

MidoNet Operations Guide

5.0-SNAPSHOT (2016-01-04 14:15 UTC)

DRAFT



MidoNet Operations Guide

5.0-SNAPSHOT (2016-01-04 14:15 UTC)

Copyright © 2016 Midokura SARL All rights reserved.

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.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

Preface	vii
Conventions	vii
1. Configuring uplinks	1
Edge Router Setup	1
BGP Setup	2
Static Setup	7
2. Authentication and authorization	9
Available authentication services in MidoNet	9
Using the Keystone authentication service	10
3. Admitting resources to MidoNet	12
What are tunnel zones?	12
Working with hosts	13
4. Device abstractions	16
Creating a router	16
Adding a port to a router	16
Adding a bridge	17
Adding a port to a bridge	17
Binding an exterior port to a host	17
Stateful port groups	18
5. Connecting devices	20
Connecting a bridge to a router	20
Connecting two routers	21
6. Routing	22
Routing process overview	22
Viewing routes	24
Adding routes	25
Deleting routes	26
7. Rule chains	27
A packet's flow within a router	27
A packet's flow within a rule chain	28
Rule types	29
Rule order	30
Rule conditions	31
MidoNet rule chain example	35
Listing the bridges for the tenant	37
Listing the OpenStack security group rule chain	38
8. Network Address Translation	40
Static NAT	40
Viewing NAT rule chain information	40
Configuring SNAT, DNAT, and REV_DNAT	42
DNAT/REV_DNAT example	42
SNAT example	43
9. Layer 4 Load Balancing	45
Load balancer configuration	46
Sticky Source IP	48
Health monitor	49
10. L2 address masking	52
L2 address mask rule chain example	52
11. Handling fragmented packets	53
Definitions and allowed values	53
Fragmented packets rule chain creation example	54
Non-Fragmented and Fragmented Packets	55

Fragmented and Non-Fragmented Packets with Different Destinations	56
12. MidoNet resource protection	58
Introduction	58
Expected Behavior	58
Configuration	59
Disabling Resource Protection	59
13. MidoNet monitoring	60
Metering	60
Monitoring Network State Database	62
Monitoring Midolman Agents	65
Monitoring events	67
Packet Tracing	76
Port mirroring	77
14. VXLAN configuration	81
VXLAN Gateway	81
VXLAN Coordinator	82
VXLAN Flooding Proxy	83
Connecting to the VTEP	83
Setting up a connection between a VTEP and a Neutron network	85
Enabling connection between VTEP and MidoNet hosts	86
Troubleshooting VTEP/VXGW configuration	87
CLI commands used for working with the VXGW	92
15. Setting up an L2 gateway	98
Configuring an L2 gateway	99
Fail-over/Fail-back	100
16. Working with the MidoNet CLI	101
Using the MidoNet CLI	101
17. Advanced configuration and concepts	103
MidoNet Configuration: mn-conf	103
Recommended configurations	108
MidoNet Agent (Midolman) configuration options	109
Advanced MidoNet REST API configuration options	111
Cassandra cache	113
18. MidoNet and OpenStack TCP/UDP service ports	115
Services on the Controller node	115
Services on the Network State Database nodes	116
Services on the Compute nodes	117
Services on the Gateway Nodes	117

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

List of Tables

2.1. Authentication Provider Classes	9
2.2. Keystone Protocol	10
7.1. CLI Rule Chain Attributes	31
7.2. CLI Rule Chain Attributes That Match Packets	32
13.1. Configuration Files/Locations	68
13.2. Event Message Files/Locations	68
17.1. Recommended Configuration Values	109
17.2. Admin Roles	111
17.3. System Properties for the HTTPS Key Store	112

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

Edge Router Setup	1
BGP Setup	2
Static Setup	7

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.

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. Determine the Keystone admin tenant ID

Use the `keystone` command to determine the Keystone `admin` tenant's ID:

```
$ keystone tenant-list
```

id	name	enabled
12345678901234567890123456789012	admin	True

2. Launch the MidoNet CLI and find the Edge Router

```
$ midonet-cli
midonet-cli>
```

Because the Edge Router is not associated with a tenant, the active tenant has to be cleared (`cleart`) first.

```
midonet-cli> cleart

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

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

3. Load the `admin` tenant

Before continuing with further configuration, the `admin` tenant has to be set (`settt`). Use the ID you got from Keystone above.

```
midonet-cli> sett 12345678901234567890123456789012
tenant_id: 12345678901234567890123456789012
```

4. Create virtual ports for the BGP sessions

For each remote BGP peer, create a port on the Edge Router that is going to be used for BGP communication.

```
midonet> router router0 add port address 198.51.100.2 net
198.51.100.0/30
router0:port0

midonet> router router0 add port address 203.0.113.2 net 203.0.113.0/30
router0:port1

midonet> router router0 port list
port port0 device router0 state up mac ac:ca:ba:11:11:11
address 198.51.100.2 net 198.51.100.0/30
port port1 device router0 state up mac ac:ca:ba:22:22:22
address 203.0.113.1 net 203.0.113.0/30
[...]
```

In this example the created ports are **port0** and **port1**.

5. Configure BGP on the virtual ports

```
midonet> router router0 set asn 64512
midonet> router router0 add bgp-peer asn 64513 address 198.51.100.1
router0:peer0

midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 198.51.100.1

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
```

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
```

8. Bind virtual ports to physical network interfaces

Bind the Edge Router's virtual ports to the physical network interfaces on the Gateway Nodes.



Important

Ensure that physical interfaces are in state UP and do not have an IP address assigned.

a. List the MidoNet hosts and find the Gateway Nodes:

```
midonet> host list
host host0 name gateway1 alive true
host host1 name gateway2 alive true
[...]
```

In this example the hosts are **host0** and **host1**.

b. List the Gateway Nodes' physical interfaces:

```
midonet> host host0 list interface
[...]
iface eth1 host_id host0 status 3 addresses [] mac 01:02:03:04:05:06
mtu 1500 type Physical endpoint PHYSICAL
[...]

midonet> host host1 list interface
[...]
iface eth1 host_id host0 status 3 addresses [] mac 06:05:04:03:02:01
mtu 1500 type Physical endpoint PHYSICAL
[...]
```

c. Bind the physical host interfaces to the Edge Router's virtual ports:

```
midonet> host host0 add binding port router0:port0 interface eth1
host host0 interface eth1 port router0:port0

midonet> host host1 add binding port router0:port1 interface eth1
host host1 interface eth1 port router0:port1
```

d. Configure a stateful port group:

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

e. Add the ports to the port group:

```
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

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

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

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
  }
}
```

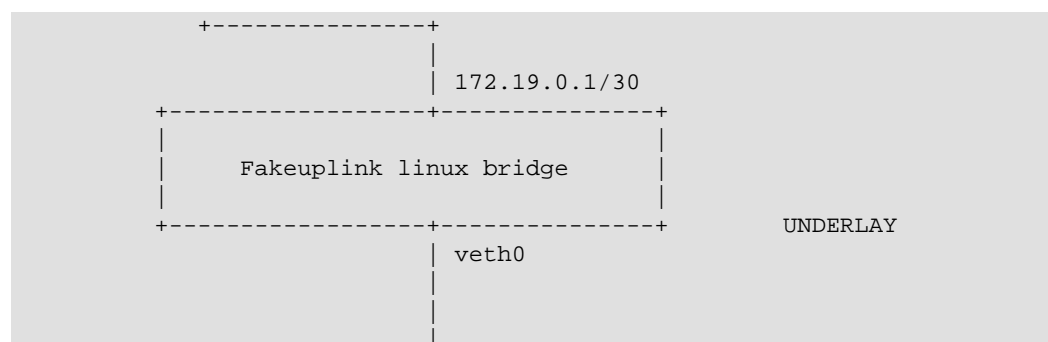
Static Setup

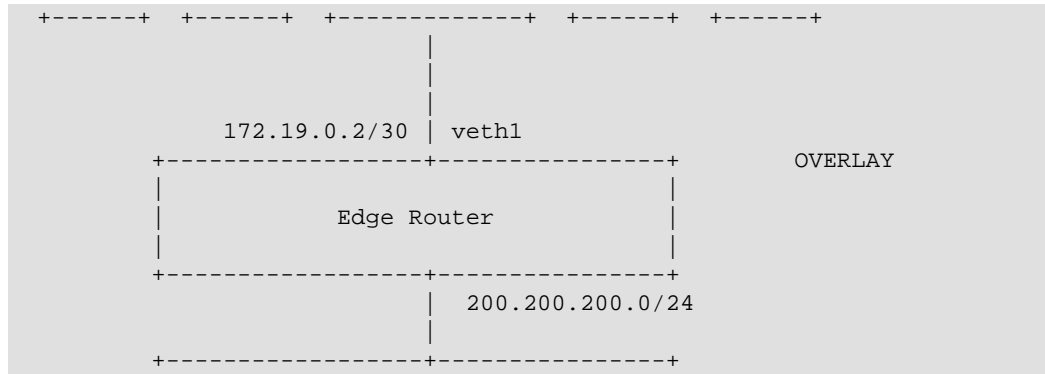
If you are not connecting through a BGP link, or you just want to use static routing follow this section.

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:





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

Available authentication services in MidoNet	9
Using the Keystone authentication service	10

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
<code>org.midonet.cluster.auth.keystone.v2_0.KeystoneService</code>	Uses external Keystone identity service, version 2.
<code>org.midonet.cluster.auth.MockAuthService</code>	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
```


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
```


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
```


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
```


AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

- AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

- AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

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'0:0:0:0:0:0:1'] 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 to 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
```


Connecting two routers

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

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

To connect two routers:

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

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

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

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

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-


```
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

A packet's flow within a router	27
A packet's flow within a rule chain	28
Rule types	29
Rule order	30
Rule conditions	31
MidoNet rule chain example	35
Listing the bridges for the tenant	37
Listing the OpenStack security group rule chain	38

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.

- AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

- AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

- AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

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 ingresses 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	<p>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.</p> <p>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.</p> <p>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.</p> <p>0000.0000.0000: No bits are considered when applying the hw-src test, so any packet will match.</p>
hw-dst-mask	<p>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.</p> <p>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.</p>

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:



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_01fcelb8-c277-4a37-a8cc-86732eea186d_INGRESS
chain chain4 name OS_SG_050593ed-56ad-44ef-8489-4052d02d99ff_EGRESS
chain chain7 name OS_SG_01fcelb8-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:

- 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)

- 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
  dn timer 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_sn timer 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:

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
```



```
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" \[48\]](#).

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).


```
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.

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
```

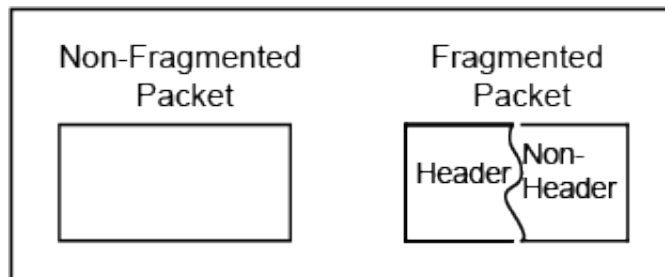
With the above rule chain, MidoNet handles fragmented packet with the destination TCP port 80 as follows:

- 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

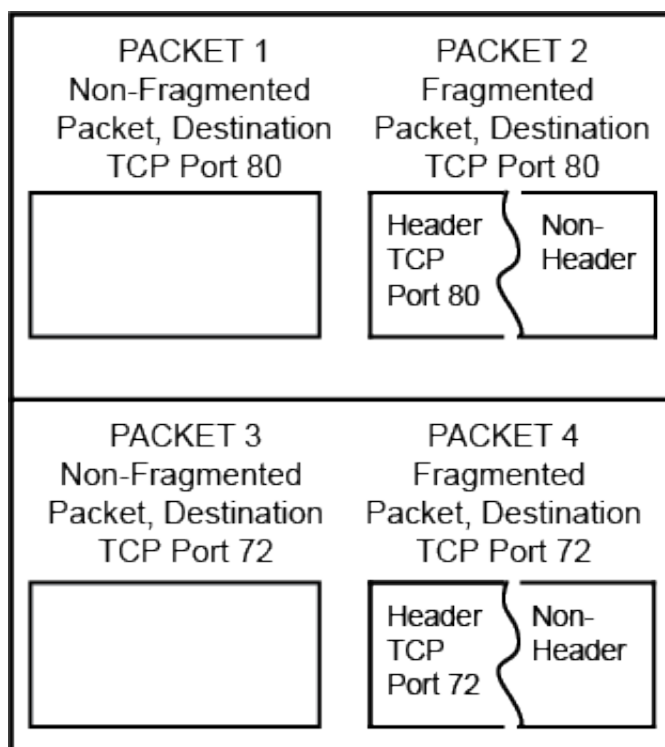
To address this problem, MidoNet provides a mechanism to handle the fragmented packets. This mechanism allows the fragmented packets to reach their destination, as shown in the following example. The drawing below simply depicts a whole, non-fragmented packet and a fragmented packet that consists of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



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 send traffic.
- `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

Metering	60
Monitoring Network State Database	62
Monitoring Midolman Agents	65
Monitoring events	67
Packet Tracing	76
Port mirroring	77

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 `FOO` in MidoNet be counted under meter `meters:device:FOO`. 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.

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.

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}.

Port

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

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.

14. VXLAN configuration

Table of Contents

VXLAN Gateway	81
VXLAN Coordinator	82
VXLAN Flooding Proxy	83
Connecting to the VTEP	83
Setting up a connection between a VTEP and a Neutron network	85
Enabling connection between VTEP and MidoNet hosts	86
Troubleshooting VTEP/VXGW configuration	87
CLI commands used for working with the VXGW	92

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:

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" \[93\]](#), 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?


```
ERROR vxqw-vtep-ba2739df-87cf-458f-9ad2-39885cab217d VTEP 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.

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
```

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((swpl,0))
```

At this point it will ensure that a connection is established 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:

90


```
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
```


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" \[96\]](#)

Example

```
midonet> vtep vtep0 delete
```



```
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

Using the MidoNet CLI 101

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.



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_buffer_capacity	128	256	128
mn-conf(1)	[agent.datapath]	tunnel_incoming_buffer_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 keypace 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-

Configuration Key	Default Value	Description
		instance where the configuration applies.
<code>cluster.rest_api.http_port</code>	8181	Specifies the listening port for the HTTP end-point.
<code>cluster.rest_api.https_port</code>	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
<code>midonet.keystore_path</code>	<code>/etc/midonet-cluster/ssl/midonet.jks</code>	The name of the key store file.
<code>midonet.keystore_password</code>	<i>none</i>	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
```

<https://www.eclipse.org/jetty/documentation/current/configuring-ssl.html>

This section describes details relative to the Cassandra-based implementation of the Cassandra cache used for connection tracking and NAT mappings.

Cassandra operations are now asynchronous, so losing connectivity to Cassandra should not impact simulations.

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.

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.

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>
```

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

AFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

Category	Service	Prot ocol	Port	Self	Controller	Compute	Mgmt. PC
OpenStack	mongod (ceilometer)	TCP	27017	x	x	x	
OpenStack	MySQL	TCP	3306	x	x	x	

Services on the Network State Database nodes

This section lists the TCP/UDP ports used by the services on the Network State Database nodes.

Category	Service	Prot ocol	Port	Self	Controller	NSDB	Compute	Comment
MidoNet	ZooKeeper communication	TCP	3888	x		x		
MidoNet	ZooKeeper leader	TCP	2888	x		x		
MidoNet	ZooKeeper/Cassandra	TCP	random	x				ZooKeeper/Cassandra "LISTEN" to a TCP high number port. The port number is randomly selected on each ZooKeeper/Cassandra host.
MidoNet	Cassandra Query Language (CQL) native transport port	TCP	9042					
MidoNet	Cassandra cluster communication	TCP	7000	x		x		
MidoNet	Cassandra cluster communication (Transport Layer Security (TLS) support)	TCP	7001	x		x		
MidoNet	Cassandra JMX	TCP	7199	x				JMX monitoring port. If you're using this port to monitor Cassandra health, enable communication from the monitoring server.
MidoNet	ZooKeeper client	TCP	2181	x	x	x	x	
MidoNet	Cassandra clients	TCP	9160	x	x	x	x	

Services on the Compute nodes

This section lists the TCP/UDP ports used by the services on the Compute nodes.

Category	Service	Protocol	Port	Self	Controller	Comment
OpenStack	qemu-kvm vnc	TCP	From 5900 to 5900 + # of VM		x	
MidoNet	midolman	TCP	random	x		midolman "LISTEN"s to a TCP high number port. The port number is randomly selected on each MN Agent host.
OpenStack	libvirtd	TCP	16509	x	x	
MidoNet	midolman	TCP	7200	x		JMX monitoring port If you're using this port to monitor health, enable communication from the monitoring server.
MidoNet	midolman	TCP	9697	x		If enabled, MidoNet Metadata Proxy listens on 169.254.169.254:9697 to accept metadata requests.

Services on the Gateway Nodes

This section lists the TCP/UDP ports used by the services on the Gateway Nodes.

Category	Service	Protocol	Port	Self	Misc.	Comment
MidoNet	midolman	TCP	random	x		midolman LISTEN"s to a TCP high number port. The port number is randomly selected on each MN Agent host.
MidoNet	midolman	TCP	7200	x	x	JMX monitoring port If you're using this port to monitor health, enable communication from the monitoring server.
MidoNet	quagga bgpd control	TCP	2606	x		Network-NameSpace mbgp[Peer Number]_ns
MidoNet	quagga bgpd bgp	TCP	179		BGP neighbor	Network-NameSpace mbgp[Peer Number]_ns