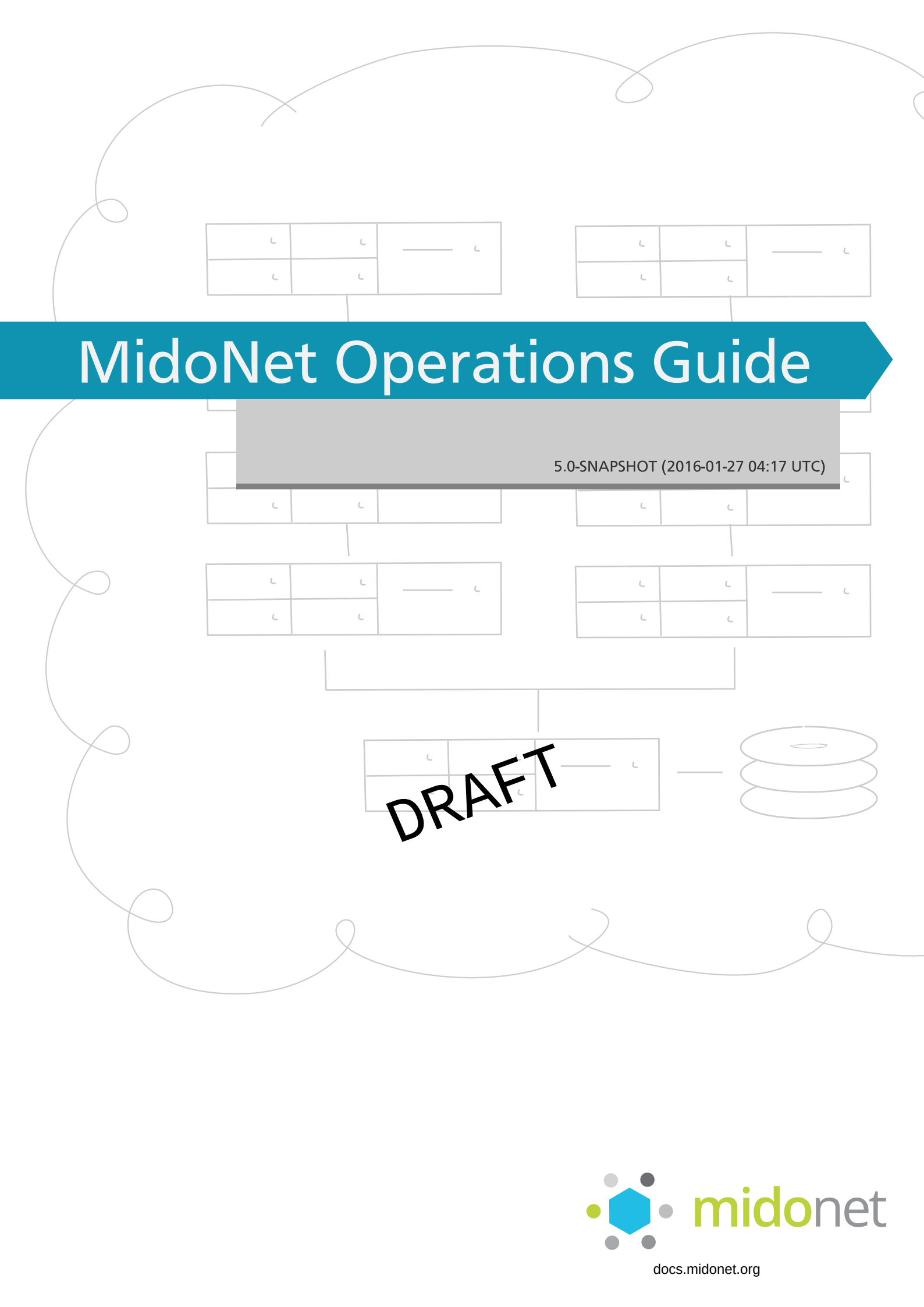


MidoNet Operations Guide

5.0-SNAPSHOT (2016-01-27 04:17 UTC)

DRAFT

midonet

docs.midonet.org

MidoNet Operations Guide

5.0-SNAPSHOT (2016-01-27 04:17 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.

Caution

This document is a DRAFT. It may be MISSING relevant information or contain UNTESTED information. Use it at your own risk.

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" [17].

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
    address 198.51.100.2 net 198.51.100.0/30
   port port1 device router0 state up mac fa:16:3e:22:22:22
    address 203.0.113.2 net 203.0.113.0/30
   [...]
   ```

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 address 10.22.22.1 net 10.22.22.0/24
router0:port1
midonet>
midonet> router router0 list port
port port0 device router0 state up plugged no mac ac:ca:ba:ab:ed:b8 address
 10.12.12.1 net 10.12.12.0/24
port port1 device router0 state up plugged no mac ac:ca:ba:5e:0a:02 address
 10.22.22.1 net 10.22.22.0/24
```

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 address 172.19.0.2 net 172.19.0.0/30
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.v2_0.KeystoneService`	Uses external Keystone identity service, version 2.
`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.v2_0.
KeystoneService" | mn-conf set -t default
```

Important

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

For additional Keystone configuration options, see the section called “Enabling Keystone authentication” [9]

Mock authentication

Mock authentication disables the authentication system by returning a 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.v2_0.KeystoneService
```

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.protocol

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

Table 2.2. Keystone Protocol

Class	Description
`http`	Uses plain text HTTP to communicate with the Keystone server.
`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.admin_token

Identifies the token of the admin user in Keystone that the API uses to make requests to Keystone.

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.

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 gre
   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 gre
   tzone tzone1 name gre type gre
   ```

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 gre
tzone tzone1 name gre type gre
```

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 gre type gre
```

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 address 10.100.1.1 net 10.0.0.0/24
   ```

```
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 address 10.100.
1.1 net 10.0.0.0/24
```

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 addresses

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 address 100.
   100.100.1 net 100.100.100.0/30
   port port1 device router0 state up mac 02:cb:3d:85:89:2a address 172.
   168.0.1 net 172.168.0.0/16
   port port2 device router0 state up mac 02:46:87:89:49:41 address 200.
   200.200.1 net 200.200.200.0/24 peer bridge0:port0
   port port3 device router0 state up mac 02:6b:9f:0d:c4:a8 address 169.
   254.255.1 net 169.254.255.0/30
   ```

   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” [15] and the section called “Adding a bridge” [16] 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 address 10.100.
1.1 net 10.0.0.0/24
```

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 address 10.100.
1.1 net 10.0.0.0/24 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” [15] 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 address 10.100.
1.1 net 10.0.0.0/24 peer bridge1:port0
port port1 device router1 state up mac 02:fa:5f:87:bb:d2 address 10.100.
1.2 net 10.0.0.0/24
```

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 address 10.100.
1.128 net 10.0.0.0/24
```

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 address 10.100.
1.1 net 10.0.0.0/24 peer bridge1:port0
port port1 device router1 state up mac 02:fa:5f:87:bb:d2 address 10.100.
1.2 net 10.0.0.0/24 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

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 destina-

tion, 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 address 169.254.255.3 net
 169.254.255.0/30
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
```

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. Otherwise, it returns exactly the (next_action, new_packet) returned by invoking the rule chain.

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” [27] 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 example

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):

1. 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 port 1:

1. 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:

1. 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:

2. 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)

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 bridgebridge 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 pre-routing rule chain

To list the pre-routing rule chain for port 1 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.
- 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.
- 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 as a result of matching previous drop 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)

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 example

This example shows how to use MidoNet CLI commands to configure SNAT on a router port.

For example, you might want to configure SNAT for a network device (like a VM) with a private IP address and a public floating IP address that sends data to a destination outside its local network.

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 private IP address to the VM's public IP address.

To create the rule chain for the above SNAT configuration:

1. Create a new rule chain:

```
midonet> chain create name "snat-test"
chain7
```

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

```
midonet> chain chain7 add rule src 10.100.1.150 out-ports router1:port0
 pos 1 type snat action accept target 198.51.100.4
chain7:rule2
```

3. Create a rule that accepts traffic from the private network and performs SNAT to translate the source IP address to the IP address of the router's gateway to the public network.

```
midonet> chain chain7 add rule out-ports router1:port0 pos 2 type snat
 action accept target 198.51.100.2
```

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 external/publicly routable addresses (for example, from users of the service all over the world) to a virtual IP address (VIP) (often a public, floating IP address, but 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.

MidoNet load balancer limitations

MidoNet uses the Neutron API to set up load balancers (as documented in https://wiki.openstack.org/wiki/Neutron/LBaaS/API_1.0) but not all the Neutron LBaaS features are supported in MidoNet:

- L7 load balancing is not supported.
- There are no pool statistics.

- Only round robin is supported in as 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).
- Only Source IP session persistence (or 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.
- No ICMP for health checks.
- There is no connection limit.
- Only one health monitor per pool.
- 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.

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” [47].

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 address 100.
100.100.1 net 100.100.100.0/30
port port1 device router0 state up mac 02:cb:3d:85:89:2a address 172.
168.0.1 net 172.168.0.0/16
port port2 device router0 state up mac 02:46:87:89:49:41 address 200.
200.200.1 net 200.200.200.0/24 peer bridge0:port0
port port3 device router0 state up mac 02:6b:9f:0d:c4:a8 address 169.
254.255.1 net 169.254.255.0/30 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. 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" [26] 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
  ```

11. 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
```

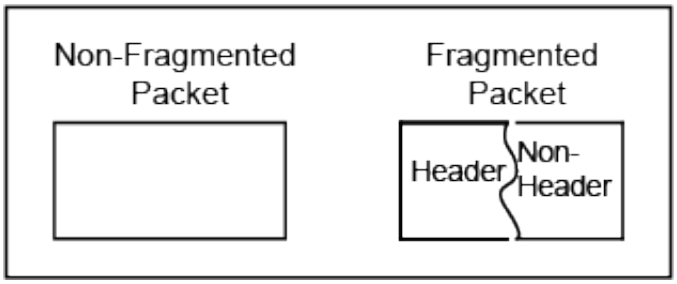
With the above rule chain, MidoNet handles fragmented packet with the destination TCP port 80 as follows:

- 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

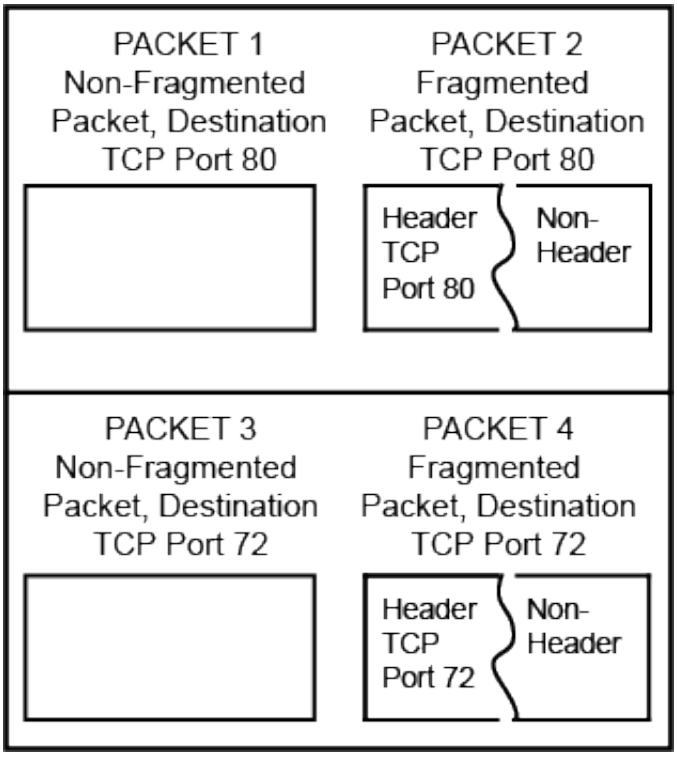
To address this problem, MidoNet provides a mechanism to handle the fragmented packets. This mechanism allows the fragmented packets to reach their destination, as shown in the following example. The drawing below simply depicts a whole, non-fragmented packet and a fragmented packet that consists of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

12. 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.

13. 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, such as Zabbix™, Munin, and so on.

This chapter describes the main available metrics for each service, as well as a procedure to configure a basic monitoring infrastructure based on Munin, based on scripts provided with the MidoNet deployment package.

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 Cassandra and ZooKeeper 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.

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").

Below are descriptions of the graphs resulting from the example Munin configurations provided in the MidoNet deployment repository. The graphs are built from a subset of the Cassandra JMX service. The available graphs are:

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.

In the aggregated view, an additional "clock" category configured in Munin shows the result of the time command in each of the nodes. Expect to see a straight line, as close as possible with the lines of all Cassandra nodes overlapping. Otherwise, this indicates divergences in the host's clocks that will almost certainly end up causing problems on conflict resolution. You should attend to this urgently and ensure that all hosts are using the Network Time Protocol (NTP).

The installation script also provides graphs to monitor the state of Cassandra's Java virtual machine (JVM):

- 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

The install_plugins.sh script also installs a set of Munin plugins and configuration files to generate graphs from ZooKeeper's exposed JMX metrics.

However, on each node, you need to indicate the ZooKeeper replica number; the script provides clear guidance on how to do this.

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.

Below are descriptions of the graphs resulting from the example Munin configurations provided in the MidoNet deployment repository. The graphs are built from a subset of the ZooKeeper JMX service. The available graphs are:

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.

The install_plugins.sh script can configure all the relevant Munin plugins.

Below are descriptions of the graphs resulting from the example Munin configurations provided in the MidoNet deployment repository. The graphs are built from a subset of the Midolman JMX service. The available graphs are:

Current Pended Packets

Midolman simulates a single packet for each wildcard-flow match. When a packet A is being simulated, if another packet B with the same flow match appears, it will be pended until A finishes its simulation. At this point, B will be sent straight to the datapath with the same actions as A applied. This metric displays the total count of pended packets at a given point in time. Ideally, this value should be as low as possible. Large values indicate that Midolman is being flooded with packets with identical matches. A steadily growing figure may also indicate that pended packets are not being executed, which is likely caused by a bug.

Datapath Flows

Shows the count of currently active flows in the datapath, and the rate of creation. This depends significantly on the nature of the traffic.

Wildcard Flows

Shows the count of currently active wildcard flows inside Midolman. This number should be similar to the Datapath Flow count, but not necessarily equal.

Simulation Latency

Shows the time taken by Midolman to run the simulation of packets through the virtual topology. This value is fundamental for network latency.

Wildcard Flow Table Latency

Shows the access times to Midolman's internal wildcard-flow table. This value is relevant for network latency.

Throughput

Shows the packets per second that are being processed and dropped. When Midolman becomes saturated, the expected behavior is to stabilize processed packets in a relatively flat line, increasing the dropped rate with any additional traffic. However, pressure from the Netlink queue may cause performance degradation.

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

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. Midolman has very specific memory-usage constraints because it aims for a very low memory and CPU footprint. At the same time, simulations generate a significant amount of short-lived garbage.

- 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 max./min. 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 ConcurrentMarkSweep's collector, (which runs only on the old generation). 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 Midolman.

Munin offers some generic metrics that are very relevant to understanding the performance of Midolman.

CPU Usage

Under high traffic, Midolman 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 Midolman 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 13.1. 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 13.2. 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 address
 192.168.1.1 net 192.168.1.0/24 peer bridge0:port0
port port1 device router0 state up plugged no mac ac:ca:ba:a0:6b:43 address
 10.0.0.1 net 10.0.0.0/24
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
```

14. 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” [81]) 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” [92].

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” [85].

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).

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" [92], 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” [95]

Example

```
midonet> vtep vtep0 delete
```

15. 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

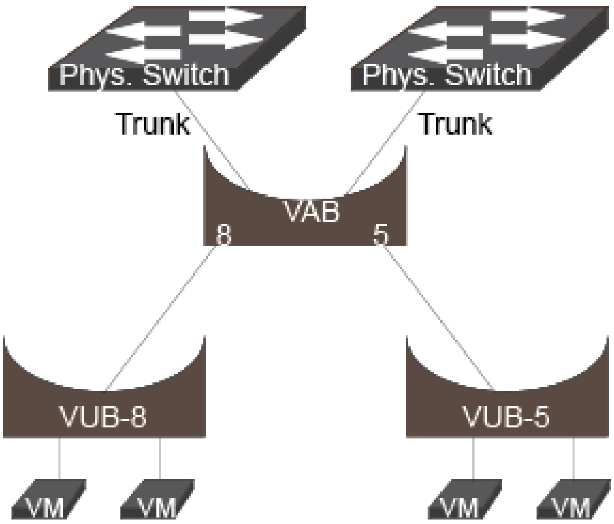
You can configure MidoNet's bridge virtual ports with a single VLAN ID, thereby introducing a change in the behavior of the processing of VLAN-tagged frames.

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 15.1. 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 15.1, “Topology with VLANs and L2 Gateway” [97].

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.

16. 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.

17. 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

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 17.1. 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 time-

out, 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
}
```

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" [102].

Configuration options

Table 17.2. Admin Roles

Configuration Key	Default Value	Description
`cluster.rest_api.enabled`	`true`	Specifies whether the REST API service is enabled at the MidoNet cluster

Configuration Key	Default Value	Description
		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 17.3. 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>
```

18. 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	8080	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-novnc-proxy	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	Prot ocol	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		Network-NameSpace mbgp[Peer Number]_ns
MidoNet	quagga bgpd bgp	TCP	179		BGP neighbor	Network-NameSpace mbgp[Peer Number]_ns