, the MidoNet Cluster manages the service container scheduling automatically for all new service containers using both the scheduling policy and the allocation policy specified by the service container group. Launching the service containers once created is therefore transparent to the user, who does not have to take any other action.

However, sometimes you may want to control the location of a container manually, for example to troubleshoot a problem with a particular container. When this is the case, MidoNet offers you the possibility to control the container scheduling at the service container granularity.

To this end, the service container API allows you to *suggest* a host to launch a given service container. If the suggestion is accepted by the service container scheduler, the container will be immediately migrated to the selected host.

To request that a service container be scheduled at a particular host, type:

```
midonet> container <container-id> set host <host-id>
```

Important

The manual scheduling of a container is only a request by the operator to MidoNet to run a service container at a particular host. The request will be ignored if the host does not meet the eligibility criteria to launch the container, or if the host fails to start the container. When this is the case, the MidoNet container scheduling algorithm will migrate the container at an alternative host or leave the container at its current location.

It is also possible to request to the MidoNet container scheduler to re-trigger the automatic scheduling for a particular service container. This allows you to troubleshoot a malfunctioning container but relying on the automatic scheduling to select the container location.

To re-trigger the automatic scheduling for a service container, type:

```
midonet> container <container-id> clear host
```

Troubleshooting

Agent Troubleshooting

At the MidoNet Agent, the service containers are managed by a Container Service. This service monitors all port bindings corresponding to the local host. When a bound port is detected as being connected to a container (this is done by checking the `containerId` field from the port object), the service loads the container data, and initiates a container state machine that tracks the current container state.

The container state transitions are the following:

- *Created* when the container is created for a new port binding, or an existing port binding references a new container.
- *Updated* when the container data changes, which includes the *service type*, *container group* and *configuration*.
- *Deleted* when the port binding for a container is deleted, or an existing port binding references a new container.

During the initialization the Container Service searches in the current agent class-path the implementation for all supported containers. These can be either containers packaged with the agent, or containers installed as stand-alone libraries. The following table lists the container implementations that packaged with the MidoNet Agent.

Service Type	Container Version	MidoNet Version	Description
IPSEC	1	>= 5.1.0	Implements an IPSec endpoint for VPNaaS.

Containers are uniquely identified by their service type. If several implementations of the same container are installed, the agent selects the highest version. You can use the agent log, with the logging level set to `INFO` to determine the container implementations loaded by the Container Service during initialization. The logger name is `org.midonet.containers`.

```
INFO  [main] containers -  Scanning classpath for service containers
INFO  [main] containers -  Service container: IPSEC version 1
INFO  [main] containers -  Starting Containers service for host <hostId>
```

Troubleshooting Checklist

If a container fails to start at a designated compute, use the following checklist to troubleshoot the cause of the problem.

- Use `mn-conf` to set the logging level for the container to DEBUG:

```
mn-conf set -t default 'agent.loggers.org.midonet.containers : DEBUG'
mn-conf set -h <hostId> 'agent.loggers.org.midonet.containers : DEBUG'
```

- Check the Container Service is enabled in the agent configuration. The configuration is logged during the agent start-up.

```
"containers" : {
    "enabled" : true,
    "scheduler_bad_host_lifetime" : "300s",
    "scheduler_timeout" : "10s"
}
```

- Check in the agent log that the Container Service has started:

```
DEBUG [containers-1] containers -  Container service is running with weight
 1
```

- Check the *container weight* for the agent is a positive integer. Agents with zero weight are not allowed to start containers. If the container weight is zero, use the MidoNet CLI to set a positive weight for that agent's host.

```
midonet> host host0 set container-weight 1
```

- Check the agent detects the port binding for the corresponding container port. You can determine the container port using MidoNet CLI. If the binding is not detected, use the cluster log to determine whether the container has been scheduled at this host.

```
DEBUG [devices-service-1] containers -  Host <hostId> bindings updated:
 Set(<portId>)
```

- Check the agent loads the container data from the NSDB, and starts the process to create the container. The *service type* must match one of the container types detected during the initialization, for example `IPSEC`.

```
INFO  [containers-1] containers -  Create container for port binding
 ContainerPort{portId=<portId>, hostId=<hostId>, interfaceName=<...>,
 containerId=<containerId>, serviceType=<serviceType>, groupId=<...>,
 configurationId=<...>}
```

- Check the service calls the `create` method for the container implementation of the current service type. The message logged for this event is specific for a given service type. For instance, the IPSec container displays the following log message. The logger name for this event depends on container implementation, in the case of IPSec containers is: `org.midonet.containers.ipsec`.

```
INFO  [containers-1] ipsec -  Create IPSec container for
 ContainerPort{portId=<portId>, hostId=<hostId>, interfaceName=<.
..>, containerId=<containerId>, serviceType=IPSEC, groupId=<...>,
 configurationId=<...>}
```

The container implementation may log additional messages, indicating the container configuration and the processes started inside the container.

- When updating a container, check the service detects the update. The *service type* must match one of the container types detected during the initialization, for example `IPSEC`.

```
DEBUG [devices-service-1] containers -  Container <containerId> updated
 with type <serviceType> group <groupId> configuration <configurationId>
```

- When deleting a container, check the service detects the deletion, and that it calls the `delete` method for the container implementation of the current service type.

```
DEBUG [devices-service-1] containers -  Container <containerId> deleted for
 binding PortBinding{portId=<portId>, hostId=<hostId>, interfaceName=<...>,
 containerId=<containerId>} upon completion
INFO  [containers-1] containers -  Delete container ContainerPort{portId=
<portId>, hostId=<hostId>, interfaceName=<...>, containerId=<containerId>,
 serviceType=<serviceType>, groupId=<...>, configurationId=<...>}
```

The container implementation may log additional messages detailing the container tear-down, such as stopping the container processes. For example, calling the `delete` method for an IPSec container results in the following log messages.

```
INFO  [containers-1] ipsec -  Deleting IPSec container <name>
INFO  [containers-1] ipsec -  Cleaning up IPSec container <name>
```

Containers Log

The MidoNet Agent creates a local record of the running containers. The main purpose of this log is to track the containers in the case of an agent failure, and perform a cleanup during the next agent restart. However, the containers log may also serve to the purpose of troubleshooting, helping to identify the active containers.

The location of the containers log directory is configurable using the `midolman.log.dir` JVM system property in the agent startup script, and it default to `/var/log/midolman`. The name of the logs directory is configurable using `mn-conf` at the configuration key `agent.containers.log_directory` and it defaults to `containers`.

The containers log directory contains a file for each container started by the local agent, where the file name is `<container-name>.<service-type>`. For example, the following agent runs two IPSec containers.

```
# ls /var/log/midolman/containers/ -la
total 8
drwxr-xr-x 2 root root 4096 Feb 22 18:42 .
drwxr-xr-x 3 root root 4096 Feb 22 18:13 ..
-rw-r--r-- 1 root root    0 Feb 22 18:42 vpn-60df5fda.IPSEC
-rw-r--r-- 1 root root    0 Feb 22 18:42 vpn-dc56189c.IPSEC
```

Logging the Container Processes

The process or processes running inside a container may provide additional logging information. Where this is applicable, the container implementation may relay those logging messages to the agent log using the same or a different logger name. The logging data can be read from local file system, or it can be exchanged using a named pipe (FIFO file).

The IPSec container uses a named pipe to read the logging data of the IPSec process, *pluto*, executing in the container. The log messages are appended to the agent log using a different logger name, `org.midonet.containers.ipsec.ipsec-pluto`, and the logging level set to DEBUG.

```
DEBUG [io-1] ipsec-pluto -  <timestamp>: NSS DB directory: sql:/etc/ipsec.d
DEBUG [io-1] ipsec-pluto -  <timestamp>: NSS initialized
DEBUG [io-1] ipsec-pluto -  <timestamp>: Starting Pluto (Libreswan Version
 v3.14-dirty-master XFRM(netkey) KLIPS NSS DNSSEC XAUTH_PAM NETWORKMANAGER
 CURL(non-NSS) LDAP(non-NSS)) pid:21787
```

18. Service Insertion and Chaining

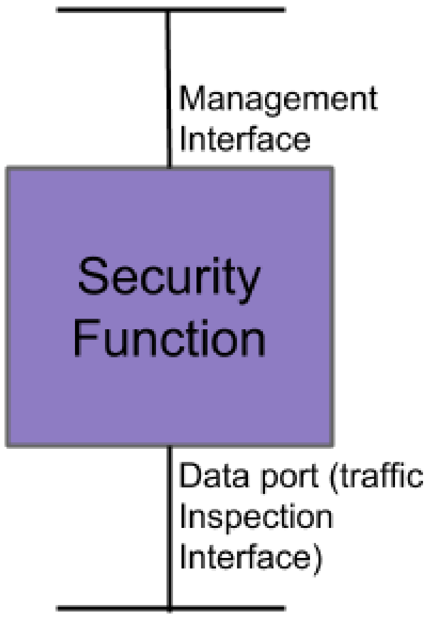
Table of Contents

Starting in version 5.0, MidoNet has a **Technical Preview** API for inserting and chaining network service functions. The API is mostly provided by a single object, L2Insertion, whose fields are documented in the REST API documentation.

Service Function Model

The L2Insertion assumes a Service Function has:

- an addressable management interface (not managed or referenced from L2Insertion)
  - Assign a Floating IP to allow access from outside the cloud.
- an unnumbered inspection interface
  - receives the traffic of inspected VMs
  - MidoNet can add a VLAN tag to signal policy, and PCP bits indicate direction (in-to/out-of VM)



Note that MidoNet drops all non-insertion traffic on an inspection port of a network function.

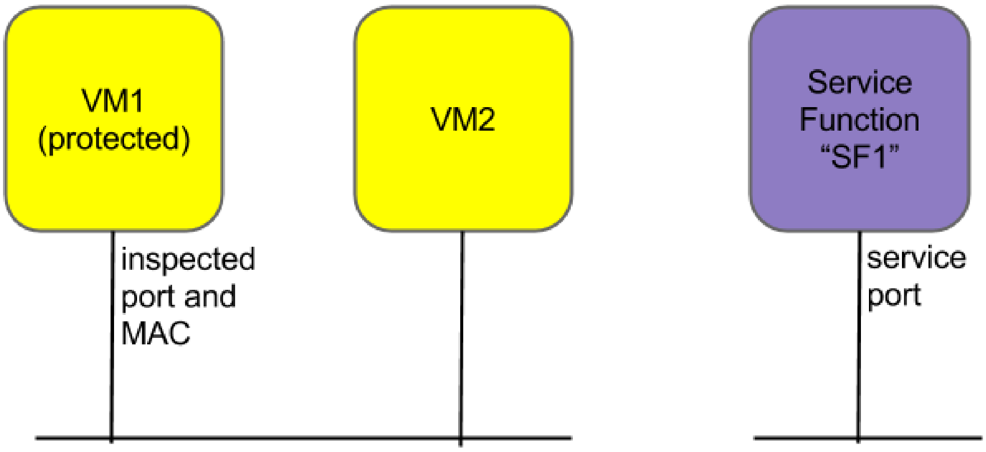
L2Insertion Object

The L2Insertion object has these fields (see the REST documentation for more detail):

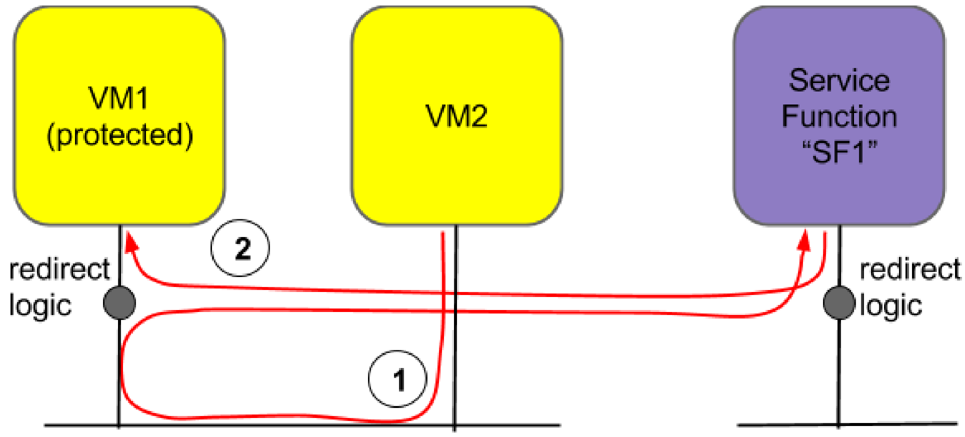
1. inspected VM port UUID
2. inspected VM MAC
3. service port UUID
4. VLAN tag
5. fail-open (true/false)
6. position

The first 3 fields are the most important to consider. Imagine an L2Insertion configured as described by the following diagram.

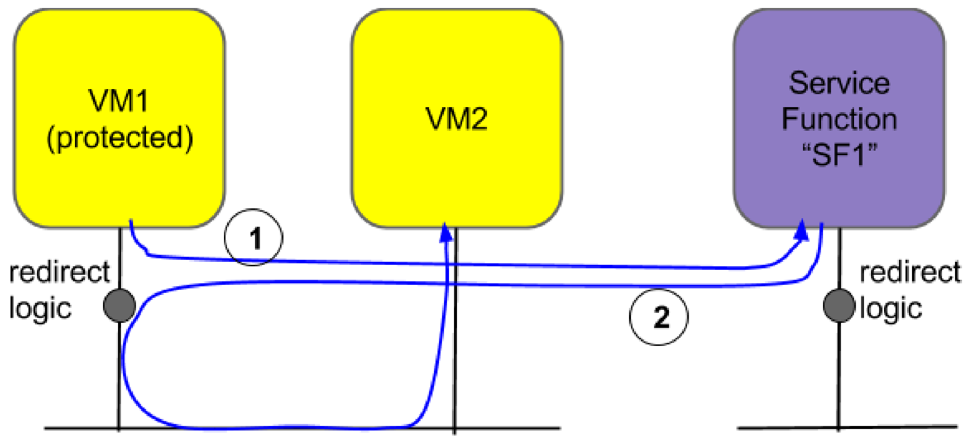
The position is an absolute position in the list of insertions. If two insertions have the same position the order is undefined. When we load the insertions we basically sort by the position field. In many ways, it acts more like a weight.



In such a scenario, MidoNet will set up redirect logic as shown in the diagram below. When VM2 sends a packet to VM1 (and specifically to VM1's MAC), at the point labelled "redirect logic" (after port-level firewall, not shown) the packet is redirected to the service function (red arrow 1). If the service function allows the packet and therefore re-emits it, the redirect logic at the service function redirects the packet so that it completes its journey to VM1 (red arrow 2).

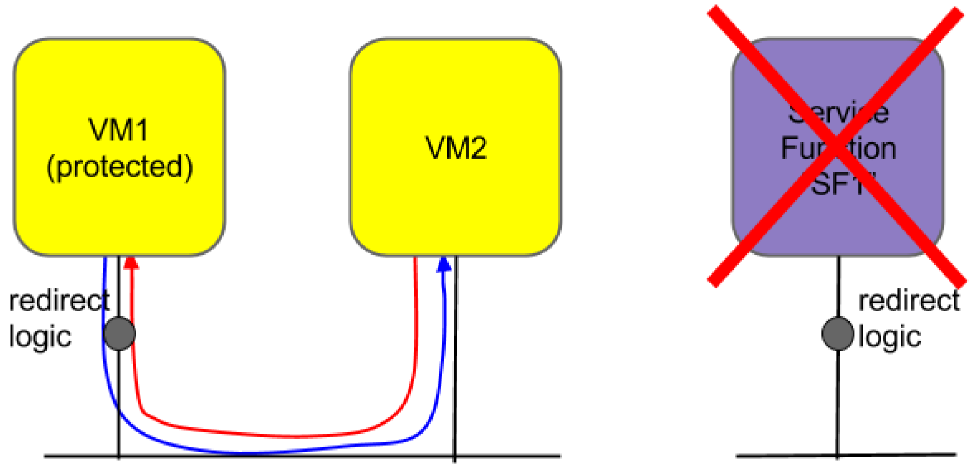


The next diagram shows how packets from the protected port (and MAC) are redirected.



The diagrams do now show how field number 4, the VLAN tag, is used. If the VLAN tag is set to a value above zero, then MidoNet will push a VLAN tag with that value onto packets redirected into the service function. MidoNet will also expect the VLAN tag to be left on the packet if it is re-emitted from the service function. MidoNet also uses the PCP bits in the VLAN header for its own purposes, these should not be modified by the service function.

The use of field number 5, fail-open, is illustrated in the diagram below. If fail-open is false, then when the service function fails all redirect traffic is essentially dropped. If fail-open is true, there are cases when MidoNet can detect failure of the service function link (link status only, we do not yet have active monitoring neither in-band or out-of band) and the redirection logic will automatically allow the traffic to skip the failed service function.



Finally, the sixth field of L2Insertion, position, allows specifying the order in which service functions should be traversed if there are multiple L2Insertion objects with the same inspected port and mac but different service ports.

Important

You can set L2Insertion's "port" (inspected port) field to a non-Neutron port's UUID, like that of a router port or a L2GW port. But remember that it's not the device, but the endpoint connected to the device (router or bridge) that's protected (and therefore whose MAC should be configured in the L2Insertion). If you want to protect a router port, you must put the L2Insertion object on the bridge port facing the router port. Therefore it's not possible to protect the exterior ports of a router (like those of the Edge Router).

L2Insertion in MidoNet CLI

The following example inserts two service functions (located on Bridge1's port0 and port1) at Bridge0's port0. Only traffic to/from MAC fa:16:3e:a4:dd:58 is protected. Different VLANs are used for the different service functions - this is not required by MidoNet, the example assumes that the VM is protected by different policies (or policies represented by different VLAN tag values) in the different service functions.

```
midonet> l2insertion add fail-open true mac fa:16:3e:a4:dd:58 port
 bridge0:port0 pos 0 srv-port bridge1:port0 vlan 10
l2insertion0
midonet> l2insertion add fail-open true mac fa:16:3e:a4:dd:58 port
 bridge0:port0 pos 1 srv-port bridge1:port1 vlan 20
l2insertion1
midonet> l2insertion list
l2insertion l2insertion0 port bridge0:port0 srv-port bridge1:port0 vlan 10
 pos 0 fail-open true mac fa:16:3e:a4:dd:58
l2insertion l2insertion1 port bridge0:port0 srv-port bridge1:port1 vlan 20
 pos 1 fail-open true mac fa:16:3e:a4:dd:58
```

Alternatives

These alternative mechanisms are often used in service-chaining use-cases:

With **port mirroring** the service function receives a copy of the traffic's packets. Compared to L2Insertion:

- Only L2Insertion supports VLAN tagging; port mirroring does not.
- Only port mirroring supports classification; L2Insertion does not.
- Both allow multiple service functions (but the term "chain" would only be accurate for L2Insertion because in the case of port mirroring each service function would receive a separate copy of the packets)
- Both allow the service function to be inserted at any port in the logical topology.

You can find more information about port mirroring in the section called “Port mirroring” [83].

MidoNet also supports **source routing** (a flavor of policy routing). Source routes can be used to steer traffic through a router differently depending on where the traffic is coming from. MidoNet’s source routing matches on the source IP address of the packet headers, NOT the port at which the packet ingresses the router. Source routing is a traditional way (popular in physical networks) of inserting a network service function when traffic between two endpoints must traverse a router.

MidoNet also supports ECMP (Equal Cost MultiPath) routing. ECMP allows the forwarding table to contain multiple routes with identical source, destination and weight (but differing in their next hop). ECMP can therefore be used to load-balance flows to several service function instances.

MidoNet selects a route by:

1. Find the matching routes whose destination prefix mask is longest (longest prefix match, LPM)
2. Filter for routes whose source matches the packet.
3. Filter for route(s) that has/have the highest weight.
4. Perform ECMP calculation (hash mod N) to select 1 of the routes.

More information about routing in MidoNet can be found in Chapter 6, “Routing” [22].

The limitations of MidoNet’s ECMP load-balancing are further described in the section called “ECMP Limitations” [27].

19. Router Peering

Table of Contents

The router peering feature of MidoNet provides overlay connectivity between multiple sites with VXLAN tunneling. The following section describes the Neutron CLI and REST API commands to set up router peering. All of the following operations are executed only by the cloud operator.

The following terminologies are used throughout this document:

- VTEP Router:

  Admin owned routers that are VXLAN endpoints in both sites. They are the virtual gateway devices of each site that enables site-to-site connectivity using VXLAN technology.

- Gateway Device:

  An abstraction of a gateway device in a cloud. A VTEP router is an example of a gateway device. This abstraction lets us define other types of gateway devices in the future, such as hardware VTEP.

- Tunnel IPs:

  The IP addresses used by the gateway device when constructing VXLAN header (the outer header). These IP addresses are known only to the gateway devices. It is also important to note that these are not associated with any particular port, which allows VXLAN traffic to egress/ingress from any edge router port.

- Multi-Site Network:

  Admin-owned network that stretches across multiple sites. Another terminology used to refer to this network is "Inter-AZ Network". This network contains a private CIDR allocated for the tenant that emulates a single L2 segment uplink network of their routers, which in actuality maps to networks across sites (with the same CIDR).

- Peer Router:

  A regular tenant router that connects to the multi-site network. This router typically already has an external network attached for Internet. In the router peering scenario, there will be an additional router interface that connects to the multi-site network.

- Peer Router Port:

  A router interface port created on the multi-site network connecting it to the peer router. These ports on remote sites become the next hops for the local peer router over the multi-site network to handle traffic for any private tenant network attached to the peer routers.

- VNI:

  A 24-bit VXLAN Network Identifier, which is used to identify a VXLAN network. Each multi-site network is associated with a VNI (when attaching it to a gateway device), and currently no two multi-site networks can share a same VNI per gateway device (but this limitation may be lifted in the future if needed).

- Remote MAC Table:

  A table on a gateway device that contains the mappings of VNI, MAC, and tunnel IP. When a gateway device receives a VXLAN packet, it looks up this table to determine what the VNI and destination IP address should be on the VXLAN header. Thus, the MAC address and the tunnel IP are of the remote sites. MAC address refers to the MAC address assigned to the remote peer router port, which is also the destination MAC address of the packets ingressing the gateway device before encapsulation. With this table, coupled with the destination remote MAC address and the VNI associated with the multi-site network that the packet ingressed from, we can determine the destination tunnel IP address to use for the VXLAN header.

- Remote-Site Port:

  MidoNet currently does not support broadcast over the cross-site VXLAN overlay network, which means that ARP does not work across sites. To get around this temporary limitation, and also to avoid broadcasting in general, we introduce a concept of a "remote-site" port. Essentially, when a peer router port is created in one site, the other site needs to be notified of the IP address and the MAC address of this port so that it can "seed" the ARP/MAC tables of the multi-site network to avoid broadcasting ARP. To accomplish this, a special port of "remote-site" type must be created on all the "remote sites" multi-site networks.

In the following example, there are two sites, Site A and Site B, each with its own MidoNet and OpenStack deployment. There are two tenants, "admin" and "tenant", representing the cloud administrator and the tenant using the cloud service.

The following setups will be configured:

Site A

Tenant Network CIDR: 10.0.0.0/24

Multi-Site Network (must match in Site B):

- CIDR: 192.168.0.0/24
- VNI: 100

Router Interface Port (Multi-Site Network < - > Peer Router):

- IP: 192.168.0.1
- MAC: 16:B7:B5:A4:57:75

Gateway Device Tunnel IP: 200.200.0.1

Site B

Tenant Network CIDR: 10.0.1.0/24

Multi-Site Network (must match in Site A):

- CIDR: 192.168.0.0/24
- VNI: 100

Router Interface Port (Multi-Site Network <-> Peer Router):

- IP: 192.168.0.2
- MAC: 6F:E4:5A:FA:8E:09

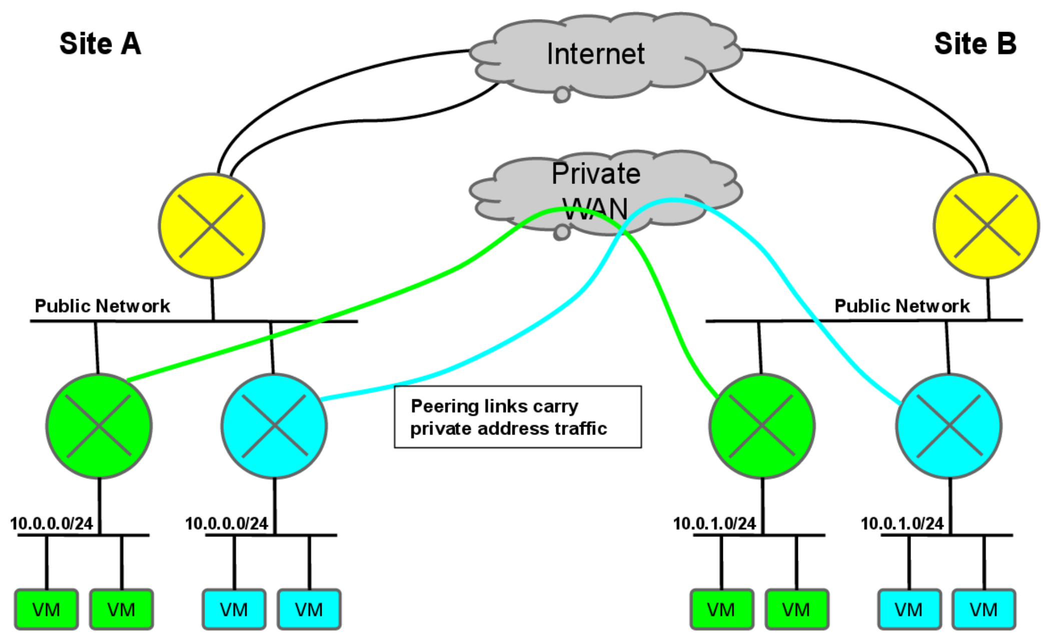
Gateway Device Tunnel IP: 200.200.0.2

Note that Gateway Device Tunnel IPs are in the same subnet in the example above, but in practice they do not need to be. They just need to be IP reachable.

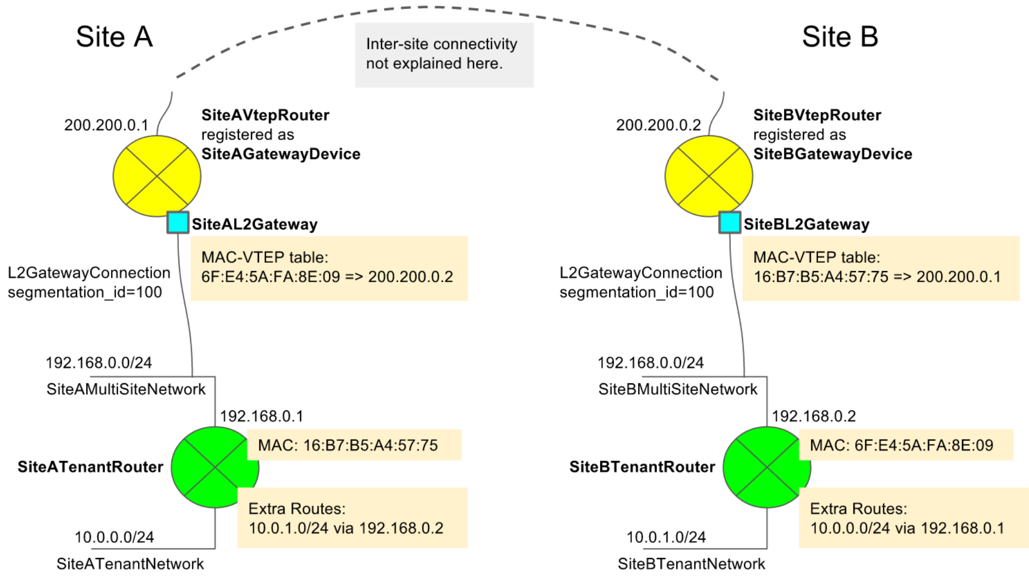
As you will see below, setting up multi-site peering requires a fairly large number of API operations and coordination among the sites that are mistake prone. Thus, it is recommended that these steps below are orchestrated and automated by an external service.

Diagrams

Overview



Details



When VM 10.0.0.3 in Site A sends an IP packet to VM 10.0.1.3 in Site B, the packet appears as follows when it egresses SiteAGatewayDevice:

- Outer Ethernet header: src is MAC of 200.200.0.1, dst is MAC of SiteAVtepRouter's peer
- Outer IP header: src is 200.200.0.1, dst is 200.200.0.2, proto is UDP
- UDP header: src is hash of inner packet, dst is standard VXLAN port
- VXLAN header: VNI is 100
- Inner Ethernet header:
  - src is 16:B7:B5:A4:57:75 (MAC of SiteATenantRouter on SiteAMultiSiteNetwork)
  - dst is 6F:E4:5A:FA:8E:09 (MAC of SiteBTenantRouter on SiteBMultiSiteNetwork)
- Inner IP header: src is 10.0.0.3, dst is 10.0.1.3

Note that SiteATenantRouter's and SiteBTenantRouter's IP addresses do not appear in the packet because they are just intermediate routers. However, traceroute from VM 10.0.0.3 to 10.0.1.3 should find these hops:

- 10.0.0.1
- 192.168.0.1
- 192.168.0.2
- 10.0.1.1
- 10.0.1.3

Traceroute will not detect SiteAVtepRouter or SiteBVtepRouter, which are part of the tunneling infrastructure.

Creating tenant networks and (peer) routers

Skip this step if you already have a tenant router and a network to do the peering with. This is here just for completeness.

Site A

```
# neutron net-create \
   --tenant-id tenant \
   SiteATenantNetwork
```

Network ID: 14ce01e6-af80-11e5-aca9-6730c321a8c7

```
# neutron subnet-create \
   --tenant-id tenant \
   14ce01e6-af80-11e5-aca9-6730c321a8c7 \
   10.0.0.0/24
```

Subnet ID: 502b8ae2-af80-11e5-8b7e-137be9ecc435

```
# neutron router-create \
   --tenant-id tenant \
   SiteATenantRouter
```

Router ID: 93adda04-af80-11e5-a1de-3f0ca86c2c1c

```
# neutron router-interface-add \
   93adda04-af80-11e5-a1de-3f0ca86c2c1c \
   subnet=502b8ae2-af80-11e5-8b7e-137be9ecc435
```

Site B

```
# neutron net-create \
   --tenant-id tenant \
   SiteBTenantNetwork
```

Network ID: dd3279be-af80-11e5-a647-67e82f86cba5

```
# neutron subnet-create \
   --tenant-id tenant \
   dd3279be-af80-11e5-a647-67e82f86cba5 \
   10.0.1.0/24
```

Subnet ID: e253ffb2-af80-11e5-bd02-c36540fab896

```
# neutron router-create \
   --tenant-id tenant \
   SiteBTenantRouter
```

Router ID: e86e6144-af80-11e5-bf25-8f151b61410e

```
# neutron router-interface-add \
   e86e6144-af80-11e5-bf25-8f151b61410e \
   subnet=e253ffb2-af80-11e5-bd02-c36540fab896
```

Creating a VTEP Router Gateway Device

Any router can be a VTEP router. A router becomes a VTEP router only when it is registered to be a gateway device as a "router_vtep" type.

Site A

```
# neutron router-create \
   --tenant-id tenant \
   SiteAVtepRouter
```

Router ID: 4bd9f56e-a792-11e5-b76a-9b045b18900a

```
# neutron gateway-device-create \
   --tenant-id tenant \
   --name SiteAGatewayDevice \
   --type router_vtep \
   --resource-id 4bd9f56e-a792-11e5-b76a-9b045b18900a \
   --tunnel-ip 200.0.0.1
```

Gateway device ID: 5f1a1a9a-a7a2-11e5-8bde-8b2506f539f8

Site B

```
# neutron router-create \
   --tenant-id tenant \
   SiteBVtepRouter
```

Router ID: 7b0bc88c-af83-11e5-9ec4-bbe5550f7e92

```
# neutron gateway-device-create \
   --tenant-id tenant \
   --name SiteBGatewayDevice \
   --type router_vtep \
   --resource-id 7b0bc88c-af83-11e5-9ec4-bbe5550f7e92 \
   --tunnel-ip 200.0.0.2
```

Gateway device ID: 8e47afce-af83-11e5-9b76-4b95f3de9711

While the `--tunnel-ip` option can be specified repeatedly, in the case of VTEP routers, only one IP address is accepted.

Creating a Multi-site Network and L2 Gateway Connection

The next step is to create a Neutron network that represents an L2 segment that spans across sites. This network must exist in both sites and must share the same CIDR and VNI. Also by associating a gateway device to an L2 gateway logical construct, it becomes an L2 gateway device.

Site A

```
# neutron net-create \
   --tenant-id tenant \
   --port-security-enabled=False \
   SiteAMultiSiteNetwork
```

Network ID: 7d107e3a-a7a3-11e5-986c-7b399d0489a5

```
# neutron midonet-l2-gateway-create \
   --tenant-id tenant \
   SiteAL2Gateway \
   --device device_id="5f1a1a9a-
a7a2-11e5-8bde-8b2506f539f8",segmentation_id=100
```

L2 Gateway ID: ea57e7e8-a7a4-11e5-89c2-b78c120baeb9

```
# neutron subnet-create \
```

```
   --tenant-id tenant \
   --no-gateway \
   --no-dhcp \
   7d107e3a-a7a3-11e5-986c-7b399d0489a5 \
   192.168.0.0/24
```

Subnet ID: 15f9485c-a7a4-11e5-850b-0b488249b443

```
# neutron l2-gateway-connection-create \
   --tenant-id tenant \
   ea57e7e8-a7a4-11e5-89c2-b78c120baeb9 \
   7d107e3a-a7a3-11e5-986c-7b399d0489a5
```

L2 Gateway Connection ID: b0aef454-a7a5-11e5-8378-b3df9d563488

Site B

```
# neutron net-create \
   --tenant-id tenant \
   --port-security-enabled=False \
   SiteBMultiSiteNetwork
```

Network ID: 8943d506-af84-11e5-bb39-e7d07e647dfc

```
# neutron midonet-l2-gateway-create \
   --tenant-id tenant \
   SiteBL2Gateway \
   --device device_id="8e47afce-
af83-11e5-9b76-4b95f3de9711",segmentation_id=100
```

L2 Gateway ID: 9514751c-af83-11e5-ac3c-affb2bc390ae

```
# neutron subnet-create \
   --tenant-id tenant \
   --no-gateway \
   --no-dhcp \
   8943d506-af84-11e5-bb39-e7d07e647dfc \
   192.168.0.0/24
```

Subnet ID: 90e9c6b2-af84-11e5-b12b-af7d2f9c54f7

```
# neutron l2-gateway-connection-create \
   --tenant-id tenant \
   9514751c-af83-11e5-ac3c-affb2bc390ae \
   8943d506-af84-11e5-bb39-e7d07e647dfc
```

L2 Gateway Connection ID: b0aef454-a7a5-11e5-8378-b3df9d563488

It is important that you specify `--no-gateway` to avoid getting an IP address reserved for it on this network. Also, you can enable DHCP (remove `-disable-dhcp`) if you plan to launch VMs directly on this network. However, in that case, be aware that an IP address will be allocated for the DHCP port.

Mandatory `segmentation_id` is the VNI, and it must be in a string format. The `devices` field of l2 gateway is a list, but only one device can be supplied.

Peering a Tenant Router

The peering steps begin by attaching the tenant router to the multi-site network in both sites. Upon attaching the tenant router to the multi-site network, an extra route must be specified on the tenant router so that the remote private subnet traffic is routed to the remote peer tenant router.

Site A

```
# neutron port-create \
   --tenant-id tenant \
   --fixed-ip ip_address=192.168.0.1 \
   --mac-address="16:B7:B5:A4:57:75" \
   7d107e3a-a7a3-11e5-986c-7b399d0489a5
```

Port ID: 2aae9544-a7aa-11e5-9fcb-1730471db09c

```
# neutron router-interface-add \
   93adda04-af80-11e5-a1de-3f0ca86c2c1c \
   port=2aae9544-a7aa-11e5-9fcb-1730471db09c
```

```
# neutron router-update \
   93adda04-af80-11e5-a1de-3f0ca86c2c1c \
   --routes type=dict list=true \
   destination=10.0.1.0/24,nexthop=192.168.0.2
```

Site B

```
# neutron port-create \
   --tenant-id tenant \
   --fixed-ip ip_address=192.168.0.2 \
   --mac-address="6F:E4:5A:FA:8E:09" \
   8943d506-af84-11e5-bb39-e7d07e647dfc
```

Port ID: 5f328a1a-af88-11e5-9a2a-3b0fc560e9e7

```
# neutron router-interface-add \
   e86e6144-af80-11e5-bf25-8f151b61410e \
   port=5f328a1a-af88-11e5-9a2a-3b0fc560e9e7
```

```
# neutron router-update \
   e86e6144-af80-11e5-bf25-8f151b61410e \
   --routes type=dict list=true \
   destination=10.0.0.0/24,nexthop=192.168.0.1
```

In addition, when the router interface port is create in one site, the IP address and the MAC address of this port must be fed into the other site as explained in the beginning of the document.

Site A

```
# neutron port-create \
   --tenant-id tenant \
   --fixed-ip ip_address=192.168.0.2 \
   --mac-address="6F:E4:5A:FA:8E:09" \
   --device-owner "network:remote-site" \
   7d107e3a-a7a3-11e5-986c-7b399d0489a5
```

Port ID: 0bf1e1aa-af8a-11e5-a013-f38f48a84d8b

Site B

```
# neutron port-create \
   --tenant-id tenant \
   --fixed-ip ip_address=192.168.0.1 \
   --mac-address="16:B7:B5:A4:57:75" \
   --device-owner "network:remote-site" \
   8943d506-af84-11e5-bb39-e7d07e647dfc
```

Port ID: 0ca3a674-af8a-11e5-be73-9369ff5103cc

Adding a Remote MAC Entry (Endpoint) to the Multi-site Network

The last step in configuring peering across sites is by adding remote MAC entries in the gateway devices' remote MAC tables. These entries tell the gateway device (and in turn the VTEP router) how to construct the VXLAN header. Each entry represents an end-point in the remote site. In this particular scenario, it is another VTEP router, but it could be a particular port of a VM or a router.

The following adds remote MAC entries in the gateway devices:

Site A

```
# neutron gateway-device-remote-mac-entry-create \
   --vtep-address 200.200.0.2 \
   --mac-address 6F:E4:5A:FA:8E:09 \
   --segmentation-id 100 \
   5f1a1a9a-a7a2-11e5-8bde-8b2506f539f8
```

Remote MAC Entry ID: 32f2f11c-a7f2-11e5-84a0-0242ac110001

Site B

```
# neutron gateway-device-remote-mac-entry-create \
   --vtep-address 200.200.0.1 \
   --mac-address 16:B7:B5:A4:57:75 \
   --segmentation-id 100 \
   8e47afce-af83-11e5-9b76-4b95f3de9711
```

Remote MAC Entry ID: 4a415f8a-af8a-11e5-aa04-1b22bb4b3872

Configure Edge Bindings

At this point, two tenant routers should be peered over VXLAN. To actually get the traffic in and out, there must be an edge router with its ports bound to the physical interfaces. Configuration of these edge bindings is out of this document’s scope, but there are several options to consider:

- Use the existing edge router that is serving north-south (Internet) traffic. In this case, just link the VTEP router to the edge router.
- Use a dedicated edge router to handle VTEP traffic. In this case, create a new edge router, configure its uplinks and connect the VTEP router to this edge router.
- Use the VTEP router as an edge router. This is the most straightforward setup, where you treat the VTEP router as an edge router and configure uplinks on its ports directly.

Deleting a Remote MAC Entry

Site A

```
# neutron gateway-device-remote-mac-entry-delete \
   32f2f11c-a7f2-11e5-84a0-0242ac110001 \
   5f1a1a9a-a7a2-11e5-8bde-8b2506f539f8
```

Site B

```
# neutron gateway-device-remote-mac-entry-delete \
   4a415f8a-af8a-11e5-aa04-1b22bb4b3872 \
   8e47afce-af83-11e5-9b76-4b95f3de9711
```

Deleting an L2 Gateway Connection

The L2 Gateway Connection must be deleted before its associated L2 Gateway is deleted.

Site A

```
# neutron l2-gateway-connection-delete b0aef454-a7a5-11e5-8378-b3df9d563488
```

Site B

```
# neutron l2-gateway-connection-delete acccad4e-af85-11e5-b48e-1b3586917939
```

Deleting an L2 Gateway

The L2 Gateway must be deleted before its associated Gateway Device is deleted.

Site A

```
# neutron l2-gateway-delete ea57e7e8-a7a4-11e5-89c2-b78c120baeb9
```

Site B

```
# neutron l2-gateway-delete 9514751c-af83-11e5-ac3c-affb2bc390ae
```

Deleting a Gateway Device

By deleting the gateway device associated with the VTEP router, the VTEP router goes back to being a regular Neutron router.

Site A

```
# neutron gateway-device-delete 5f1a1a9a-a7a2-11e5-8bde-8b2506f539f8
```

Site B

```
# neutron gateway-device-delete 8e47afce-af83-11e5-9b76-4b95f3de9711
```

Neutron CLI Gateway Device Reference

This section describes the Neutron CLI commands for the Gateway Device API.

List of Gateway Device API commands

```
gateway-device-create
   Create Gateway Device information.

gateway-device-delete
   Delete a given Gateway Device.

gateway-device-list
   List Gateway Devices.

gateway-device-show
   Show information of a given Gateway Device.
```

```
gateway-device-update
   Update a given Gateway Device.

gateway-device-remote-mac-entry-create
   Create Gateway Device Remote Mac Entry information.

gateway-device-remote-mac-entry-delete
   Delete a given Gateway Device Remote Mac Entry.

gateway-device-remote-mac-entry-list
   List Gateway Device Remote Mac Entries.

gateway-device-remote-mac-entry-show
   Show information of a given Gateway Device Remote Mac Entry.
```

neutron gateway-device-create

```
usage: neutron gateway-device-create [-h]
                                     [-f {json,shell,table,value,yaml}]
                                     [-c COLUMN] [--max-width <integer>]
                                     [--noindent] [--prefix PREFIX]
                                     [--request-format {json}]
                                     [--tenant-id TENANT_ID]
                                     [--management-ip MANAGEMENT_IP]
                                     [--management-port MANAGEMENT_PORT]
                                     [--management-protocol
 MANAGEMENT_PROTOCOL]
                                     [--type <hw_vtep | router_vtep>]
                                     [--resource-id RESOURCE_ID] [--name
 NAME]
                                     [--tunnel-ip TUNNEL_IP]
```

Create Gateway Device information.

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
--tenant-id TENANT_ID
                      The owner tenant ID.
--management-ip MANAGEMENT_IP
                      Management IP to the device. Defaults to None.
--management-port MANAGEMENT_PORT
                      Management port to the device. Defaults to None.
--management-protocol MANAGEMENT_PROTOCOL
                      Management protocol to manage the device: ovsdb or
                      none. If management IP and port are specified,
                      defaults to ovsdb. Otherwise to none.
--type <hw_vtep | router_vtep>
                      Type of the device: hw_vtep or router_vtep. Defaults
                      to hw_vtep.
--resource-id RESOURCE_ID
                      Resource UUID or None (for type router_vtep will be
                      router UUID).
--name NAME           User defined device name.
--tunnel-ip TUNNEL_IP
                      IP address on which gateway device originates or
                      terminates tunnel.
```

neutron gateway-device-delete

```
usage: neutron gateway-device-delete [-h]
```

```
                                     [--request-format {json}]
                                     GATEWAY_DEVICE
```

Delete a given gateway-device.

Positional arguments:

```
GATEWAY_DEVICE        ID or name of gateway_device to delete.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
```

neutron gateway-device-show

```
usage: neutron gateway-device-show [-h]
                                   [-f {json,shell,table,value,yaml}]
                                   [-c COLUMN] [--max-width <integer>]
                                   [--noindent] [--prefix PREFIX]
                                   [--request-format {json}] [-D] [-F
 FIELD]
                                   GATEWAY_DEVICE
```

Show information of a given gateway-device.

Positional arguments:

```
GATEWAY_DEVICE        ID or name of Gateway Device to look up.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
-D, --show-details    Show detailed information.
-F FIELD, --field FIELD
                      Specify the field(s) to be returned by server. You
                      can repeat this option.
```

neutron gateway-device-update

```
usage: neutron gateway-device-update [-h]
                                     [--request-format {json}]
                                     [--name NAME] [--tunnel-ip TUNNEL_IP]
                                     GATEWAY_DEVICE
```

Update a given gateway-device.

Positional arguments:

```
GATEWAY_DEVICE        ID or name of Gateway Device to update.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
--name NAME           User defined device name.
--tunnel-ip TUNNEL_IP
                      IP address on which gateway device originates or
                      terminates tunnel.
```

neutron gateway-device-remote-mac-entry-create

```
usage: neutron gateway-device-remote-mac-entry-create [-h]
                                                       [-f {json,shell,
table,value,yaml}]
                                                       [-c COLUMN]
                                                       [--max-width
 <integer>]
                                                       [--noindent]
                                                       [--prefix PREFIX]
                                                       [--request-format
 {json}]
                                                       --mac-address
                                                       MAC_ADDRESS
                                                       --vtep-address
                                                       VTEP_ADDRESS
                                                       --segmentation-id
                                                       SEGMENTATION_ID
                                                       GATEWAY_DEVICE
```

Create Gateway Device Remote Mac Entry information.

Positional arguments:

```
GATEWAY_DEVICE        ID or name of the Gateway Device.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
--mac-address MAC_ADDRESS
                      Remote MAC address.
--vtep-address VTEP_ADDRESS
                      Remote VTEP Tunnel IP.
--segmentation-id SEGMENTATION_ID
                      VNI to be used.
```

neutron gateway-device-remote-mac-entry-delete

```
usage: neutron gateway-device-remote-mac-entry-delete [-h]
                                                       [--request-format
 {json}]
                                                       REMOTE_MAC_ENTRY
                                                       GATEWAY_DEVICE
```

Delete a given Gateway Device Remote MAC Entry.

Positional arguments:

```
REMOTE_MAC_ENTRY      ID of Remote MAC Entry to delete.
GATEWAY_DEVICE        ID or name of the Gateway Device.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
```

neutron gateway-device-remote-mac-entry-list

```
usage: neutron gateway-device-remote-mac-entry-list [-h]
```

```
                                                    [-f {csv,json,table,
value,yaml}]
                                                    [-c COLUMN]
                                                    [--max-width <integer>]
                                                    [--noindent]
                                                    [--quote {all,minimal,
none,nonnumeric}]
                                                    [--request-format
 {json}]
                                                    [-D] [-F FIELD] [-P
 SIZE]
                                                    [--sort-key FIELD]
                                                    [--sort-dir {asc,desc}]
                                                    GATEWAY_DEVICE
```

List Gateway Device Remote Mac Entries.

Positional arguments:

```
GATEWAY_DEVICE        ID or name of the Gateway Device.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
-D, --show-details    Show detailed information.
-F FIELD, --field FIELD
                      Specify the field(s) to be returned by server. You
 can
                      repeat this option.
-P SIZE, --page-size SIZE
                      Specify retrieve unit of each request, then split one
                      request to several requests.
--sort-key FIELD      Sorts the list by the specified fields in the
                      specified directions. You can repeat this option, but
                      you must specify an equal number of sort_dir and
                      sort_key values. Extra sort_dir options are ignored.
                      Missing sort_dir options use the default asc value.
--sort-dir {asc,desc}
                      Sorts the list in the specified direction. You can
                      repeat this option.
```

neutron gateway-device-remote-mac-entry-show

```
usage: neutron gateway-device-remote-mac-entry-show [-h]
                                                    [-f {json,shell,table,
value,yaml}]
                                                    [-c COLUMN]
                                                    [--max-width <integer>]
                                                    [--noindent]
                                                    [--prefix PREFIX]
                                                    [--request-format
 {json}]
                                                    [-D] [-F FIELD]
                                                    REMOTE_MAC_ENTRY
                                                    GATEWAY_DEVICE
```

Show information of a given Gateway Device Remote MAC Entry.

Positional arguments:

```
REMOTE_MAC_ENTRY      ID of Remote MAC Entry to look up.
GATEWAY_DEVICE        ID or name of the Gateway Device.
```

Optional arguments:

```
-h, --help            Show this help message and exit.
--request-format {json}
                      DEPRECATED! Only JSON request format is supported.
-D, --show-details    Show detailed information.
-F FIELD, --field FIELD
                      Specify the field(s) to be returned by server. You
                      can repeat this option.
```

20. FWaaS Logging

Table of Contents

Midonet's implementation of Neutron FWaaS Logging offers a single method of configuring firewalls to log events. FWaaS Logging can be configured by either cloud operator or tenants. The log files are generated on the physical hosts, so only the cloud operators can access them. It is expected that an external service collects and aggregates the generated logs. FWaaS Logging lets you audit and debug firewall rules that you manage.

There are two resources defined for FWaaS Logging.

- Logging Resource - Top level model that represents a collection of firewall logging objects.
- Firewall Logging - Object that represents logging of a particular firewall object.

The flow of operations is as follows:

1. Create a router that will be associated with the firewall for which events will be logged
2. Create a firewall for which events will be logged
3. Create a logging resource
4. Create a firewall log associated with the firewall created in step 2, and the logging resource in step 3.

Multiple firewall logs can be associated with a single logging resource. The logging resource can then be enabled or disabled, providing the ability to enable or disable event logging for multiple firewall logs in a single command.

The following is a concrete example of the topology creation.

Creating a router and firewall

```
$ neutron router-create example_router
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| description           |                                      |
| external_gateway_info |                                      |
| id                    | d7acbd7b-8a1b-44ff-a92f-291fac96b4ca |
| name                  | example_router                       |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | admin                                |
```

```
+------------------------+--------------------------------------+
```

```
$ neutron firewall-policy-create example_policy
Created a new firewall_policy:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| audited        | False                                |
| description    |                                      |
| firewall_rules |                                      |
| id             | 52c3a819-4846-4a8b-8418-453abf965210 |
| name           | example_policy                       |
| shared         | False                                |
| tenant_id      | admin                                |
+----------------+--------------------------------------+
```

```
$ neutron firewall-create example_policy --router d7acbd7b-8a1b-44ff-
a92f-291fac96b4ca
Created a new firewall:
+--------------------+--------------------------------------+
| Field              | Value                                |
+--------------------+--------------------------------------+
| admin_state_up     | True                                 |
| description        |                                      |
| firewall_policy_id | 52c3a819-4846-4a8b-8418-453abf965210 |
| id                 | 2b3cd306-df9b-47c8-adef-ab8381e42bb2 |
| name               |                                      |
| router_ids         | d7acbd7b-8a1b-44ff-a92f-291fac96b4ca |
| status             | PENDING_CREATE                       |
| tenant_id          | admin                                |
+--------------------+--------------------------------------+
```

```
$ neutron firewall-rule-create --protocol tcp --destination-port 80 --
action deny
Created a new firewall_rule:
+------------------------+--------------------------------------+
| Field                  | Value                                |
+------------------------+--------------------------------------+
| action                 | deny                                 |
| description            |                                      |
| destination_ip_address |                                      |
| destination_port       | 80                                   |
| enabled                | True                                 |
| firewall_policy_id     |                                      |
| id                     | 386fa9fd-eb39-41c6-aadc-f1eaef1714cf |
| ip_version             | 4                                    |
| name                   |                                      |
| position               |                                      |
| protocol               | tcp                                  |
| shared                 | False                                |
| source_ip_address      |                                      |
| source_port            |                                      |
| tenant_id              | admin                                |
+------------------------+--------------------------------------+
```

```
$ neutron firewall-policy-insert-rule example_policy 386fa9fd-eb39-41c6-
aadc-f1eaef1714cf
Inserted firewall rule in firewall policy example_policy
```

Creating a logging resource and firewall log

```
$ neutron logging-create example_lr
Created a new logging_resource:
+---------------+---------------------------------------+
| Field         | Value                                 |
```

```
+---------------+--------------------------------------+
| description   |                                      |
| enabled       | False                                |
| firewall_logs |                                      |
| id            | 09e17388-479c-40ea-a124-a4fa53935322 |
| name          | example_lr                           |
| tenant_id     | admin                                |
+---------------+--------------------------------------+
```

```
$ neutron logging-firewall-create --firewall-id 2b3cd306-df9b-47c8-adef-
ab8381e42bb2 example_lr
Created a new firewall_log:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| description         |                                      |
| firewall_id         | 2b3cd306-df9b-47c8-adef-ab8381e42bb2 |
| fw_event            | ALL                                  |
| id                  | 36a5440b-dde5-47b6-9cf1-85ee325b4600 |
| logging_resource_id | 09e17388-479c-40ea-a124-a4fa53935322 |
| tenant_id           | admin                                |
+---------------------+--------------------------------------+
```

Updating a logging resource and firewall log

```
$ neutron logging-update --enabled True 09e17388-479c-40ea-a124-
a4fa53935322
Updated logging_resource: 09e17388-479c-40ea-a124-a4fa53935322
```

```
$ logging-firewall-update 2b3cd306-df9b-47c8-adef-ab8381e42bb2
 09e17388-479c-40ea-a124-a4fa53935322 --fw-event DROP
Updated firewall_log: 2b3cd306-df9b-47c8-adef-ab8381e42bb2
```

Deleting a logging resource and firewall log

```
$ logging-firewall-delete 09e17388-479c-40ea-a124-a4fa53935322
 09e17388-479c-40ea-a124-a4fa53935322
Deleted firewall_log: 09e17388-479c-40ea-a124-a4fa53935322
```

```
$ neutron logging-delete 09e17388-479c-40ea-a124-a4fa53935322
Deleted logging_resource: 09e17388-479c-40ea-a124-a4fa53935322
```

21. Neutron BGP (Dynamic Routing)

Table of Contents

MidoNet's implementation of Neutron BGP provides two different ways of configuring BGP.

Router Insertion

In this mode, when you create a BGP speaker object, you specify the router to which you want to associate it to. This does not match Neutron's Dynamic Advertising Routes for Public Ranges specification, but matches MidoNet's BGP configuration more closely where you configure BGP on a router. This feature is implemented as a Neutron vendor extension, `bgp-speaker-router-insertion`. Router association is specified on creation, and cannot be updated. You can choose to use this extension on BGP Speaker basis.

In this mode, `add_gateway_network` and `delete_gateway_network` APIs are not applicable. Calling these APIs will result in an error.

Non Router Insertion

In this mode, when you create a BGP speaker object, you do not specify a router to associate with. This matches the expected behavior of the reference implementation API. Even in this mode, however, a router must be associated with the BGP speaker implicitly for MidoNet to be able to provide the BGP service. The router association occurs when the `add_gateway_network` API is called on a BGP speaker. In this operation, the specified network is always an external network. If there is an edge router attached to the external network as its gateway router, then this router is chosen to associate with the BGP speaker. The `delete_gateway_network` API is used to disassociate a router from a BGP speaker in this mode. If the specified external network does not have any edge router as its gateway router, an error is thrown.

It is implied in both cases that with MidoNet a BGP speaker is always only associated with one router. Hence, a BGP peer cannot be associated with multiple BGP speakers. While this restriction deviates from the reference implementation of BGP in Neutron, it closely matches the MidoNet implementation of BGP while maintaining compatibility with the Neutron API. Note, however, that several BGP peers can be associated with the same BGP speaker. MidoNet uses routing to decide via which router interface/port to forward packets for a specific BGP session.

Also, with MidoNet, all the networks attached to the router that have BGP configured via router interface command will be advertised automatically. It does not advertise gateway networks, and it does not advertise individual /32 Floating IP routes. Thus, the fields `advertise_floating_ip_host_routes` and `advertise_tenant_networks` of BGP Speaker are ignored. Also, extra routes added on the router are automatically advertised.

Creating a router, network and subnet

In this example, we have an edge router (router acting as the gateway on an external network) but the same workflow applies to regular tenant routers as well.

Skip this step if you already created the Edge Router and external network and subnets. This is here just for completeness.

```
# neutron router-create EdgeRouter
```

Router ID: 14ce01e6-af80-11e5-aca9-6730c321a8c7

```
# neutron net-create \
   --router:external True \
   ExtNetwork
```

Network ID: cb98c61f-b299-4e66-8c29-0da8e4e0146d

```
# neutron subnet-create \
   cb98c61f-b299-4e66-8c29-0da8e4e0146d \
   200.0.0.0/24
```

Subnet ID: d97e2ee8-f194-11e5-949d-0242ac110001

```
# neutron router-interface-add \
   14ce01e6-af80-11e5-aca9-6730c321a8c7 \
   d97e2ee8-f194-11e5-949d-0242ac110001
```

Creating a BGP speaker

Without router specified (bgp-router-insertion):

```
# neutron bgp-speaker-create \
   --tenant-id admin \
   --local-as 4000 \
   --ip-version 4 \
   BGP-Speaker
```

BGP Speaker ID: 5ef786aa-7f83-4762-bbbe-8d0dd51a7704

With router specified:

```
# neutron bgp-speaker-create \
   --tenant-id admin \
   --local-as 4000 \
   --ip-version 4 \
   BGP-Speaker \
   --logical-router 14ce01e6-af80-11e5-aca9-6730c321a8c7
```

BGP Speaker ID: 5ef786aa-7f83-4762-bbbe-8d0dd51a7704

The `advertise_floating_ip_routes` and `advertise_tenant_networks` parameters are ignored.

Updating a BGP Speaker

Only `name` can be updated.

```
# neutron bgp-speaker-update \
   --name BGP-Speaker-New-Name \
   BGP-Speaker
```

Adding a gateway network to a BGP speaker

If the BGP speaker is already associated with a router, an error is thrown. If the external network specified in the request does not have any edge router attached as its gateway router, an error is thrown. Otherwise, the edge router is chosen and associated with the BGP Speaker. Once associated, it cannot be associated to any other BGP speaker.

```
# neutron bgp-speaker-network-add \
   BGP-Speaker \
   ExtNetwork
```

Creating a BGP peer

```
# neutron bgp-peer-create \
   --tenant-id admin \
   --peer-ip 10.0.0.1 \
   --remote-as 4000 \
   --auth-type md5 \
   --password "$1$RzrnCAAt$FWS7sy1ZwWIc0ln7moO0W0" \
   BGP-Peer
```

BGP Peer ID: ba1fc7ca-f197-11e5-8b10-0242ac110001

Supported auth types are `md5` and `none`.

Updating a BGP Peer

`name` and `password` can be updated.

```
# neutron bgp-peer-update \
   --password "$1$b6mXFzDF$KZ0pIqWZQN6DRp4tSUWj.0" \
   BGP-Peer
```

Adding a BGP peer to a BGP Speaker

This operation only succeeds if the BGP speaker is already associated with a router.

```
# neutron bgp-speaker-peer-add \
   BGP-Speaker \
   BGP-Peer
```

Listing advertised routes

```
# neutron bgp-speaker-advertiseroute-list \
   ea57e7e8-a7a4-11e5-89c2-b78c120baeb9
```

Removing a gateway network

This operation only succeeds if BGP Speaker was originally associated with a router via the `add_gateway_network` API.

```
# neutron bgp-speaker-network-remove \
   BGP-Speaker \
   ExtNetwork
```

Removing a BGP peer from a BGP Speaker

```
# neutron bgp-speaker-peer-remove \
   BGP-Speaker \
   BGP-Peer
```

Deleting a BGP Peer

```
# neutron bgp-peer-delete \
   BGP-Peer
```

Deleting a BGP Speaker

```
# neutron bgp-speaker-delete \
   BGP-Speaker
```

22. VPN as a Service (VPNaaS)

Table of Contents

The VPNaaS feature of MidoNet provides secure overlay connectivity between multiple private networks possibly hosted on different sites.

We'll show here how to setup a basic environment for reference purposes.

Create the private networks and routers

The following two networks will be connected through the VPNaaS feature. They may be created in different sites.

Create the left side:

```
# neutron net-create LEFT

Created a new network:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| created_at            | 2016-07-08T08:14:51                  |
| description           |                                      |
| id                    | 1331e08a-8f9c-4d75-b7ff-661a474290d0 |
| name                  | LEFT                                 |
| port_security_enabled | True                                 |
| provider:network_type | midonet                              |
| router:external       | False                                |
| shared                | False                                |
| status                | ACTIVE                               |
| subnets               |                                      |
| tags                  |                                      |
| tenant_id             | admin                                |
| updated_at            | 2016-07-08T08:14:51                  |
+-----------------------+--------------------------------------+
```

```
# neutron subnet-create \
   --name LEFT_SUB \
   1331e08a-8f9c-4d75-b7ff-661a474290d0 \
   10.1.0.0/24 \
   --gateway 10.1.0.1

Created a new subnet:
+-------------------+------------------------------------------+
| Field             | Value                                    |
+-------------------+------------------------------------------+
| allocation_pools  | {"start": "10.1.0.2", "end": "10.1.0.254"} |
| cidr              | 10.1.0.0/24                              |
| created_at        | 2016-07-08T08:15:20                      |
| description       |                                          |
| dns_nameservers   |                                          |
```

```
| enable_dhcp       | True                                     |
| gateway_ip        | 10.1.0.1                                 |
| host_routes       |                                          |
| id                | 538fd8eb-b36f-400e-8bba-995b8f674f21     |
| ip_version        | 4                                        |
| ipv6_address_mode |                                          |
| ipv6_ra_mode      |                                          |
| name              | LEFT_SUB                                 |
| network_id        | 1331e08a-8f9c-4d75-b7ff-661a474290d0     |
| subnetpool_id     |                                          |
| tenant_id         | admin                                    |
| updated_at        | 2016-07-08T08:15:20                      |
+-------------------+------------------------------------------+
```

```
# neutron router-create LEFT_ROUTER

Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| description           |                                      |
| external_gateway_info |                                      |
| id                    | 384ed43a-3297-4fac-9f1a-a29e1749bffa |
| name                  | LEFT_ROUTER                          |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | admin                                |
+-----------------------+--------------------------------------+
```

```
# neutron router-interface-add \
   384ed43a-3297-4fac-9f1a-a29e1749bffa \
   538fd8eb-b36f-400e-8bba-995b8f674f21

Added interface b0c5fbcb-d944-4eaf-83b5-a30039449900
to router 384ed43a-3297-4fac-9f1a-a29e1749bffa.
```

```
# neutron router-gateway-set \
   384ed43a-3297-4fac-9f1a-a29e1749bffa \
   EXTERNAL

Set gateway for router 384ed43a-3297-4fac-9f1a-a29e1749bffa
```

Create the right side:

```
# neutron net-create RIGHT

Created a new network:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| created_at            | 2016-07-11T01:37:37                  |
| description           |                                      |
| id                    | 75b9789e-6e13-4bfe-9668-c38dcdbe7a67 |
| name                  | RIGHT                                |
| port_security_enabled | True                                 |
| provider:network_type | midonet                              |
| router:external       | False                                |
| shared                | False                                |
| status                | ACTIVE                               |
| subnets               |                                      |
| tags                  |                                      |
| tenant_id             | admin                                |
| updated_at            | 2016-07-11T01:37:37                  |
+-----------------------+--------------------------------------+
```

```
# neutron subnet-create \
   --name RIGHT_SUB \
   75b9789e-6e13-4bfe-9668-c38dcdbe7a67 \
   10.2.0.0/24 \
   --gateway 10.2.0.1

Created a new subnet:
+-------------------+--------------------------------------------+
| Field             | Value                                      |
+-------------------+--------------------------------------------+
| allocation_pools  | {"start": "10.2.0.2", "end": "10.2.0.254"} |
| cidr              | 10.2.0.0/24                                |
| created_at        | 2016-07-11T01:38:15                        |
| description       |                                            |
| dns_nameservers   |                                            |
| enable_dhcp       | True                                       |
| gateway_ip        | 10.2.0.1                                   |
| host_routes       |                                            |
| id                | 8058c633-4616-42ec-9838-2d2a8786441d       |
| ip_version        | 4                                          |
| ipv6_address_mode |                                            |
| ipv6_ra_mode      |                                            |
| name              | RIGHT_SUB                                  |
| network_id        | 75b9789e-6e13-4bfe-9668-c38dcdbe7a67       |
| subnetpool_id     |                                            |
| tenant_id         | admin                                      |
| updated_at        | 2016-07-11T01:38:15                        |
+-------------------+--------------------------------------------+
```

```
# neutron router-create RIGHT_ROUTER

Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| description           |                                      |
| external_gateway_info |                                      |
| id                    | 24011587-0b8c-484b-9e35-f6779aa27b98 |
| name                  | RIGHT_ROUTER                         |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | admin                                |
+-----------------------+--------------------------------------+
```

```
# neutron router-interface-add \
   24011587-0b8c-484b-9e35-f6779aa27b98 \
   8058c633-4616-42ec-9838-2d2a8786441d

Added interface 5dbf6189-a24b-415e-a934-bbc3f4761b8b
to router 24011587-0b8c-484b-9e35-f6779aa27b98.
```

```
# neutron router-gateway-set \
   24011587-0b8c-484b-9e35-f6779aa27b98 \
   EXTERNAL

Set gateway for router 24011587-0b8c-484b-9e35-f6779aa27b98
```

Create the VPN Policies

```
# neutron vpn-ikepolicy-create IKEPOLICY

Created a new ikepolicy:
+-------------------------+--------------------------------------+
| Field                   | Value                                |
```

```
+-------------------------+--------------------------------------+
| auth_algorithm          | sha1                                 |
| description             |                                      |
| encryption_algorithm    | aes-128                              |
| id                      | 4d1bdf17-4c1a-48c9-a880-9d7a31356ab3 |
| ike_version             | v1                                   |
| lifetime                | {"units": "seconds", "value": 3600}  |
| name                    | IKEPOLICY                            |
| pfs                     | group5                               |
| phase1_negotiation_mode | main                                 |
| tenant_id               | admin                                |
+-------------------------+--------------------------------------+
```

```
# neutron vpn-ipsecpolicy-create IPSECPOLICY

Created a new ipsecpolicy:
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| auth_algorithm       | sha1                                 |
| description          |                                      |
| encapsulation_mode   | tunnel                               |
| encryption_algorithm | aes-128                              |
| id                   | 7607ac27-8708-451c-9df7-d913ec99c11a |
| lifetime             | {"units": "seconds", "value": 3600}  |
| name                 | IPSECPOLICY                          |
| pfs                  | group5                               |
| tenant_id            | admin                                |
| transform_protocol   | esp                                  |
+----------------------+--------------------------------------+
```

Create the VPN Services and Connections

For the left side:

```
# neutron vpn-service-create \
   --name LEFT_CONN \
   384ed43a-3297-4fac-9f1a-a29e1749bffa \
   538fd8eb-b36f-400e-8bba-995b8f674f21

Created a new vpnservice:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| admin_state_up | True                                 |
| description    |                                      |
| external_v4_ip | 200.200.200.2                        |
| external_v6_ip |                                      |
| id             | 54b02ff5-698c-421e-807b-b1fd9ee69e45 |
| name           | LEFT_CONN                            |
| router_id      | 384ed43a-3297-4fac-9f1a-a29e1749bffa |
| status         | PENDING_CREATE                       |
| subnet_id      | 538fd8eb-b36f-400e-8bba-995b8f674f21 |
| tenant_id      | admin                                |
+----------------+--------------------------------------+
```

```
> neutron ipsec-site-connection-create \
   --name LEFT_SITE_CONN \
   --vpnservice-id 54b02ff5-698c-421e-807b-b1fd9ee69e45 \
   --ikepolicy-id 4d1bdf17-4c1a-48c9-a880-9d7a31356ab3 \
   --ipsecpolicy-id 7607ac27-8708-451c-9df7-d913ec99c11a \
   --peer-address 200.200.200.3 \
   --peer-id 200.200.200.3 \
   --peer-cidr 10.2.0.0/24 \
```

```
   --psk secret

Created a new ipsec_site_connection:
+-------------------+----------------------------------------------------+
| Field             | Value                                              |
+-------------------+----------------------------------------------------+
| admin_state_up    | True                                               |
| auth_mode         | psk                                                |
| description       |                                                    |
| dpd               | {"action": "hold", "interval": 30, "timeout": 120} |
| id                | 52684388-c74e-4c06-bf37-a16a045e6ecc               |
| ikepolicy_id      | 4d1bdf17-4c1a-48c9-a880-9d7a31356ab3               |
| initiator         | bi-directional                                     |
| ipsecpolicy_id    | 7607ac27-8708-451c-9df7-d913ec99c11a               |
| local_ep_group_id |                                                    |
| mtu               | 1500                                               |
| name              | LEFT_SITE_CONN                                     |
| peer_address      | 200.200.200.3                                      |
| peer_cidrs        | 10.2.0.0/24                                        |
| peer_ep_group_id  |                                                    |
| peer_id           | 200.200.200.3                                      |
| psk               | secret                                             |
| route_mode        | static                                             |
| status            | PENDING_CREATE                                     |
| tenant_id         | admin                                              |
| vpnservice_id     | 54b02ff5-698c-421e-807b-b1fd9ee69e45               |
+-------------------+----------------------------------------------------+
```

For the right side:

```
# neutron vpn-service-create \
   --name RIGHT_CONN \
   24011587-0b8c-484b-9e35-f6779aa27b98 \
   8058c633-4616-42ec-9838-2d2a8786441d

Created a new vpnservice:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| admin_state_up | True                                 |
| description    |                                      |
| external_v4_ip | 200.200.200.3                        |
| external_v6_ip |                                      |
| id             | 0784e875-5ad0-4757-8005-e6b00aab9bd3 |
| name           | RIGHT_CONN                           |
| router_id      | 24011587-0b8c-484b-9e35-f6779aa27b98 |
| status         | PENDING_CREATE                       |
| subnet_id      | 8058c633-4616-42ec-9838-2d2a8786441d |
| tenant_id      | admin                                |
+----------------+--------------------------------------+
```

```
# neutron ipsec-site-connection-create \
   --name RIGHT_SITE_CONN \
   --vpnservice-id 0784e875-5ad0-4757-8005-e6b00aab9bd3 \
   --ikepolicy-id 4d1bdf17-4c1a-48c9-a880-9d7a31356ab3 \
   --ipsecpolicy-id 7607ac27-8708-451c-9df7-d913ec99c11a \
   --peer-address 200.200.200.2 \
   --peer-id 200.200.200.2 \
   --peer-cidr 10.1.0.0/24 \
   --psk secret

Created a new ipsec_site_connection:
+-------------------+----------------------------------------------------+
| Field             | Value                                              |
+-------------------+----------------------------------------------------+
```

```
| admin_state_up    | True                                                |
| auth_mode         | psk                                                 |
| description       |                                                     |
| dpd               | {"action": "hold", "interval": 30, "timeout": 120} |
| id                | 53368c20-9f1c-49c3-9f69-2566cd8656bd                |
| ikepolicy_id      | 4d1bdf17-4c1a-48c9-a880-9d7a31356ab3                |
| initiator         | bi-directional                                      |
| ipsecpolicy_id    | 7607ac27-8708-451c-9df7-d913ec99c11a                |
| local_ep_group_id |                                                     |
| mtu               | 1500                                                |
| name              | RIGHT_SITE_CONN                                     |
| peer_address      | 200.200.200.2                                       |
| peer_cidrs        | 10.1.0.0/24                                         |
| peer_ep_group_id  |                                                     |
| peer_id           | 200.200.200.2                                       |
| psk               | secret                                              |
| route_mode        | static                                              |
| status            | PENDING_CREATE                                      |
| tenant_id         | admin                                               |
| vpnservice_id     | 0784e875-5ad0-4757-8005-e6b00aab9bd3                |
+-------------------+-----------------------------------------------------+
```

23. MidoNet - Quality of Service (QoS)

Table of Contents

Quality of Service (QoS) is used to provide differentiated services (DS) on network traffic. This will allow policies to be created to filter and govern network traffic flow. When applied to a virtual network environment, this filtering and policing takes place at the *Network Port* level, specifically the port to which the virtual machine or container is connected to the virtual network. This effectively policies all traffic exiting the VM and entering the virtual network.

QoS is configured with standard Neutron operations via the MidoNet Neutron plugin or directly via the MidoNet CLI. Note that using the MidoNet CLI will bypass the Neutron operations and end up with policies and rules that are not tracked in the Neutron database. For this reason, it is highly suggested to use the standard Neutron QoS commands when the virtual environment is based on OpenStack (Chapter 25, "Neutron - Quality of Service (QoS)" [166]), and MidoNet CLI (this document) when OpenStack is not present.

Creating Quality of Service Policies

In order to use QoS policies, a policy must be created, rules added to it, and then that policy must be applied to a *Port* (or to all ports on a *Network*).

Creating a QoS Policy object

If the MidoNet CLI is used to create a QoS policy directly, use the command:

```
midonet> qos-policy create name <name> description <desc> shared <true|
false>
```

The parameters have the same meaning as in the Neutron command. This will create a QoS policy in the MidoNet system (bypassing Neutron, so this should only be used when OpenStack is not present, or in advanced use cases only).

Creating QoS Policy Rules for a Policy

Even though a QoS policy is created, it's just an empty box with no rules set until some rules are created on the policy. Rules must be created directly on a specific policy, and the rule *type* must be decided ahead of time. Rules cannot be created standalone nor can they be shared between policies. Rules are only applicable to the policy on which they were created.

Bandwidth Limit Rules

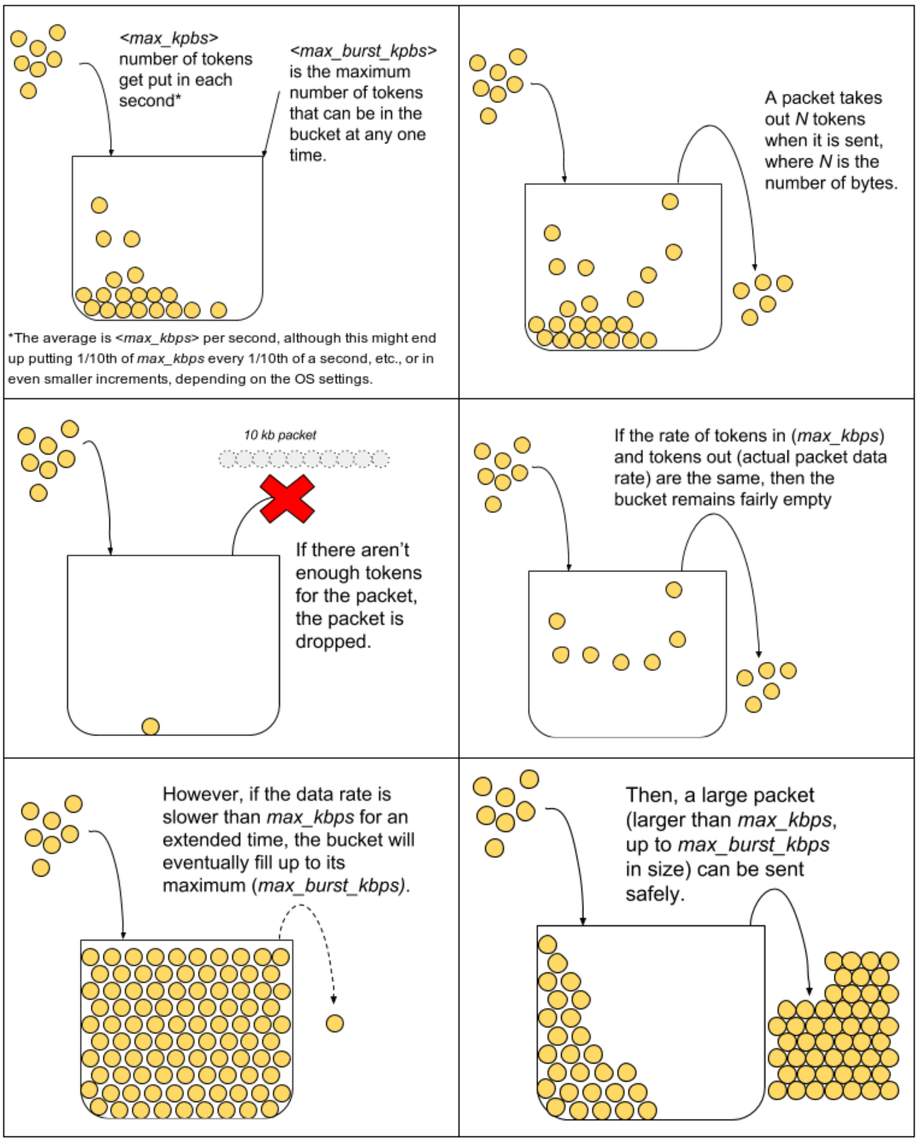
A Bandwidth Limit type rule will limit data throughput on a port. There are two meters that can be applied via a bandwidth limit rule: maximum data rate over a long period of time (referred to as "*max_kbps*"), and maximum data burst allowed with a very short window (referred to as "*max_burst_kbps*", which unfortunately implies a data rate, when in practice, this is a maximum limit on data sent at a particular time).

Note

How Linux Meters Traffic

The best way to contextualize this is to actually explain briefly how the mechanics of how Linux implements data rate limiting. Basically, picture a bucket with tokens. The *max_kbps* represents how many tokens put into the bucket each second (e.g. 1,000 kbps means 1,000 tokens are put into the bucket every second). The *max_burst_kbps* represents the maximum number of tokens allowed in the bucket. For every byte of data that passes through the port, a token is removed from the bucket. If there are not enough tokens in the bucket for the packet to take, the packet is dropped.

This means that the tokens can fill up over time up to a maximum of *max_burst_kbps*, which can then allow a large packet to pass through, however if the traffic is constant at *max_kbps*, this means tokens are being removed from the bucket as fast as they are being added, and larger packets will be dropped as there will never be enough tokens available.



To add a rule directly, use the command:

```
midonet> qos-policy <policy_id> bw-limit-rule create max-kbps <max_kbps>
 max-burst-kbps <max_burst_kbps>
```

Where the parameters are the same as in the Neutron command, except that *policy ID* is the UUID or ID specifier of the policy to be updated.

An ID specifier is the short ID given for a policy in the `qos-policy list` command inside the CLI. It is usually of the form `qos-policyN` (where N is an increasing number from 0). Other objects also have ID specifiers when using the interactive CLI. *Note that the ID specifier is neither returned nor can it be used when using* `midonet-cli -e` *option to run a single command from the shell. In this case, use the UUID instead.*

DSCP Marking Rules

A QoS policy can have rules added to alter the IPv4 ToS field (now used for DSCP) of every packet that enters the cloud via a port. This will affect all IPv4 packets coming into the cloud on the port with the QoS policy set. Packets exiting the cloud on the port will not be affected. At this time, the DSCP marking rules will affect all packets, meaning the last DSCP marking rule will replace the ToS field with its marking value, overwriting any previous values set (even by other DSCP marking rules). This is to say that it really only makes sense to have a single DSCP marking rule on any QoS policy.

Creating a DSCP Marking Rule in MidoNet

To add a rule directly, use the command:

```
midonet> qos-policy <policy_id> dscp-rule create dscp-mark <dscp-mark>
```

Where the parameters are the same as in the Neutron command, except that *policy ID* is the UUID or ID specifier of the policy to be updated.

Applying QoS Policies

QoS rules are not actually put into effect unless they are attached to a port on a virtual network that has ingress traffic from a VM or container. Once attached to a port, the policy will regulate traffic on that port.

There are two ways to apply the effects of a QoS policy: either apply it directly to a specific port, or apply it to an entire network, which will affect all ports on that network.

Important

When a QoS policy is specified on both a port AND that port's network, the specific port's policy will apply, overriding the network setting.

If a port does not have a specific policy, or if it's specific policy is removed, traffic will then conform to the QoS policy on the network, if present. In this way, a default can be set for an entire network, with specific ports overriding the policy to their own individual needs.

If a QoS policy which is attached to a port or network has its rule set updated (either a rule is deleted or a rule is created), that change will take effect on that network or port immediately upon the update. A separate update of the port or network object is not necessary.

Applying a QoS Policy to a Single Port

Policies are applied to a port at the QoS policy object-level. The individual rules on that policy object will be applied. It is not possible to only apply a subset of rules on a policy to a port. If a specific port needs special handling where only a subset of rules should be applied, then an entire new QoS policy object must be created with only the desired subset of rules added to it, and that new policy can then be applied to the special port.

Note that, unlike rules, the QoS policy object **can** be set on many ports or networks, in which case the policy's rules will apply in the same way to each port with the policy.

To create a QoS policy directly, use the command:

```
midonet> port <port_id> set qos-policy <policy-id>
```

The parameters are the same as in the neutron command, except that *port ID* is the UUID or ID specifier of the port to be updated.

Likewise, ports can be created with the QoS policy set:

```
midonet> bridge <bridge_id> port create <...other params...> qos-policy
 <policy-id>
```

Where *bridge ID* is the UUID or ID specifier of the MidoNet bridge (i.e. network) and *policy ID* is the UUID or ID specifier of the QoS policy object. Other parameters for port-creation can be specified in this command as well.

Applying a QoS Policy to All Ports on a Network

QoS Policies can also be applied to an entire network, in which case all VM/container ports on that network will have to rules applied. Again, if a particular port ever has a separate QoS policy assigned to it (via the above command, for example), that port's specific policy will take effect instead of the network's policy. The QoS policy that will be applied will be determined at any time by: a) If the port has a policy, use that, otherwise b) if the network has a policy, use that, otherwise c) use no QoS policy. Any updates to either the port's or network's QoS policy settings will take effect immediately.

To create a QoS policy directly, use the command:

```
midonet> bridge <bridge_id> set qos-policy <policy-id>
```

The *bridge ID* is the UUID or ID specifier of the MidoNet bridge (i.e. network) to update and the *policy ID* is the UUID or ID specifier of the QoS policy object.

Likewise, bridges can be created with the QoS policy set:

```
midonet> bridge create <...other params...> qos-policy <policy-id>
```

Where *policy ID* is the UUID or ID specifier of the QoS policy object. Other parameters for bridge-creation can be specified in this command as well.

Listing Active Policies and their Rules

QoS policies and their rules can be listed, displaying their parameters.

Listing All Policies

These commands will list all policy objects.

```
midonet> qos-policy list
```

This will list all policies. To list rules on a MidoNet QoS policy object, see the next section.

Listing Rules on a Policy

These commands will list all rules for a given policy.

```
midonet> qos-policy <policy_id> bw-limit-rules list
```

```
midonet> qos-policy <policy_id> dscp-rules list
```

Where *policy ID* is a UUID or ID specifier of the policy object to list the rules from. This will list all Bandwidth Limit or DSCP Marking rules on the given policy.

Listing the Policy Active for a Port or Network

These commands show the configured QoS policy for a port or network.

```
midonet> port <port_id> qos-policy show
```

Where *port ID* is the UUID or ID specifier for the port to show the configured QoS policy. This will show the QoS policy information configured for the bridge. If no QoS policy is set on the port, this will just print out the port's information.

```
midonet> bridge <bridge_id> qos-policy show
```

Where *bridge ID* is the UUID or ID specifier for the MidoNet bridge (i.e. network) to show the configured QoS policy. This will show the QoS policy information configured for the bridge.

Updating Policies

QoS policies can be updated to change the policy parameters and the parameters of the rules on the policy. A policy must have been created to update it, but it does not have to have active rules or be attached to any port or network in order to update it. To update rules on a policy, the rule must be updated directly, which means bulk updates of rules are not supported.

Updating General Policy Information

General policy information includes the name of the policy, it's description and whether or not it' shared across tenants.

To update a policy, use the command:

```
midonet> qos-policy <policy_id> set name <name> description <desc> shared
 <true|false>
```

Where the *policy ID* is the UUID or ID specifier of the QoS policy to update, and the other parameters are the same as the Neutron command. If a parameter is omitted, it will not be updated and will remain unchanged.

Updating Rules on a Policy

QoS policies can also have updates to the set of rules that are set on that policy. This is done through an update to the rule on the policy. Rules can only be updated one-by-one. When a rule's parameters are updated, the change will take effect immediately upon the update.

Updating a Bandwidth Limit Rule

These commands update the Bandwidth Limit rules for a QoS policy.

To update a rule directly, use the command:

```
midonet> qos-policy <policy_id> bw-limit-rule <rule_id> set max-kbps
 <max_kbps> max-burst-kbps <max_burst_kbps>
```

Where the parameters are the same as in the Neutron command, except that *rule ID* is the UUID or ID specifier of the rule to be updated, and *policy ID* is the UUID or ID specifier of the policy the rule belongs to.

Updating a DSCP Marking Rule

These commands update the DSCP marking rules for a QoS policy.

To update a rule directly, use the command:

```
midonet> qos-policy <policy_id> dscp-rule <rule_id> set dscp-mark
 <dscp_mark>
```

Where the parameters are the same as in the Neutron command, except that *rule ID* is the UUID or ID specifier of the rule to be updated, and *policy ID* is the UUID or ID specifier of the policy the rule belongs to.

Changing Rules on a Policy

Changing which rules are active on a QoS policy basically boils down to either creating or deleting rules directly. You cannot affect the rules on a policy by manipulating the policy object itself, it must be done by targeting the individual rule on a specific policy. This also means that bulk changes of rules on a policy (like deleting many rules at a time, clearing all rules, creating many rules at once) are not supported. Please see the section called “Creating QoS Policy Rules for a Policy” [153] and the section called “Deleting a Rule on a Policy” [159].

Deleting QoS Policies

QoS policy objects or individual rules can be deleted. Rules must be deleted individually from a specific QoS policy object. If a QoS policy object is deleted, all rules that belong to that object are also deleted. If a deleted QoS policy is attached to a port or network, that link is cleared and the *qos-policy-id* field of the port or network object is set to empty.

Deleting a Rule on a Policy

A single rule on a QoS policy object can be deleted to remove its effects from the policy. Once a rule is deleted from a policy, traffic through any associated ports or networks will cease to be affected by the rule’s parameters immediately after it is deleted.

To delete a rule directly, use the command:

```
midonet> qos-policy <policy_id> bw-limit-rule <rule_id> delete
```

Or:

```
midonet> qos-policy <policy_id> dscp-rule <rule_id> delete
```

Where *rule ID* is the UUID or ID specifier of the rule to be deleted, and *policy ID* is the UUID or ID specifier of the policy the rule belongs to.

Deleting an Entire Policy

An entire QoS policy, and any attached rules, can be deleted at a single time with the following commands. Once a policy is deleted, it will immediately cease to have any effect on any network traffic. All rules created on the deleted policy will be cleaned up and removed, and all networks and ports attached to this policy will have their *qos-policy-id* fields cleared.

To delete a policy directly, use the command:

```
midonet> qos-policy <policy_id> delete
```

Where *policy ID* is the UUID or ID specifier of the policy to delete.

24. Floating IPv6 to Fixed IPv4

Table of Contents

Starting with release v5.4, MidoNet supports mapping a floating IPv6 address to a host with a fixed IPv4 address. This capability allows IPv6 Internet clients to reach virtual instances in IPv4 tenant networks.

This feature works with virtual network topologies where the edge router is linked to an IPv6 external network from which IPv6 floating IP addresses are assigned to tenant instances. The edge router is also required to have an IPv6 uplink interface.

The feature implements stateful NAT64 as specified by RFC 6146, which is a mechanism for IPv4-IPv6 transition and IPv4-IPv6 coexistence. It allows an IPv6 client from the Internet to initiate communication with an IPv4-only server. MidoNet does not support communication initiated by the IPv4-only hosts through statically configured bindings in the stateful NAT64.

Note

Floating IPv6-IPv4 works alongside floating IPv4, and you can use this dual-stack configuration to support communication with both IPv4 and IPv6 hosts from a tenant network. Similarly, the physical uplink interfaces work simultaneously with IPv4 and IPv6.

Important

Floating IPv6-IPv4 is only supported for virtual topologies where the IPv6 external network is directly connected to the edge router.

Setup

Floating IPv6-IPv4 is configured from Neutron. Use the following commands to create a virtual topology with floating IPv6-IPv4. You can skip certain steps if you already have an existing network topology.

1. Create an edge router.

   ```
   neutron router-create edge-router
   ```

2. Create an IPv6 uplink network and subnet. In principle, these will use a globally routable IPv6 prefix. In this example, we use `2001::/64` as the uplink IPv6 prefix.

   ```
   neutron net-create uplink-network \
     --tenant_id admin \
     --provider:network_type uplink
   ```

   ```
   neutron subnet-create \
     --tenant_id admin \
     --disable-dhcp \
   ```

```
  --ip-version 6 \
  --name uplink-subnet-6 \
  uplink-network 2001::0/64
```

3. Create a port on the uplink network. This port will be bound to the physical uplink interface, <UPLINK>. The IPv6 address assigned to the port will be an address from the uplink IPv6 address, in this example we use `2001::2`. We denote by <UPLINK-PORT-ID> the identifier of the uplink port, and by <HOST-ID> the name of the gateway host.

```
neutron port-create uplink-network \
  --binding:host_id <HOST-ID> \
  --binding:profile type=dict interface_name=<UPLINK> \
  --fixed-ip ip_address=2001::2
```

Note

If you setup the uplink port for a dual-stack IPv4-IPv6 configuration, during this step you can specify multiple fixed IP addresses for the uplink port, one for each subnet created on the uplink network. For example, to setup a dual-stack configuration where the IPv4 uplink subnet is 200.0.0.0/30, create a second IPv4 subnet on the uplink network and then assign both an IPv4 and IPv6 address to the uplink port.

```
neutron subnet-create \
  --tenant_id admin \
  --disable-dhcp --ip-version 4 \
  --name uplink-subnet-4 \
  uplink-network 200.0.0.0/30
```

```
neutron port-create uplink-network \
  --binding:host_id <HOST-ID> \
  --binding:profile type=dict interface_name=<UPLINK> \
  --fixed-ip ip_address=2001::2 \
  --fixed-ip ip_address=200.0.0.2
```

4. Add a router interface from the edge router to the uplink network.

```
neutron router-interface-add edge-router \
  port=<UPLINK-PORT-ID>
```

At the point you should be able to ping the IPv6 address of the uplink port from the uplink network.

5. Add to the edge router one or more static routes to allow the tenants to communicate with other IPv6 networks. For example, to add a default route for the IPv6 Internet, use:

```
neutron router-update \
  --route destination=::/0,nexthop=2001::1 \
  edge-router
```

To add a specific route for an IPv6 network, for instance `3001::/64`, use:

```
neutron router-update \
  --route destination=3001::/64,nexthop=2001::1 \
  edge-router
```

Important

The MidoNet 5.4 BGP implementation does not support IPv6. Therefore, all IPv6 routes must be configured statically on the edge router.

6. Create an external network to interconnect the edge and tenant routers. For floating IPv6-IPv4, this network must have an IPv6 subnet that is also globally routable. In this example, we use `1000::/120` as the public IPv6 prefix.

   ```
   neutron net-create public-network \
     --router:external true
   ```

   ```
   neutron subnet-create \
     --name public-subnet-6 \
     --ip-version 6 \
     public-network 1000::/120
   ```

   ### Note

   If you setup a dual-stack IPv4 and IPv6 external network to allow both IPv4 and IPv6 communication for the tenant network, during this step you can also create an IPv4 subnet on the external network. Subsequently, this will allow you to create both IPv4 and IPv6 floating IPs. Alternatively, you can also create multiple external networks with one subnet, one for each IP version.

7. Add a router interface from the edge router to the external network.

   ```
   neutron router-interface-add edge-router public-subnet-6
   ```

8. Create a tenant router.

   ```
   neutron router-create tenant-router
   ```

9. Set the gateway for the tenant router to the external network.

   ```
   neutron router-gateway-set tenant-router public-network
   ```

   ### Note

   If the external network is configured with dual-stack IPv4 and IPv6 subnets, this step will create a network port with multiple IP addresses, one for each subnet from the external network.

10. Create an IPv4 tenant network. In this example, we use `192.168.0.0/24` for the tenant network.

    ```
    neutron net-create tenant-network
    ```

    ```
    neutron subnet-create \
      --name tenant-subnet \
      tenant-network 192.168.0.0/24
    ```

11. Add a router interface from the tenant router to the tenant network.

    ```
    neutron router-interface-add tenant-router tenant-subnet
    ```

12. Create a floating IP on the external network.

    ```
    neutron floatingip-create public-network
    ```

    ### Important

    If the external network is configured with dual-stack IPv4 and IPv6 subnets, you must specify a subnet in the previous command that indicates the subnet for which the floating IP is created. Otherwise, Neutron will select the first subnet as the default one.

```
neutron floatingip-create --subnet public-subnet-6 public-network
```

13. Associate the floating IP with identifier <FIP-ID> created in the previous step to an instance port <VIF-PORT-ID>.

```
neutron floatingip-associate <FIP-ID> <VIF-PORT-ID>
```

Note

You may also create and associate a floating IP to a port during the same command.

```
neutron floatingip-create \
  --port-id <VIF-PORT-ID> \
  --subnet public-subnet-6 \
  public-network
```

Cleanup

Use the following steps to tear-down a virtual network topology that uses floating IPv6-IPv4.

1. Disassociate any floating IPs associated with virtual ports on the tenant network.

```
neutron floatingip-disassociate <FIP-ID>
```

2. Delete all ports from the tenant network (delete any corresponding compute instances as needed).

3. Delete the router interface from the tenant router to the tenant network.

```
neutron router-interface-delete tenant-router tenant-subnet
```

4. Delete the tenant network

```
neutron subnet-delete tenant-subnet
neutron net-delete tenant-network
```

5. Clear the tenant router gateway.

```
neutron router-gateway-clear tenant-router
```

6. Delete the tenant router.

```
neutron router-delete tenant-router
```

7. Delete the router interfaces from the edge router to the external network.

```
neutron router-interface-delete edge-router public-subnet-6
```

Note

If the external network has multiple subnets, delete the router interfaces for all subnets, as needed.

8. Delete the external network.

```
neutron subnet-delete public-subnet-6
neutron net-delete public-network
```

Note

If the external network has multiple subnets, delete all subnets from the external network, as needed.

9. Delete the static routes from the edge router.

```
neutron router-update --no-routes edge-router
```

10. Delete the router interface from the edge router to the uplink network.

```
neutron router-interface-delete edge-router \
  port=<UPLINK-PORT-ID>
```

11. Delete the uplink network.

```
neutron subnet-delete uplink-subnet-6
neutron net-delete uplink-network
```

Note

If the uplink network has multiple subnets, delete all subnet from the uplink network, as needed.

12. Delete the edge router.

```
neutron router-delete edge-router
```

Stateful NAT64

Floating IPv6 to fixed IPv4 uses NAT64 implemented between the edge and tenant routers, where inbound IPv6 traffic is translated to IPv4 according to IPv6-IPv4 translation mappings stored in an internal Binding Information Base (BIB) [RFC 6146]. These mappings are allocated on demand for new flows on a first-come-first-served basis, such that:

- The destination floating IPv6 address maps to the corresponding IPv4 address from the tenant network.
- The source IPv6 address maps to an allocated IPv4 address from a special IPv4 subnet called a NAT64 pool.

NAT64 Pool

There exists a separate NAT64 IPv4 address pool for each tenant router using the Shared Address Space for Carrier-Grade NAT with the IPv4 address range 100.64.0.0/10, as defined by RFC 6598.

Important

Floating IPv6-IPv4 cannot be used with IPv4 tenant networks that use address ranges overlapping with the NAT64 IPv4 address pool.

The following diagram illustrates the NAT64 address translation for the addresses configured in the previous steps. The IP addresses are as follows:

- The floating IPv6 address associated to an instance from a tenant network is `1000::100`.

- The remote IPv6 address of a host from the IPv6 Internet is `3000:cccc::10`.
- The IPv6 address of the instance from the tenant network is `192.168.0.20`.
- The allocated IPv4 address from the NAT64 pool is `100.64.5.32`.

Translation for traffic inbound to the tenant network.

```
IPv6                     IPv4
Src: 3000:cccc::10  ->   Src: 100.64.5.32
Dst: 1000::100           Dst: 192.168.0.20
```

Translation for traffic outbound from the tenant network.

```
IPv4                     IPv6
Src: 192.168.0.20   ->   Src: 1000::100
Dst: 100.64.5.32         Dst: 3000:cccc::10
```

NAT64 State Sharing

MidoNet supports scalable layer 3 gateways where an edge router may have several physical uplink ports distributed across different physical hosts. This allows ingress and egress traffic for a floating IPv6-IPv4 flow to use different paths and it requires MidoNet running on different gateways to exchange the changes to their Binding Information Bases.

MidoNet uses port groups to determine the gateways that must exchange floating IPv6 flow state for a given set of uplink ports. For the edge routers created with Neutron, all uplink ports are added by default to the same stateful port group ensuring that they exchange flow state.

You can use the MidoNet CLI to view and modify the port group configuration. For more information, see: the section called “Stateful port groups” [18].

25. Neutron - Quality of Service (QoS)

Table of Contents

Quality of Service (QoS) is used to provide differentiated services (DS) on network traffic. This will allow policies to be created to filter and govern network traffic flow. When applied to a virtual network environment, this filtering and policing takes place at the *Network Port* level, specifically the port to which the virtual machine or container is connected to the virtual network. This effectively policies all traffic exiting the VM and entering the virtual network.

QoS is configured with standard Neutron operations via the MidoNet Neutron plugin or directly via the MidoNet CLI. Note that using the MidoNet CLI will bypass the Neutron operations and end up with policies and rules that are not tracked in the Neutron database. For this reason, it is highly suggested to use the standard Neutron QoS commands when the virtual environment is based on OpenStack (this doc), and MidoNet CLI (Chapter 23, "MidoNet - Quality of Service (QoS)" [153]) when OpenStack is not present.

Creating Quality of Service Policies

In order to use QoS policies, a policy must be created, rules added to it, and then that policy must be applied to a *Port* (or to all ports on a *Network*).

Creating a QoS Policy object

QoS policies can be created via the Neutron command:

```
$ neutron qos-policy-create --description <desc> --shared <true|false>
 <name>
```

Where *desc* is an optional description, *shared* can be set to **true** (share this policy among all tenants) or **false** (make this policy available only to this tenant), and *name* will name this policy (the name is the only required field).

Creating QoS Policy Rules for a Policy

Even though a QoS policy is created, it's just an empty box with no rules set until some rules are created on the policy. Rules must be created directly on a specific policy, and the rule *type* must be decided ahead of time. Rules cannot be created standalone nor can they be shared between policies. Rules are only applicable to the policy on which they were created.

Bandwidth Limit Rules

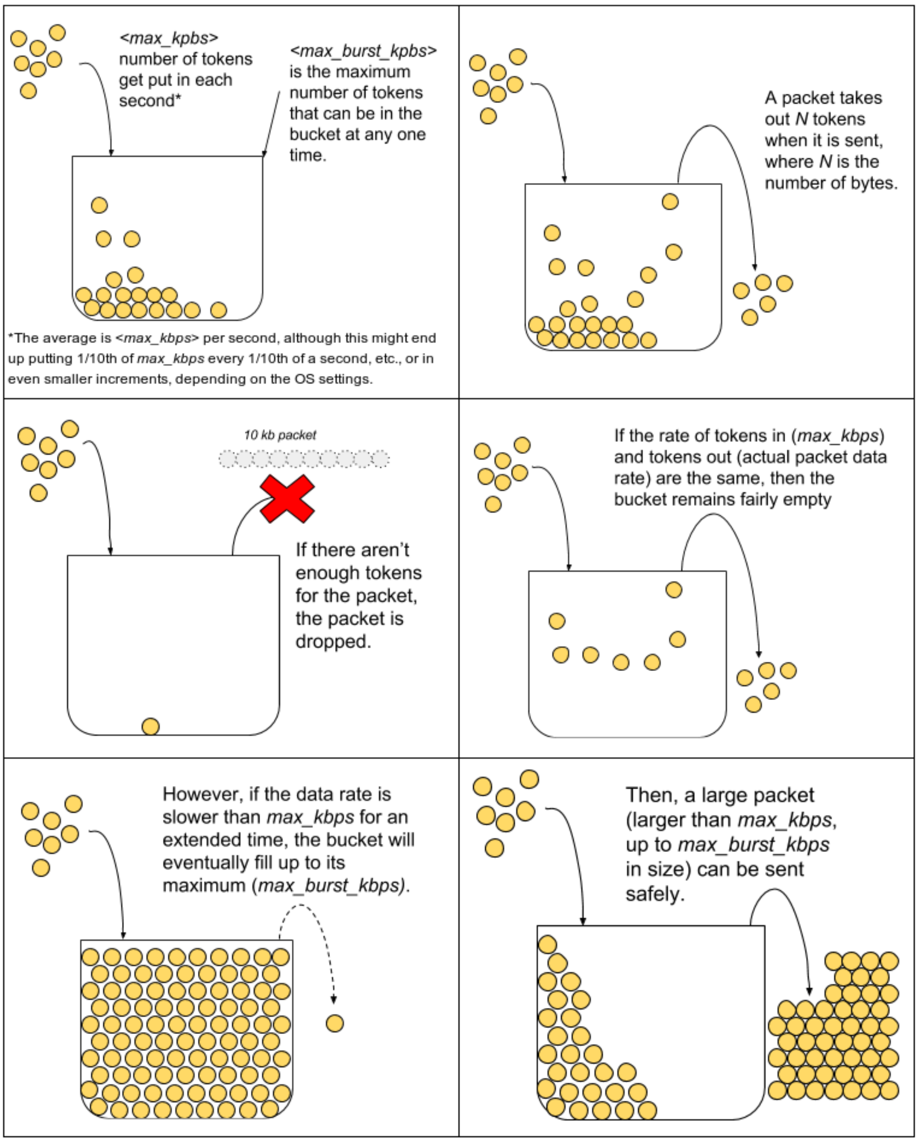
A Bandwidth Limit type rule will limit data throughput on a port. There are two meters that can be applied via a bandwidth limit rule: maximum data rate over a long period of time (referred to as "*max_kbps*"), and maximum data burst allowed with a very short window (referred to as "*max_burst_kbps*", which unfortunately implies a data rate, when in practice, this is a maximum limit on data sent at a particular time).

Note

How Linux Meters Traffic

The best way to contextualize this is to actually explain briefly how the mechanics of how Linux implements data rate limiting. Basically, picture a bucket with tokens. The *max_kbps* represents how many tokens put into the bucket each second (e.g. 1,000 kbps means 1,000 tokens are put into the bucket every second). The *max_burst_kbps* represents the maximum number of tokens allowed in the bucket. For every byte of data that passes through the port, a token is removed from the bucket. If there are not enough tokens in the bucket for the packet to take, the packet is dropped.

This means that the tokens can fill up over time up to a maximum of *max_burst_kbps*, which can then allow a large packet to pass through, however if the traffic is constant at *max_kbps*, this means tokens are being removed from the bucket as fast as they are being added, and larger packets will be dropped as there will never be enough tokens available.



Creating a Bandwidth Limit Rule

Bandwidth limit rules can be created via the Neutron command:

```
$ neutron qos-bandwidth-limit-rule-create --max-kbps <max_kbps> --max-
burst-kbps <max_burst_kbps> <policy_id>
```

Where *policy ID* is the UUID of the policy to add the rule to, *max_kbps* is the maximum sustained data rate allowed (rate to fill the token bucket in the example above), and *max_burst_kbps* is the maximum packet size allowed (maximum tokens allowed in the token bucket in the example above).

DSCP Marking Rules

A QoS policy can have rules added to alter the IPv4 ToS field (now used for DSCP) of every packet that enters the cloud via a port. This will affect all IPv4 packets coming into the cloud on the port with the QoS policy set. Packets exiting the cloud on the port will not be affected. At this time, the DSCP marking rules will affect all packets, meaning the last DSCP marking rule will replace the ToS field with its marking value, overwriting any previous values set (even by other DSCP marking rules). This is to say that it really only makes sense to have a single DSCP marking rule on any QoS policy.

Creating a DSCP Marking Rule

DSCP marking rules can be created via the Neutron command:

```
$ neutron qos-dscp-marking-rule-create --dscp-mark <dscp_mark>
```

Where *policy ID* is the UUID of the policy to add the rule to and *dscp_mark* is the value that will be written to each IPv4 packet's ToS field.

Applying QoS Policies

QoS rules are not actually put into effect unless they are attached to a port on a virtual network that has ingress traffic from a VM or container. Once attached to a port, the policy will regulate traffic on that port.

There are two ways to apply the effects of a QoS policy: either apply it directly to a specific port, or apply it to an entire network, which will affect all ports on that network.

> **Important**
>
> When a QoS policy is specified on both a port AND that port's network, the specific port's policy will apply, overriding the network setting.

If a port does not have a specific policy, or if it's specific policy is removed, traffic will then conform to the QoS policy on the network, if present. In this way, a default can be set for an entire network, with specific ports overriding the policy to their own individual needs.

If a QoS policy which is attached to a port or network has its rule set updated (either a rule is deleted or a rule is created), that change will take effect on that network or port immediately upon the update. A separate update of the port or network object is not necessary.

Applying a QoS Policy to a Single Port

Policies are applied to a port at the QoS policy object-level. The individual rules on that policy object will be applied. It is not possible to only apply a subset of rules on a policy to a port. If a specific port needs special handling where only a subset of rules should be applied, then an entire new QoS policy object must be created with only the desired subset of rules added to it, and that new policy can then be applied to the special port.

Note that, unlike rules, the QoS policy object **can** be set on many ports or networks, in which case the policy's rules will apply in the same way to each port with the policy.

A port can be updated to have a QoS policy via the Neutron command:

```
$ neutron port-update --qos-policy <policy_id> <port_id>
```

The *policy ID* is the UUID of the created QoS policy object, and the port ID is the UUID of the port to be updated.

Ports can also be created with the QoS policy set:

```
$ neutron port-create <...other params...> --qos-policy <policy_id>
 <network_id>
```

Where *policy ID* is the UUID of the policy object and network ID is the UUID of the network on which to create the port. Other parameters can be specified along with *qos-policy*.

Applying a QoS Policy to All Ports on a Network

QoS Policies can also be applied to an entire network, in which case all VM/container ports on that network will have to rules applied. Again, if a particular port ever has a separate QoS policy assigned to it (via the above command, for example), that port's specific policy will take effect instead of the network's policy. The QoS policy that will be applied will be determined at any time by: a) If the port has a policy, use that, otherwise b) if the network has a policy, use that, otherwise c) use no QoS policy. Any updates to either the port's or network's QoS policy settings will take effect immediately.

A network can be updated to have a QoS policy via the Neutron command:

```
$ neutron net-update --qos-policy <policy_id> <port_id>
```

The *policy ID* is the UUID of the created QoS policy object, and the *network ID* is the UUID of the network to be updated.

Networks can also be created with the QoS policy set:

```
$ neutron net-create <...other params...> --qos-policy <policy_id> <name>
```

Where *policy ID* is the UUID of the policy object and name is the desired name of the network. Other parameters can be specified along with *qos-policy*.

Listing Active Policies and their Rules

QoS policies and their rules can be listed, displaying their parameters.

Listing All Policies

These commands will list all policy objects.

```
$ neutron qos-policy-list
```

This will list all policies along with their rules.

Listing Rules on a Policy

These commands will list all rules for a given policy.

$ neutron qos-bandwidth-limit-rule-list <policy_id>

Where *policy ID* is the UUID of the policy object to list the rules from.

Listing the Policy Active for a Port or Network

These commands show the configured QoS policy for a port or network.

```
$ neutron port-show <port_id>
```

Where *port ID* is the UUID of the port to show the configured QoS policy. This will display the port information. The UUID of the configured QoS policy will be set under the "qos_policy_id" field. If the field is absent or not set, then no QoS policy is set on the port.

For a network, the process is the same:

```
$ neutron net-show <network_id>
```

Where *network ID* is the UUID of the network to show the configured QoS policy. This will display the network information. The UUID of the configured QoS policy will be set under the "qos_policy_id" field. If the field is absent or not set, then no QoS policy is set on the network.

Updating Policies

QoS policies can be updated to change the policy parameters and the parameters of the rules on the policy. A policy must have been created to update it, but it does not have to have active rules or be attached to any port or network in order to update it. To update rules on a policy, the rule must be updated directly, which means bulk updates of rules are not supported.

Updating General Policy Information

General policy information includes the name of the policy, it’s description and whether or not it' shared across tenants.

To update a policy, simply issue the command:

```
$ neutron qos-policy-update --name <name> --description <desc> --shared
 <true|false> <policy_id>
```

Where *name* is the name of the policy, *desc* is an optional short description, *shared* specifies whether to allow other tenants to access this policy, and *policy ID* is the UUID of the QoS policy object to update. If a parameter is omitted, it will not be updated and will remain unchanged.

Updating Rules on a Policy

QoS policies can also have updates to the set of rules that are set on that policy. This is done through an update to the rule on the policy. Rules can only be updated one-by-one. When a rule’s parameters are updated, the change will take effect immediately upon the update.

Updating a Bandwidth Limit Rule

These commands update the Bandwidth Limit rules for a QoS policy.

```
$ neutron qos-bandwidth-limit-rule-update --max-kbps <max_kbps> --max-
burst-kbps <max_burst_kbps> <rule_id> <policy_id>
```

Where *rule ID* is the UUID of the rule to update, *policy ID* is the UUID of the policy the rule belongs to, *max_kbps* is the maximum sustained data rate allowed (rate to fill the token bucket in the example above), and *max_burst_kbps* is the maximum packet size allowed (maximum tokens allowed in the token bucket in the example above).

Updating a DSCP Marking Rule

These commands update the DSCP marking rules for a QoS policy.

```
$ neutron qos-dscp-marking-rule-update --dscp-mark <dscp_mark>
```

Where *rule ID* is the UUID of the rule to update, *policy ID* is the UUID of the policy the rule belongs to, and *dscp_mark* is the value to set in each IPv4 packet’s ToS field.

Changing Rules on a Policy

Changing which rules are active on a QoS policy basically boils down to either creating or deleting rules directly. You cannot affect the rules on a policy by manipulating the policy object itself, it must be done by targeting the individual rule on a specific policy. This also means that bulk changes of rules on a policy (like deleting many rules at a time, clearing all rules, creating many rules at once) are not supported. Please see the section called “Creating QoS Policy Rules for a Policy” [153] and the section called “Deleting a Rule on a Policy” [159].

Deleting QoS Policies

QoS policy objects or individual rules can be deleted. Rules must be deleted individually from a specific QoS policy object. If a QoS policy object is deleted, all rules that belong to that object are also deleted. If a deleted QoS policy is attached to a port or network, that link is cleared and the *qos-policy-id* field of the port or network object is set to empty.

Deleting a Rule on a Policy

A single rule on a QoS policy object can be deleted to remove its effects from the policy. Once a rule is deleted from a policy, traffic through any associated ports or networks will cease to be affected by the rule’s parameters immediately after it is deleted.

```
$ neutron qos-bandwidth-limit-rule-delete <rule_id> <policy_id>
```

Or:

```
$ neutron qos-dscp-marking-rule-delete <rule_id> <policy_id>
```

Where *rule ID* is the UUID of the rule to delete, *policy ID* is the UUID of the policy the rule belongs to.

Deleting an Entire Policy

An entire QoS policy, and any attached rules, can be deleted at a single time with the following commands. Once a policy is deleted, it will immediately cease to have any effect on any network traffic. All rules created on the deleted policy will be cleaned up and removed, and all networks and ports attached to this policy will have their *qos-policy-id* fields cleared.

```
$ neutron qos-policy-delete <policy_id>
```

Where *policy ID* is the UUID of the policy to delete.

26. State Proxy

Table of Contents

The State Proxy is a feature introduced in MidoNet v5.2.1 that has been designed to improve the scalability and reduce the load that MidoNet Agents impose on ZooKeeper nodes.

In previous versions, a MidoNet Agent subscribed to a router or bridge receives continuous updates for the corresponding control plane tables, such as MAC-port, ARP, and peering tables. With the current control plane design and due to ZooKeeper limitations, agents will read the entire contents of the table for every changed table entry.

The State Proxy adds a new service, running inside the MidoNet Cluster, that will proxy state table data from ZooKeeper to the MidoNet Agents. The service consolidates the state table changes through differential updates in order to significantly reduce the bandwidth needed for the agents. This makes it technically possible to support tens of thousands of ports on a single bridge without degrading performance.

Configuration

The State Proxy feature is enabled by default, where all cluster nodes will run the State Proxy service and the MidoNet Agents (Midolman) will subscribe to the cluster for reading the state tables. The discovery of cluster nodes running the State Proxy service is done automatically using service discovery.

When the State Proxy service becomes unavailable, either because all cluster nodes are down or because the service has been administratively disabled, the MidoNet Agents will fallback to the legacy mode of operation, where they connect and read the state tables directly from ZooKeeper.

The State Proxy configuration can be inspected and customized using `mn-conf(1)`.

Important

After changing the State Proxy configuration, you must restart the MidoNet Agents on the affected controller or compute nodes for the changes to take effect.

Network State Database Configuration

This configuration controls whether the MidoNet Agents (and certain services inside the MidoNet Cluster) will read bridge and router state tables from the State Proxy service instead of ZooKeeper.

Enabling and Disabling

Use the following command to enable or disable the State Proxy deployment-wide:

```
$ echo "state_proxy.enabled : {true | false}" | mn-conf set -t default
```

You may also choose to enable or disable the feature for specific nodes, using the -h switch of the mn-conf(1) command.

The State Proxy is enabled at all NSDB clients by default (true)

Advanced Options

The client features an automatic reconnection mechanism with fallback to ZooKeeper when no MidoNet Cluster nodes running the State Proxy service can be reached after a specified time interval.

When there are no servers available, or when an existing connection to a server is lost, the service enters the *soft* reconnection state. During this phase, the subscriptions to the state tables are maintained while the client tries to contact other proxy servers. To avoid flooding a server with requests, there is a configurable interval between connection attempts.

After a fixed number of unsuccessful retries, the client enters the *hard* reconnection phase during which the State Proxy feature is disabled at the NSDB client and reading the state tables falls-back to ZooKeeper. The client will periodically attempt to reconnect and if successful, the feature is re-enabled.

You can use the following configuration options to tune the client connection behavior during the *soft* and *hard* phases.

Name	Default value	description
state_proxy.soft_reconnect_delay	1 second	The interval between reconnect attempts while in *soft* phase.
state_proxy.max_soft_reconnect_attempts	30	The maximum number of reconnect attempts before deeming the service unavailable.
state_proxy.hard_reconnect_delay	30 seconds	The delay between reconnection attempts when the service is unavailable.

The State Proxy client uses a configurable number of threads to handle updates received from the State Proxy service. You can change the number of allocated threads using the following configuration option:

Name	Default	description
state_proxy.network_threads	2	The number of threads used to handle network events

State Proxy Service Configuration

The State Proxy service runs on the MidoNet Cluster nodes.

Enabling and Disabling

Use the following command to enable or disable the State Proxy service on the cluster.

```
$ echo "cluster.state_proxy.enabled : { true | false }" | mn-conf set -t
 default
```

You can also use the -h switch of the mn-conf(1) command to enable or disable the service on specific cluster nodes.

Important

The State Proxy service at the cluster can be enabled or disabled independently of enabling or disabling the feature on the NSDB clients, mainly the MidoNet Agents. However, if you disable the State Proxy service while keeping the feature enabled at the client, the clients will automatically fallback to using ZooKeeper.

Advanced Options

Name	Default value	Description
state_proxy.cluster.cache_threads	4	Number of threads used to handle updates for the state tables. Increase this value on large deployments if big latencies are experienced.
state_proxy.cluster.notify_batch_size	64	This option specifies the maximum number of state table entry updates that can be sent to subscribed clients in a single notification message.
state_proxy.cluster.initial_subscriber_queue_size	16	Default size for the queue of messages for a subscriber.
state_proxy.cluster.server.address	0.0.0.0	The local-interface IP address.
state_proxy.cluster.server.port	2346	The listening TCP port number.
state_proxy.cluster.server.supervisor_threads	1	Number of threads used to accept inbound connections.
state_proxy.cluster.server.worker_threads	4	The number of threads used to handle client requests.
state_proxy.cluster.server.max_pending_connections	1024	The maximum number of half-opened incoming connections (backlog).
state_proxy.cluster.server.bind_retry_interval	60s	The retry interval if the service fails to bind to a local TCP port because the port is already in use. Set to 0 to disable retries.
state_proxy.cluster.server.channel_timeout	15s	The timeout for a channel operation such as closing a connection.
state_proxy.cluster.server.shutdown_quiet_period	0s	The interval during server shutdown when the server may continue accepting new connections or requests. If new connections or requests are received during the quiet period, the server guarantees to process them and the quiet period starts over postponing the shutdown.
state_proxy.cluster.server.shutdown_timeout	15s	The shutdown timeout interval, which begins after the expiration of the quiet period. The timeout interval allows tasks that are in progress to complete before the server shuts down.

27. Flow State Storage

Table of Contents

Flow state is flow-scoped data that agents need to query when processing a packet to make decisions that depend on other previously seen packets belonging to the same connection.

These previous packets may belong to the same flow (have the same or equivalent flow match) or they may belong to traffic that flows in the opposite direction within the same connection.

We call this stateful packet processing. And we call these pieces of state *state keys*, because not only do they keep state of the flow, but are globally unique across all of MidoNet.

Some examples of such state keys are:

- **NAT mappings.** Which keep record network/transport address translations that a virtual device may perform on a connection.
- **Connection tracking keys.** Which tell MidoNet's virtual devices whether a packet belongs to a "forward flow" (the direction of traffic that initiated a particular connection) or to a "return flow" (the opposite direction).

Because packets belonging to the same connection may be simulated at different hosts, MidoNet Agents need a mechanism to share this data. MidoNet Agents push this state to each other proactively over the underlay network but the details are out of scope of this document. We refer the reader to the development documentation for a more detailed explanation of how this flow state is shared among MidoNet Agents.

Even though this *state keys* are pushed among MidoNet Agents and kept in in-memory volatile tables, there are a couple of corner cases that require these keys to be pushed to durable storage: i) after an agent reboot, in which case the state keys will be lost; and ii) upon a port migration, in which case the new owner will be missing the state keys for connections that were active on that port before the migration occurred.

Starting in MidoNet v5.2.1, these flow state keys are stored on disk locally on each agent instead of storing them to Cassandra, which is now considered optional and not necessary for the normal operation of the MidoNet Agents.

Among other benefits, storing flow state locally simplifies the operation of MidoNet as operators don't need to configure and monitor a separate Cassandra cluster.

Configuration

Storing state keys locally on persistent disk on the agents is enabled by default starting from MidoNet v5.2.1. State keys are stored under the `/var/db/midolman/flow-`

`state` directory by default (see Advanced Configuration Options section for options to change it). Each active port on the MidoNet Agent will have a local file under this directory identified with the UUID of the port.

Each of these files contain the associated flow state keys in binary compressed format. These files are of fixed size according to the configuration and implement a log rotation mechanism by which older keys are overwritten by new keys when the file is full so we don't waste storage space.

The Flow State Storage configuration can be inspected and customized using `mn-conf(1)`. The relevant keys start with `agent.minions.flow_state`.

Important

After changing the Flow State Storage configuration, you must restart the MidoNet Agents on the affected controller or compute nodes for the changes to take effect.

Compute Nodes

The default configuration on the `mn-conf` schema is meant to be used by compute nodes. These nodes are meant to have a high number of ports of small size, meaning that each port attached to a VM generate a rather low rate of flow state. We encourage operators to query those values through the `mn-conf` tool.

Gateway Nodes

However, gateway nodes are meant to have usually only one uplink port from/to which all north-south traffic pass through. This means that the MidoNet Agent need to store all flow state corresponding to a single port (one file) that writes to storage at a higher rate. For that purpose, the `agent-gateway-{medium|large}` `mn-conf` templates MUST be used on those nodes. How to setup and apply a given template to a specific host is out of scope of this document and we refer operators to the `mn-conf(1)`.

Seamless Upgrade from a Previous Version

Prior to v5.2.1, the Agent stored flow state keys on Cassandra, which is no longer mandatory. To support a seamless upgrade from previous versions and for backwards compatibility, the MidoNet Agent read state keys upon reboot or after a port migration from Cassandra and from local storage.

Thus, the upgrade path is the same as the operator would expect in normal circumstances, i.e. install the new package version and reboot the MidoNet Agent. After reboot, the agent will start writing to local disk and stop storing data on Cassandra. After all agents have been upgraded to the new version, it is safe to shut down the Cassandra daemons and repurpose the nodes for other functionalities.

Reverting to Cassandra Storage

Use the following commands to disable the local storage and revert to Cassandra and reboot all MidoNet Agents afterwards:

```
$ echo "agent.minions.flow_state.legacy_push_state : true" | mn-conf set -t
 default
```

```
$ echo "agent.minions.flow_state.local_push_state : false" | mn-conf set -t
 default
```

Advanced Configuration Options

A complete list of the configuration options (under the `agent.minions.flow_state` mn-conf key) available for the service is as follows:

Name	Default value	Description
legacy_push_state	false	Whether the FlowState minion will save incoming flow state messages Cassandra.
local_push_state	true	Whether the FlowState minion will save incoming flow state messages to local storage on the MidoNet Agent.
port	6688	The value of the UDP and TCP ports used to listen for incoming flow state messages from the parent Agent process through the loopback interface, and flow state transfer requests.
connection_timeout	2s	Timeout for TCP requests during flow state exchanges.
block_size	262144	The size in bytes of the compressed block for the flow state storage. Changing this has an impact on the amount of memory used as well as the compression ratio achieved by the snappy algorithm. Setting a higher value implies an increased memory usage and potentially a higher compression ratio. Defaults to 256 KB.
blocks_per_port	512	The number of allowed blocks of compressed flow state per virtual port. By default, it will use up to 128 MB (512 blocks of 256 KB) per virtual port. It does not pre-allocate blocks so disk space will not be used unless it's filled with data. With this space, we can hold around 500k messages at a rate of 4k flows per second for a given port (not considering compression).
expiration_time	120s	How long should we keep flow state stored. Flow state entries older than this period of time will be eligible for removal.
expiration_delay	30s	The delay between consecutive flow state invalidation tasks that remove and clear the blocks that are older than the expiration time. This task only marks the block headers as invalid if they are expired so the overhead is minimum.
clean_unused_files_delay	12h	The delay between consecutive runs of the flow state file cleaner task. This is a background task that looks into the current list of flow state files and removes those not being used (written to or reading from). This is a background housekeeping activity to prevent storage from being used needlessly.
log_directory	flowstate	The name of the flow state log directory. The MidoNet Agent will write to this directory the records of the

Name	Default value	Description
		current flow state associated to the ports bound to this agent. This directory will be created in /var/db/midolman by default. You can change the location of the base directory by specifying it in the MIDO_DB_DIR environment variable.

28. Working with the MidoNet CLI

Table of Contents

You can explore and edit all of MidoNet's virtual topology through the CLI, however, you should use write operations with caution, as they are likely to create inconsistencies between MidoNet's idea of the virtual network and OpenStack's view of it.

Note

When using MidoNet with OpenStack, please be careful not to introduce inconsistencies between OpenStack and MidoNet virtual topologies.

With that warning in mind, there are certain tasks for which the CLI can be particularly useful:

- Creating an Edge Router
- Setting up the cloud's up-link using the Border Gateway Protocol (BGP)
- Upgrading MidoNet
- Registering MidoNet Agents
- Setting up an L2 gateway
- Troubleshooting problems with the MidoNet API. Because the CLI uses the MidoNet API directly, it's the easiest way to make requests to it to verify that it works.
- Viewing and exploring MidoNet's virtual topology and the status and network addresses of all hosts running the MidoNet agent

Using the MidoNet CLI

To use the MidoNet CLI you need to connect to the MidoNet host running it.

1. Using SSH to connect to the host running the MidoNet CLI.

   You must know the IP address of the machine you are connecting to as well as your login credentials, that is your username and password. Example of an SSH command:

   ```
   $ ssh root@192.168.17.5
   root@192.168.17.5's password:
   ```

   You have already provided your username, 'root' as part of the command, and now the server is prompting you for a password. Type in your password to get in.

2. The CLI is documented in a set of man pages. To view the man pages, from the system command line, enter:

   ```
   $ man midonet-cli
   ```

3. To start the midonet-cli, at the system prompt, enter:

```
$ midonet-cli
midonet>
```

The midonet> prompt that gets displayed indicates system readiness to accept MidoNet commands. Type in help and hit Enter to get a list of all available commands. You can also infer proper usage and syntax from context-aware auto-complete and by using the describe command.

29. Advanced configuration and concepts

Table of Contents

This section describes configuration options that aren't essential when deploying MidoNet and provides information for professional services and advanced MidoNet users.

MidoNet Configuration: mn-conf

MidoNet stores its configuration in the same ZooKeeper cluster that it uses to store network topology data.

Configuration can be read and managed via the mn-conf command line tool. Refer to its manpage for its full documentation, and to the next paragraphs for a light introduction.

Bootstrapping

The entire configuration of MidoNet is stored in ZooKeeper, however operators need to point mn-conf itself to the ZooKeeper cluster. The same is true for any MidoNet processes, such as its agent. They will fetch configuration from ZooKeeper but need its address to bootstrap that process.

Bootstrap configuration is read, in order of preference, from these sources:

- The MIDO_ZOOKEEPER_HOSTS and MIDO_ZOOKEEPER_ROOT_KEY environment variables.
- ~/.midonetrc
- /etc/midonet/midonet.conf
- /etc/midolman/midolman.conf (for backwards compatibility).

These files should have .ini format, and look like this:

```
[zookeeper]
zookeeper_hosts = 127.0.0.1:2181
root_key = /midonet/v1
```

Warning

Do not modify the root key unless you know what you are doing.

Configuration sources

mn-conf can store and manage a few different configuration documents:

- Node specific configuration. These contain configuration keys that will apply only to a particular MidoNet node, identified by its UUID.
- Configuration templates. These are chunks of configuration, stored by a user-given name that can be assigned to a set of MidoNet nodes.
- The "default" configuration template. Configuration keys found here will be used by every MidoNet node in the system. But only if the first two sources don't override their values.
- The configuration schemas. These are read only and contain default values and *documentation* strings for every configuration key in MidoNet. The defaults found in these schemas will only apply if they are not overriden by one of the higher priority sources.

A single mn-conf command will dump the schemas, effectively getting the full list of all existing MidoNet configuration keys, their default values and their documentation:

```
$ mn-conf dump -s
```

mn-conf man page

OPTIONS LIST

```
$ mn-conf dump  [-h HOST_ID | -t TEMPLATE_NAME -s] [-H]
$ mn-conf get   [-h HOST_ID | -t TEMPLATE_NAME -s] [-H] CONFIG_KEY
$ mn-conf set   [-h HOST_ID | -t TEMPLATE_NAME] [-c] < CONFIG_FILE
$ mn-conf unset [-h HOST_ID | -t TEMPLATE_NAME] CONFIG_KEY
$ mn-conf template-get [-a | -h HOST_ID]
$ mn-conf template-set -h HOST_ID -t TEMPLATE_NAME
$ mn-conf template-clear -h HOST_ID
$ mn-conf template-list
$ mn-conf hosts
$ mn-conf doc CONFIG_KEY
$ mn-conf import [-h HOST_ID | -t TEMPLATE_NAME] [--all] -f CONFIG_FILE
```

DESCRIPTION

MidoNet stores the configuration for all of its nodes in a central repository. **mn-conf** is a command line tool for the inspection and manipulation of this configuration.

The configuration for a particular MidoNet node is composed of several different sources that can be managed using `mn-conf`. By order of priority, it's composed of the following pieces of configuration:

- *The node-specific configuration* assigned to the node based on its UUID. Options found here apply to no other MidoNet nodes. This is useful while troubleshooting the behaviour of an individual node or when there is reason for a node to have a different value on a specific configuration key.
- *The configuration template assigned to this node*. Configuration templates allow applying sets of configuration values to groups of MidoNet nodes. For instance, one might like to assign all MidoNet agents running on the gateway hosts of a cloud the same configuration template, so they can be sized differently than agents running on hosts dedicated to compute services.

- *The "default" configuration template*. This template is applied, with less priority, to all nodes regardless of whether they are assigned to a different template. This is the best place to store cloud-wide configuration settings. MidoNet does not touch this template, even across upgrades, settings written here by operators remain in place until they are removed or unless they are overridden by a higher-priority configuration source.
- *The configuration schema*. The schema is managed by MidoNet and changes across upgrades. Thus, operators may not edit the schema. It contains the default values for all existing configuration options. Also, many configuration keys are annotated with documentation. This documentation can be found by appending the **_description** suffix to a configuration option.

Note that, at runtime, MidoNet nodes will read legacy configuration files too, and they will take precedence over central repository configuration. Thus MidoNet agent will give priority to any values it finds in `/etc/midolman/midolman.conf` over the configuration sources described above. This cannot be managed with `mn-conf`, because it doesn't know if its running on the same host.

`mn-conf` is capable of importing the non-default values in a legacy configuation file into one of the central repository config sources described above. While the `mn-conf` configuration system is fully backwards compatible with legacy configuration files, the `import` command is a seamless mechanism to transition to the new configuration system. See below, under `import`, for a description of this feature.

CONFIGURATION FORMAT

MidoNet uses the HOCON configuration file format. While it's a super set of JSON, it's is much more compact and forgiving and, among other things, allows comments.

Please refer to the HOCON documentation for a complete reference. But, rest assured that using plain JSON will just work.

Here are a two examples of equivalent pieces of configuration that will be understood by MidoNet:

JSON:

```
foo {
    bar {
        baz : 42
        aString : "value"
        aDuration : "100ms"
    }
}
```

Flat:

```
 foo.bar.baz : 42
 foo.bar.aString : "value"
 foo.bar.aDuration : 100ms
```

Although both examples will yield the same results, the second example shows what configuration keys look like when queried. Either in code or in `mn-conf` commands keys should be referred to using the dotted notation.

SUBCOMMANDS

`dump [-h` HOST_ID `| -t` TEMPLATE_NAME `| -s] [-H]`

Dumps the entire selected configuration.

- `--host` HOST_ID, `-h` HOST_ID: Selects a host id for which to dump configuration. The special value `local` will resolve to the local host id.
- `--template` TEMPLATE, `-t` TEMPLATE: Asks for a dump of a particular configuration template.
- `--schema`, `-s`: Asks for a dump of the configuration schema.
- `--host-only`, `-H`: If `-h` was given, print only the node-specific configuration for the given host. Otherwise `mn-conf` will print the configuration the host would have at runtime, combining the node-specific config with its template, the defaults and the schema.

`get [-h` HOST_ID `| -t` TEMPLATE_NAME `| -s] [-H]` CONFIG_KEY

Print the value assigned to a particular configuration key. See the above, in `dump`, the options for selecting the configuration source(s) to read from.

`set [-h` HOST_ID `| -t` TEMPLATE_NAME`] [-c] <var>`

Reads configuration from STDIN and stores it in the selected configuration source. The input will be merged on top of the existing configuration, unless `--clear` is given, in that case the input will replace the existing configuration. See above, in `dump`, the options for selecting the configuration source to write to.

- `-c`, `--clear`: Clear the configuration before storing the values read from standard input.

`unset [-h` HOST_ID `| -t` TEMPLATE_NAME`]` CONFIG_KEY

Clears the value of a configuration key in a selected configuration source. See above, in `dump`, the options for selecting the configuration source to write to.

`template-get [-a | -h` HOST_ID`]`

Queries the assignments of configuration templates to MidoNet node IDs.

- `-a`, `--all`: Prints all template assignments.
- `-h` HOST_ID, `--host` HOST_ID: Prints the template assignment for the given host id. The special value `local` will resolve to the local host id.

`template-set -h` HOST_ID `-t` TEMPLATE_NAME

Assigns a configuration template to a host id.

`template-clear -h` HOST_ID

Clears the template assignment for a host id

`template-list`

Lists existing templates and the hosts that use them.

`hosts`

Displays information about hosts that have custom configurations or non-default templates assigned to them.

`import [-h` HOST_ID | `-t` TEMPLATE_NAME`] [-a] -f` CONFIG_FILE

Imports values from a legacy configuration file into the centralized configuration repository. The default behavior is to import values that differ from the configuration schema only, but the `-a` option will make `mn-conf` import all values. `import` will automatically handle keys that have changed name, path or type. See above, in `dump`, the options for selecting the configuration source where `mn-conf` will write the imported values.

- `-a`, `--all`: Import all values, instead of just those that differ from the configuration schema.
- `-f` CONFIG_FILE, `--file` CONFIG_FILE: Path to the file were `mn-conf` will find the legacy configuration.

MISCELLANEOUS OPTIONS

- `-h`, `--help`: Print a brief help message.

EXAMPLES

Set a configuration key in the default agent template:

```
$ echo "a.config.key : 42" | mn-conf set -t default
```

Dump the runtime configuration (minus local configuration files) of a MidoNet agent:

```
$ mn-conf dump -h a5ff1460-d00c-11e4-8830-0800200c9a66
```

Create a configuration template and assign it to a particular agent:

```
$ echo "a.config.key : 42" | mn-conf set -t "new_template"
$ mn-conf template-set -h a5ff1460-d00c-11e4-8830-0800200c9a66 -t
 new_template
```

Importing non-default values from a legacy configuration file:

```
$ mn-conf import -t default -f /etc/midolman/midolman.conf

Importing legacy configuration:
agent {
    midolman {
        "bgp_holdtime"="120s"
    }
}
zookeeper {
    "session_gracetime"="30000ms"
}
```

Notice how `mn-conf` will automatically handle keys that have been moved or renamed.

FILES

While all configuration is stored in ZooKeeper and both `mn-conf` and MidoNet processes that will make use of it, they all need to bootstrap their connection to ZooKeeper before they can access configuration.

Both `mn-conf` and MidoNet nodes will read bootstrap configuration from these sources, in order of preference:

- Environment variables. See **ENVIRONMENT**, below.
- `~/.midonetrc`
- `/etc/midonet/midonet.conf`
- `/etc/midolman/midolman.conf`

These files are expected to be in .ini format, and contain the following keys:

```
[zookeeper]
zookeeper_hosts = 127.0.0.1:2181
root_key = /midonet/v1
```

ENVIRONMENT

- `MIDO_ZOOKEEPER_HOSTS`: The ZooKeeper connect string.
- `MIDO_ZOOKEEPER_ROOT_KEY`: Root path in the ZooKeeper server where configuation is stored.

Recommended configurations

This section contains information on recommended configurations that impact MidoNet performance.

Overview of Performance-Impacting Configuration Options

DSCP field in flow state packets

The flow state packets that Agent hosts exchange for flow state replication are marked for QoS using the type of service IP field, which is configured by the `control_packet_tos` MidoNet configuration option.

It defaults defaults to the DSCP (differentiated services) field being set to Expedited Forwarding. These packets should be prioritized using classifiers (ingress qdiscs) at the hosts and also by configuring the underlay network hardware accordingly.

agent.midolman configuration section in mn-conf(1)

simulation_threads - Number of threads to dedicate to packet processing

output_channels - Number of output channels to use for flow creation and packet execution. Each channel gets a dedicated thread.

input_channel_threading - The MidoNet Agent creates dedicated Netlink channels to receive packets from each datapath port. This option tunes the threading strategy for reading on those channels: one_to_many will make the Agent use a single thread for all input channels. one_to_one will make the Agent use one thread for each input channel.

agent.datapath section in mn-conf(1)

global_incoming_burst_capacity - Incoming packets are rate-limited by a Hierarchical Token Bucket (HTB) that gets refilled as packets are processed and exit the system. This setting controls the size in packets of the root bucket in the HTB. It is the number of packets that the daemon will accept in a burst (at a higher rate that it can handle) before it starts dropping packets. It's also the maximum number of in-flight packets in the system. Changes to this value affect latency at high-throughput rates.

tunnel_incoming_burst_capacity - Tunnel ports get their own bucket in the HTB, allowing them to accumulate burst capacity if they are not sending at the max. rate. This setting controls the size in packets of that bucket.

vm_incoming_burst_capacity - Each VM port gets its own bucket in the HTB, allowing the ports to accumulate burst capacity if they are not sending at the max. rate. This setting controls the size in packets of that bucket.

agent.loggers section in mn-conf(1)

root

level - Default process-wide log level. The DEBUG setting is meant for development and troubleshooting only because it severely affects performance and is highly verbose.

midolman-env.sh

MAX_HEAP_SIZE - Total amount of memory to allocate for the JVM.

HEAP_NEWSIZE - The total amount of the JVM memory that will be dedicated to the Eden garbage-collection generation. 75% of the total is a good ballpark figure, as the Agent uses most of the memory for short-lived objects during packet processing.

Recommended Values

Table 29.1. Recommended Configuration Values

File	Section	Option	Compute	Gateway	L4 Gateway
logback.xml	root	level	INFO	INFO	INFO
midolman-env.sh	-	MAX_HEAP_SIZE	2048M	6144M	6144M
midolman-env.sh	-	HEAP_NEWSIZE	1536M	5120M	5120M
mn-conf(1)	[agent.midolman]	simulation_threads	1	4	16
mn-conf(1)	[agent.midolman]	output_channels	1	2	2
mn-conf(1)	[agent.midolman]	input_channel_threading	one_to_many	one_to_many	one_to_many
mn-conf(1)	[agent.datapath]	global_incoming_burst_capacity	128	256	128
mn-conf(1)	[agent.datapath]	tunnel_incoming_burst_capacity	64	128	64
mn-conf(1)	[agent.datapath]	vm_incoming_burst_capacity	16	32	16

MidoNet Agent (Midolman) configuration options

This section covers all configuration options for the MidoNet Agent.

We don't recommend making changes to the default values, except possibly the zookeeper.session_gracetime and agent.datapath.send_buffer_pool_buf_size_kb setting values.

Warning

Do not modify the root key, cluster name, or keyspace unless you know what you are doing.

MidoNet behavior after ZooKeeper cluster failure

Nodes running the MidoNet Agent, Midolman, depend on a live ZooKeeper session to load pieces of a virtual network topology on-demand and watch for updates to those virtual devices.

When ZooKeeper becomes inaccessible, a MidoNet Agent instance will continue operating for as long as there's a chance to recover connectivity while keeping the same ZooKeeper session. The amount of operating time is thus dictated by the session timeout, which you can control by editing the zookeeper session_gracetime setting in mn-conf(1).

Once the session expires, the MidoNet Agent will give up and shut itself down, prompting upstart to re-launch it. If the ZooKeeper connection and session are recovered within the session_gracetime, MidoNet Agent operation will resume uneventfully. The MidoNet Agent will learn of all the updates that happened to the virtual topology while it was disconnected and will update its internal state and flow tables accordingly.

While the MidoNet Agent is running disconnected from ZooKeeper, waiting for the session to come back, traffic will still be processed, but with reduced functionality, as follows:

- The MidoNet Agent will not see updates to the virtual topology, thus packets may be processed with a version of the network topology that's up to session_gracetime too old.
- The MidoNet Agent will be unable to load new pieces of the network topology. Packets that traverse devices that had never been loaded on a particular MidoNet Agent will error out.
- The MidoNet Agent will not be able to perform or see updates to Address Resolution Protocol (ARP) tables and Media Access Control (MAC) learning tables.

As time passes, a disconnected MidoNet Agent will become less and less useful. The trade-offs presented above are key to choosing a sensible session_gracetime value; the default is 30 seconds.

ZooKeeper connectivity is not an issue for the MidoNet API server. The API requests are stateless and will simply fail when there is no ZooKeeper connectivity.

ZooKeeper configuration

You may use the ZooKeeper configuration section in mn-conf(1) to adjust:

- the ZooKeeper session timeout value (in milliseconds). This value determines when the system considers the connection between ZooKeeper and the MidoNet Agent to be interrupted.
- the session grace timeout value (in milliseconds). This value determines the period of time during which the Agent can reconnect to ZooKeeper without causing node outage.
- the root path for MidoNet data.

```
zookeeper {
    zookeeper_hosts = <comma separated IPs>
```

```
    session_timeout = 30000
    root_key = /midonet/v1
    session_gracetime = 30000
}
```

Considerations for network partitions and ZooKeeper failure and maintenance

For the MidoNet Agents that run at compute hosts or L2GW hosts, setting the session gracetime is primarily dictated by tradeoffs between maintaining liveness of the datapath and maintaining consistency relative to the authoritative network topology and security policy.

For L3GW hosts with BGP enabled, the considerations are more difficult. While the MidoNet Agent at the L3GW continues to operate disconnected, it continues to announce routes to its BGP peers. If the disconnection from ZooKeeper is due to a network partition that also separates the L3GW from a majority of compute hosts (rather than ZooKeeper failure/maintenance), then it is practically black-holing the traffic destined to computes that are outside its partition. Hence session gracetime presents a tradeoff between increasing L3GW tolerance to ZooKeeper failure (by increasing gracetime) and reducing duration of traffic blackholing (by reducing gracetime).

MidoNet currently does not have a good solution to this tradeoff. In the future we intend to build a blackhole detection mechanism at the L3GW so that it can shut down its BGP session during gracetime if it determines that it cannot deliver most of the traffic it receives.

Scheduling ZooKeeper maintenance

When it's necessary to take all (or a majority) of ZooKeeper Quorum nodes down for maintenance, the best practice is to increase the session_gracetime to a value higher than the anticipated maintenance. This should be done for all MidoNet Agents, both on computes and gateway hosts. After the maintenance event the session_gracetime should be returned to the usual values, targeted at the desired tradeoffs of consistency, ZooKeeper failure-tolerance and blackhole duration (see discussion above).

Cassandra configuration

You may use the Cassandra configuration section to adjust:

- the database replication factor
- the MidoNet cluster name

```
cassandra {
    servers = <comma separated IPs>
    replication_factor = 1
    cluster = midonet
}
```

Datapath configuration

The agent uses a pool of reusable buffers to send requests to the datapath. You may use the options in the agent.datapath of mn-conf(1) to tune the pool's size and its buffers. One pool is created for each output channel, the settings defined here will apply to each of those pools.

If you notice decreased performance because packet sizes exceed the maximum buffer size, you can increase the value for the buf_size_kb setting. This setting controls the buffer size (in KB). Be aware that the buffer size puts a limit on the packet size that the MidoNet Agent can send. In a network that jumbo frames traverse, adjust the size so one buffer will accommodate a whole frame, plus enough room for the flow's actions.

BGP failover configuration

The default BGP fail-over time is 2-3 minutes. However, you can reduce this time by changing some parameters on both ends of the session: in mn-conf(1) (the MidoNet side) and the remote end BGP peer configuration. The example below shows how to reduce the BGP fail-over time to one minute on the MidoNet side:

```
agent {
    midolman {
        bgp_connect_retry=1
        bgp_holdtime=3
        bgp_keepalive=1
    }
}
```

The settings in mn-conf must match those on the remote end BGP peer configuration. For more information about how to set them, see the section called "BGP failover configuration on a BGP peer" [5].

Advanced MidoNet REST API configuration options

This section describes the configuration options for advanced users. The values for the following keys are modified using the MidoNet configuration tool, `mn-conf`. For more information on the MidoNet configuration, see the section called "MidoNet Configuration: mn-conf" [182].

Configuration options

Table 29.2. Admin Roles

Configuration Key	Default Value	Description
`cluster.rest_api.enabled`	`true`	Specifies whether the REST API service is enabled at the MidoNet cluster instance where the configuration applies.
`cluster.rest_api.http_port`	8181	Specifies the listening port for the HTTP end-point.
`cluster.rest_api.https_port`	8443	Specifies the listening port for the HTTPS end-point.

Specifying the private and public keys for HTTPS

To enable the HTTPS end-point of the MidoNet Cluster REST API service, you must configure a JKS key store containing the private and public key X.509 certificate used for encrypting such connections.

The location of the key store file and the password for the private key are specified as the following Java system properties.

Table 29.3. System Properties for the HTTPS Key Store

Property Name	Default Value	Description
`midonet.keystore_path`	`/etc/midonet-cluster/ssl/midonet.jks`	The name of the key store file.
`midonet.keystore_password`	*none*	The password for the private key entry. If not set, the HTTPS end-point of the REST API will be disabled (default).

To change the previous properties, and enable HTTPS, you can add the corresponding property values to the environmental MidoNet Cluster script file found at `/etc/midonet-cluster/midonet-cluster-env.sh`:

```
JVM_OPTS="$JVM_OPTS -Dmidonet.keystore_path=<key-store-file>"
JVM_OPTS="$JVM_OPTS -Dmidonet.keystore_password=<key-entry-password>"
```

Generating self-signed keys

To generate a self-signed key, you can use the following procedure. Note that you will be prompted for passwords during this process, and need to keep the keystore password for later use.

```
openssl genrsa -des3 -out midonet.key 2048
openssl rsa -in midonet.key -out midonet.key
openssl req -sha256 -new -key midonet.key -out midonet.csr -subj '/CN=
localhost'
openssl x509 -req -days 365 -in midonet.csr -signkey midonet.key -out
 midonet.crt
```

Now we will combine the private key into the cert, because we generated them separately:

```
openssl pkcs12 -inkey midonet.key -in midonet.crt -export -out midonet.
pkcs12
```

And load the certificate into the keystore:

```
keytool -importkeystore -srckeystore midonet.pkcs12 -srcstoretype PKCS12 -
destkeystore midonet.jks
```

Now place the keystore in the default location:

```
mv midonet.jks /etc/midonet-cluster/ssl
```

For more advanced key management, including adding your own certificate to the keystore, please refer to the following documentation:

https://www.eclipse.org/jetty/documentation/current/configuring-ssl.html

Cassandra cache

This section describes details relative to the Cassandra-based implementation of the Cassandra cache used for connection tracking and NAT mappings.

Client library and normal operation

Cassandra operations are now asynchronous, so losing connectivity to Cassandra should not impact simulations.

The loss of connectivity to a majority of nodes creates a risk that connections will break on vport migration or MidoNet restart. But this is a low risk if the operator can avoid vport migrations and MidoNet Agent restarts until Cassandra is returned to normal operation.

To summarize, MidoNet can function with Cassandra down (but vport migrations and agent restarts may break connections) so it should only be tolerated for very brief periods.

Agent behavior on node failure

The MidoNet Agent writes to Cassandra asynchronously after processing a packet that produced stateful connection data, such as NAT mappings or connection tracking keys.

The data written to Cassandra is purely a back up, as MidoNet agents share all stateful connection data directly amongst themselves. The back up is used upon agent reboot (to prevent lost connections across the reboot) and upon port migration. An agent binding a port will fetch the existing connection state from Cassandra for all connections traversing that port.

Should a Cassandra node fail, agents connected to it will fail over to other nodes in the Cassandra cluster. If the host tries to bind a port during that interval it may fail to fetch the state for the port, breaking pre-existing connections on that port. Additionally, state for flows processed during that interval may not be written to Cassandra, making it unavailable if it’s needed later in the process of binding a port. State is refreshed every minute, thus a port migration would only break connections that were established in the minute prior to the port migration and only if the disconnection happened in that interval.

Network State Database connectivity

In order to modify the MidoNet Network State Database, you must configure the following parameters in your web.xml file:

- zookeeper-zookeeper_hosts
- zookeeper-session_timeout
- cassandra-servers

Below are example web.xml snippets.

Note

The rest of the configuration is cloud-controller dependent and is covered in the relevant sections of the documentation.

zookeeper-zookeeper_hosts

Lists the ZooKeeper hosts used to store the MidoNet configuration data. The entries are comma delimited:

```
<context-param>
    <param-name>zookeeper-zookeeper_hosts</param-name>
    <param-value>192.168.1.100:2181,192.168.1.101:2181,192.168.1.102:2181</
param-value>
</context-param>
```

zookeeper-session_timeout

Sets the timeout value (in milliseconds) after which ZooKeeper considers clients to be disconnected from the ZooKeeper server:

```
<context-param>
    <param-name>zookeeper-session_timeout</param-name>
    <param-value>30000</param-value>
</context-param>
```

30. MidoNet and OpenStack TCP/UDP service ports

Table of Contents

This section lists the TCP/UDP ports used by services in MidoNet and OpenStack.

Services on the Controller node

This section lists the TCP/UDP ports used by the services on the Controller node.

Category	Service	Prot ocol	Port	Self	Controller	Compute	Mgmt. PC
OpenStack	glance-api	TCP	9292	x	x	x	x
OpenStack	httpd (Horizon)	TCP	80	x			x
MidoNet	midonet-api	TCP	8181	x	x		x
OpenStack	swift-object-server	TCP	6000	x	x	x	
OpenStack	swift-container-server	TCP	6001	x	x	x	
OpenStack	swift-account-server	TCP	6002	x	x	x	
OpenStack	keystone	TCP	35357	x	x	x	x
OpenStack	neutron-server	TCP	9696	x	x	x	x
OpenStack	nova-novncproxy	TCP	6080	x	x		x
OpenStack	heat-api	TCP	8004	x	x		x
OpenStack	nova-api	TCP	8773	x	x		x
Tomcat	Tomcat shutdown control channel	TCP	8005	x	x		
OpenStack	nova-api	TCP	8774	x	x	x	x
OpenStack	nova-api	TCP	8775	x	x	x	x
OpenStack	glance-registry	TCP	9191	x	x	x	
OpenStack	qpidd	TCP	5672	x	x	x	
OpenStack	keystone	TCP	5000	x	x	x	x
OpenStack	cinder-api	TCP	8776	x	x	x	x
Tomcat	Tomcat management port (not used)	TCP	8009	x	x		
OpenStack	ceilometer-api	TCP	8777	x	x	x	x

Category	Service	Prot ocol	Port	Self	Controller	Compute	Mgmt. PC
OpenStack	mongod (ceilometer)	TCP	27017	x	x	x	
OpenStack	MySQL	TCP	3306	x	x	x	

Services on the Network State Database nodes

This section lists the TCP/UDP ports used by the services on the Network State Database nodes.

Category	Service	Prot ocol	Port	Self	Controller	NSDB	Compute	Comment
MidoNet	ZooKeeper communication	TCP	3888	x		x		
MidoNet	ZooKeeper leader	TCP	2888	x		x		
MidoNet	ZooKeeper/Cassandra	TCP	random	x				ZooKeeper/Cassandra "LISTEN" to a TCP high number port. The port number is randomly selected on each ZooKeeper/Cassandra host.
MidoNet	Cassandra Query Language (CQL) native transport port	TCP	9042					
MidoNet	Cassandra cluster communication	TCP	7000	x		x		
MidoNet	Cassandra cluster communication (Transport Layer Security (TLS) support)	TCP	7001	x		x		
MidoNet	Cassandra JMX	TCP	7199	x				JMX monitoring port If you're using this port to monitor Cassandra health, enable communication from the monitoring server.
MidoNet	ZooKeeper client	TCP	2181	x	x	x	x	
MidoNet	Cassandra clients	TCP	9160	x	x	x	x	

Services on the Compute nodes

This section lists the TCP/UDP ports used by the services on the Compute nodes.

Category	Service	Protocol	Port	Self	Controller	Comment
OpenStack	qemu-kvm vnc	TCP	From 5900 to 5900 + # of VM		x	
MidoNet	midolman	TCP	random	x		midolman "LISTEN"s to a TCP high number port. The port number is randomly selected on each MN Agent host.
OpenStack	libvirtd	TCP	16509	x	x	
MidoNet	midolman	TCP	7200	x		JMX monitoring port If you're using this port to monitor health, enable communication from the monitoring server.
MidoNet	midolman	TCP	9697	x		If enabled, MidoNet Metadata Proxy listens on 169.254.169.254:9697 to accept metadata requests.

Services on the Gateway Nodes

This section lists the TCP/UDP ports used by the services on the Gateway Nodes.

Category	Service	Protocol	Port	Self	Misc.	Comment
MidoNet	midolman	TCP	random	x		midolman LISTEN"s to a TCP high number port. The port number is randomly selected on each MN Agent host.
MidoNet	midolman	TCP	7200	x	x	JMX monitoring port If you're using this port to monitor health, enable communication from the monitoring server.
MidoNet	quagga bgpd control	TCP	2606	x		NetworkNameSpace mbgp[Peer Number]_ns
MidoNet	quagga bgpd bgp	TCP	179		BGP neighbor	NetworkNameSpace mbgp[Peer Number]_ns

31. Backup / Restore

Table of Contents

Note

This documentation does not cover the backup of non MidoNet-related OpenStack components. Please refer to the OpenStack Operations Guide for further information on how to backup and recover OpenStack.

ZooKeeper

Backing up the ZooKeeper data consists of making a copy of the latest snapshot file and the transaction log that contains the transactions from the time the snapshot was taken. If in doubt, a copy of the whole data directory should be taken (as in the following example).

See the ZooKeeper Administrator's Guide for further information about ZooKeeper's snapshot and transaction files.

Backup

Important

To ensure that Neutron's MySQL database and MidoNet's ZooKeeper data are in sync, set the MidoNet API to read-only before starting the backup.

Backup both ZooKeeper data and MySQL database (as described in the section called "MySQL / MariaDB" [203]) at the same time!

1. **Set MidoNet API to READONLY**

   Set the MidoNet API to READONLY to avoid topology changes during the backup process:

   ```
   # midonet-cli
   midonet> set system-state availability READONLY
   ```

2. **Backup ZooKeeper Configuration**

   On all NSDB nodes, make a copy of the ZooKeeper configuration:

   Ubuntu:

   ```
   # cp /etc/zookeeper/conf/zoo.cfg /tmp/zk_backup/config/
   ```

```
# cp /var/lib/zookeeper/data/myid /tmp/zk_backup/config/
```

RHEL:

```
# cp /etc/zookeeper/zoo.cfg /tmp/zk_backup/config/
# cp /var/lib/zookeeper/myid /tmp/zk_backup/config/
```

3. **Backup ZooKeeper Data**

   On all NSDB nodes, make a copy of the ZooKeeper data:

   Ubuntu:

```
# cp /var/lib/zookeeper/version-2/* /tmp/zk_backup/data/
```

   RHEL:

```
# cp /var/lib/zookeeper/data/version-2/* /tmp/zk_backup/data/
```

4. **Create MySQL Backup**

   Create the MySQL backup (as described in the section called “MySQL / MariaDB” [203]) to ensure that ZooKeeper and MySQL data are in sync.

5. **Reset MidoNet API to READWRITE**

   Reset the MidoNet API to READWRITE:

```
# midonet-cli
midonet> set system-state availability READWRITE
```

Restore

1. **Stop ZooKeeper**

   On all NSDB nodes, stop the ZooKeeper service:

```
# service zookeeper stop
```

```
# systemctl stop zookeeper.service
```

2. **Restore ZooKeeper Configuration**

   On all NSDB nodes, restore the ZooKeeper configuration from your backup:

   Ubuntu:

```
# cp /tmp/zk_backup/config/zoo.cfg /etc/zookeeper/conf/zoo.cfg
# cp /tmp/zk_backup/config/myid /var/lib/zookeeper/data/myid
```

   RHEL:

```
# cp /tmp/zk_backup/config/zoo.cfg /etc/zookeeper/zoo.cfg
# cp /tmp/zk_backup/config/myid /var/lib/zookeeper/myid
```

3. **Restore ZooKeeper Data**

   On all NSDB nodes, restore the ZooKeeper data from your backup:

   Ubuntu:

```
# rm /var/lib/zookeeper/version-2/*
# cp /tmp/zk_backup/data/* /var/lib/zookeeper/version-2/
```

RHEL:

```
# rm /var/lib/zookeeper/data/version-2/*
# cp /tmp/zk_backup/data/* /var/lib/zookeeper/data/version-2/
```

4. **Restart ZooKeeper**

   On all NSDB nodes, start the ZooKeeper service:

   Ubuntu:

```
# service zookeeper start
```

   RHEL:

```
# systemctl start zookeeper.service
```

5. **Verify ZooKeeper Operation**

   On all NSDB nodes, verify that ZooKeeper is operating properly.

   A basic check can be done by executing the `ruok` (Are you ok?) command. This will reply with `imok` (I am ok.) if the server is running in a non-error state:

```
# echo ruok | nc 127.0.0.1 2181
imok
```

   More detailed information can be requested with the `stat` command, which lists statistics about performance and connected clients:

```
# echo stat | nc 127.0.0.1 2181
Zookeeper version: 3.4.5--1, built on 06/10/2013 17:26 GMT
Clients:
 /127.0.0.1:34768[0](queued=0,recved=1,sent=0)
 /192.0.2.1:49703[1](queued=0,recved=1053,sent=1053)

Latency min/avg/max: 0/4/255
Received: 1055
Sent: 1054
Connections: 2
Outstanding: 0
Zxid: 0x260000013d
Mode: follower
Node count: 3647
```

Cassandra

MidoNet exchanges connection tracking and NAT information directly between its Agents, but also stores this data in Cassandra as a backup.

Whilst normal packet processing does not require access to Cassandra, it is necessary to support port migrations and to keep connections working across Agent reboots. Agents pull this information from Cassandra anytime they bind a new local interface to the virtual topology.

If you do not rely on above, a full data backup of Cassandra is not required.

Backup

1. **Stop Cassandra**

On all NSDB nodes, stop the Cassandra service:

Ubuntu:

```
# service cassandra stop
```

RHEL:

```
# systemctl stop cassandra.service
```

2. **Backup Cassandra Configuration**

   On all NSDB nodes, make a copy of the Cassandra configuration:

   Ubuntu:

```
# cp /etc/cassandra/cassandra.yaml /tmp/cass_backup/config/
```

   RHEL:

```
# cp /etc/cassandra/conf/cassandra.yaml /tmp/cass_backup/config/
```

3. **Backup Cassandra Data**

   On all NSDB nodes, make a copy of the Cassandra data:

```
# cp -r /var/lib/cassandra/* /tmp/cass_backup/data/
```

4. **Restart Cassandra**

   On all NSDB nodes, start the Cassandra service:

```
# service cassandra start
```

```
# systemctl start cassandra.service
```

Restore

1. **Stop Cassandra**

   On all NSDB nodes, stop the Cassandra service:

```
# service cassandra stop
```

```
# systemctl stop cassandra.service
```

2. **Restore Cassandra Configuration**

   On all NSDB nodes, restore the Cassandra configuration from your backup:

   Ubuntu:

```
# cp /tmp/cass_backup/config/cassandra.yaml /etc/cassandra/
```

   RHEL:

```
# cp /tmp/cass_backup/config/cassandra.yaml /etc/cassandra/conf/
```

3. **Restore Cassandra Data**

   On all NSDB nodes, restore the Cassandra data from your backup:

```
# rm -r /var/lib/cassandra/*
```

```
# cp -r /tmp/cass_backup/data/* /var/lib/cassandra/
```

4. **Restart Cassandra**

   On all NSDB nodes, start the Cassandra service:

   Ubuntu:

```
# service cassandra start
```

   RHEL:

```
# systemctl start cassandra.service
```

5. **Verify Cassandra Operation**

   On all NSDB nodes, verify that Cassandra is operating properly.

   A basic check can be done by executing the `nodetool status` command. This will reply with UN (Up / Normal) in the first column if the servers are running in a non-error state:

```
$ nodetool --host 127.0.0.1 status
[...]
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns   Host ID
       Rack
UN  192.0.2.1  123.45 KB  256     33.3%
 11111111-2222-3333-4444-555555555555  rack1
UN  192.0.2.2  234.56 KB  256     33.3%
 22222222-3333-4444-5555-666666666666  rack1
UN  192.0.2.3  345.67 KB  256     33.4%
 33333333-4444-5555-6666-777777777777  rack1
```

MidoNet Agent

Backup

1. **Backup Agent Configuration**

   On the Agent nodes, make a copy of the Agent configuration:

```
# cp /etc/midolman/midolman.conf /tmp/mn-agent_backup/config/
# cp /etc/midolman/midolman-env.sh /tmp/mn-agent_backup/config/
```

Restore

1. **Restore Agent Configuration**

   On the Agent nodes, restore the Agent configuration from your backup:

```
# cp /tmp/mn-agent_backup/config/* /etc/midolman/
```

2. **Restart Agent**

   On the Agent nodes, restart the Midolman service:

   Ubuntu:

```
# service midolman restart
```

RHEL:

```
# systemctl restart midolman.service
```

MidoNet Cluster

Backup

1. **Backup Cluster Configuration**

   On the Cluster nodes, make a copy of the Cluster configuration:

   ```
   # cp /etc/midonet/midonet.conf /tmp/mn-cluster_backup/config/
   ```

Restore

1. **Restore Cluster Configuration**

   On the Cluster nodes, restore the Cluster configuration from your backup:

   ```
   # cp /tmp/mn-cluster_backup/config/* /etc/midonet/
   ```

2. **Restart Cluster**

   On the Cluster nodes, restart the Cluster service:

   Ubuntu:

   ```
   # service midonet-cluster restart
   ```

   RHEL:

   ```
   # systemctl restart midonet-cluster.service
   ```

MySQL / MariaDB

Technically MidoNet only relies on the Neutron and Keystone tables, but for completeness and ease of use a full backup of the database is recommended.

Backup

Important

To ensure that Neutron's MySQL database and MidoNet's ZooKeeper data are in sync, set the MidoNet API to read-only before starting the backup.

Backup both MySQL database and ZooKeeper data (as described in the section called "ZooKeeper" [198]) at the same time!

1. **Set MidoNet API to READONLY**

   Set the MidoNet API to READONLY to avoid topology changes during the backup process:

   ```
   # midonet-cli
   midonet> set system-state availability READONLY
   ```

2. **Create Database Dump**

   Use the `mysqldump` command to create a database backup:

```
# mysqldump \
   -uroot -p_*password*_ \
   --lock-tables --events --routines --triggers --all-databases | \
  gzip > /tmp/ost_backup/data/ost_database.sql.gz
```

3. **Create ZooKeeper Backup**

   Create the ZooKeeper backup (as described in the section called “ZooKeeper” [198]) to ensure that MySQL and ZooKeeper data are in sync.

4. **Reset MidoNet API to READWRITE**

   Reset the MidoNet API to READWRITE:

```
# midonet-cli
midonet> set system-state availability READWRITE
```

Restore

1. **Stop Affected Services**

   Stop all services that access the database.

   This includes, but is **not limited** to the following services (depending on your OpenStack installation):

   Ubuntu:

```
# service apache2 stop
# service keystone stop
# service neutron-server stop
[...]
```

   RHEL:

```
# systemctl stop openstack-keystone.service
# systemctl stop neutron-server.service
# systemctl stop httpd.service
[...]
```

2. **Restore Database Dump**

```
# gunzip -c /tmp/ost_backup/data/ost_database.sql.gz | mysql -uroot -
p_*password*_
```

3. **Restart Services**

   Restart the services.

   Ubuntu:

```
# service apache2 start
# service keystone start
# service neutron-server start
[...]
```

   RHEL:

```
# systemctl start openstack-keystone.service
```

```
# systemctl start neutron-server.service
# systemctl start httpd.service
[...]
```

Neutron

Backup

1. **Backup Neutron Configuration**

   Make a copy of the Neutron configuration:

   ```
   # cp /etc/neutron/neutron.conf /tmp/neutron_backup/config/
   # cp /etc/neutron/plugins/midonet/midonet.ini /tmp/neutron_backup/
   config/
   ```

Restore

1. **Restore Neutron Configuration**

   Restore the Neutron configuration from your backup:

   ```
   # cp /tmp/neutron_backup/config/neutron.conf /etc/neutron/
   # cp /tmp/neutron_backup/config/midonet.ini /etc/neutron/plugins/
   midonet/
   ```

2. **Restart Neutron**

   Restart the Neutron service.

   Ubuntu:

   ```
   # service neutron-server restart
   ```

   RHEL:

   ```
   # systemctl restart neutron-server.service
   ```

Libvirt

Backup

1. **Backup Libvirt Configuration**

   On the Compute nodes, make a copy of the Libvirt configuration:

   ```
   # cp /etc/libvirt/qemu.conf /tmp/libvirt_backup/
   ```

Restore

1. **Restore Libvirt Configuration**

   Restore the Libvirt configuration from your backup:

   ```
   # cp /tmp/libvirt_backup/qemu.conf /etc/libvirt/
   ```

2. **Restart Libvirt**

Restart the Libvirt service.

Ubuntu:

```
# service libvirt-bin restart
```

RHEL:

```
# systemctl restart libvirtd.service
```