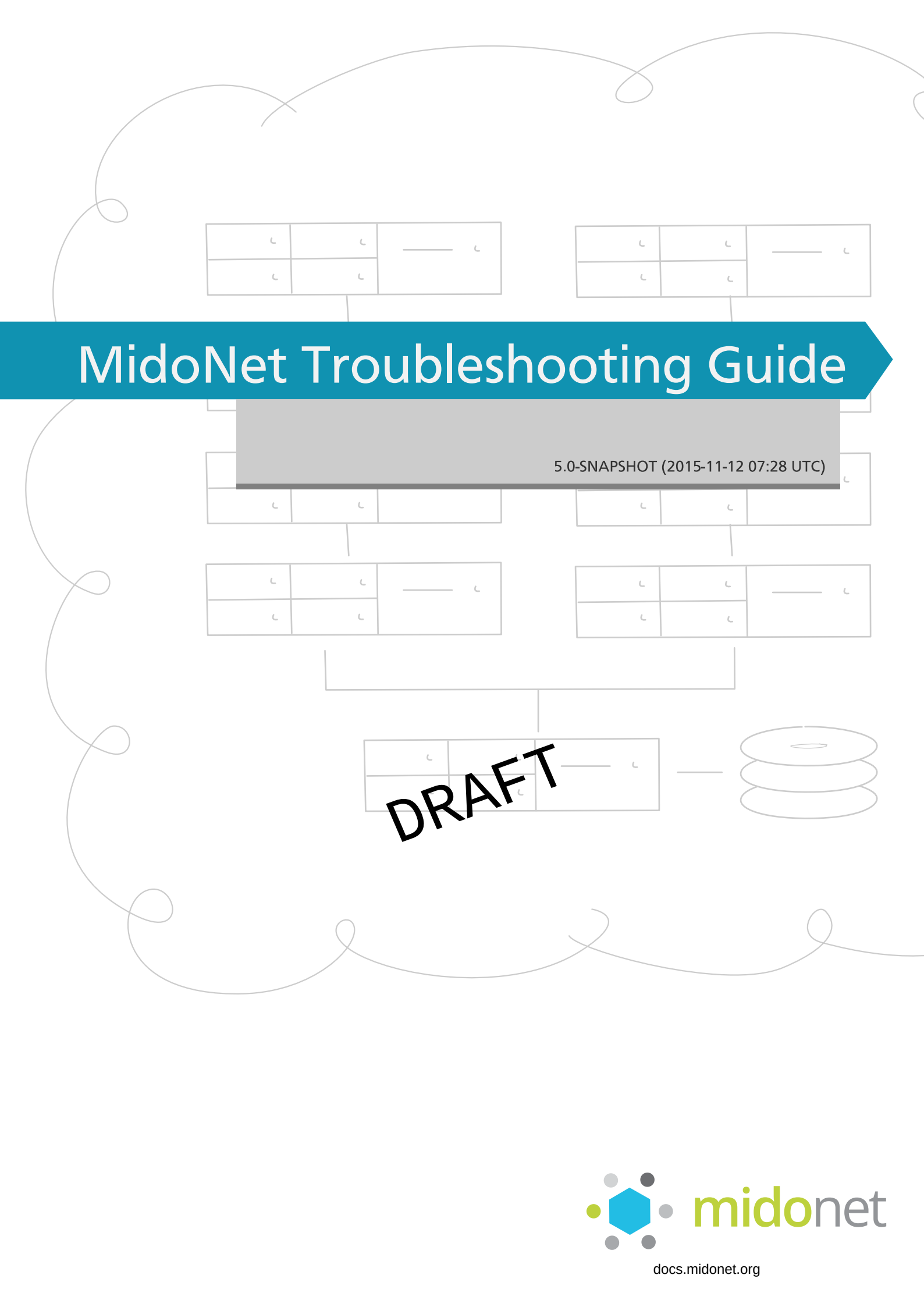

MidoNet Troubleshooting Guide

5.0-SNAPSHOT (2015-11-12 07:28 UTC)

midonet

docs.midonet.org

MidoNet Troubleshooting Guide

5.0-SNAPSHOT (2015-11-12 07:28 UTC)

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

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

This document contains useful information on troubleshooting MidoNet and OpenStack related issues.

Caution

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

Note

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

Table of Contents

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. Overall Approach

Table of Contents

When troubleshooting a MidoNet environment, there are multiple layers to be checked:

- Underlay Network
- Overlay Network
- Virtual Network Topology Simulation
- Virtual Network Topology

Layer	Components
Virtual Network Topology	Neutron, MidoNet NSDB
Virtual Network Topology Simulation	MidoNet Agent
Overlay Network	Tunnel, Datapath
Underlay Network	Physical Environment, Operating System

To rule out possible issues these layers shall be checked from bottom to top.

Underlay Network

The underlay network, i.e. the physical network, should be the first starting point to check for any connectivity issues:

- Hardware / Cabling
- Routing
- Firewall
- Access Control
- Linux Kernel / Open vSwitch
- Time Synchronization

Hardware / Cabling

Ensure that the hardware works properly.

1. **Is the physical link established?**

```
# ethtool eth0
```

```
Settings for eth0:
[...]
   Link detected: yes
```

2. **Is the interface up?**

```
# ip link
[...]
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
 mode DEFAULT group default qlen 1000
    link/ether aa:bb:cc:dd:ee:ff brd ff:ff:ff:ff:ff:ff
```

Routing

Ensure that the routing is configured correctly and check connectivity between hosts via the `ping` command.

```
# netstat -nr
Destination   Gateway       Genmask         Flags   MSS   Window   irtt   Iface
0.0.0.0       192.168.0.1   0.0.0.0         UG      0     0        0      eth0
192.168.0.0   0.0.0.0       255.255.255.0   U       0     0        0      eth0
```

```
# ip route
default via 192.168.0.1 dev eth0  proto static
192.168.0.0/24 dev eth0  proto kernel  scope link  src 192.168.96.100
 metric 1
```

Firewall

Ensure that the firewall is not blocking necessary protocols, hosts or ports.

If unsure, disable the firewall and verify if connectivity issues still persist.

```
# iptables -L
```

Access Control

Ensure that no access control system, such as SELinux or AppArmor, is blocking necessary functionality.

If unsure, disable the ACL system and verify if issues still persist.

Linux Kernel / Open vSwitch

Ensure that the Open vSwitch kernel module is loaded and matches the running Kernel's version.

```
# modinfo openvswitch
filename:       /lib/modules/kernel_version/kernel/net/openvswitch/
openvswitch.ko
license:        GPL
description:    Open vSwitch switching datapath
depends:        libcrc32c,vxlan,gre
intree:         Y
```

```
# lsmod | grep openvswitch
openvswitch             70743  0
vxlan                   37584  1 openvswitch
gre                     13808  1 openvswitch
```

```
libcrc32c              12644  2 xfs,openvswitch
```

Time Synchronization

Ensure that the time is synchronized across all nodes.

```
# ntpq -pn
     remote           refid      st t when poll reach   delay   offset
 jitter
===========================================================================
===
*157.7.153.56    133.243.238.164  2 u  114  128  377    4.239    2.713   6.
608
+106.186.114.89  9.22.27.124      3 u   73  128  377    4.845    5.069   4.
802
+157.7.235.92    10.84.87.146     2 u  115  128  377    4.326   14.744   8.
498
+122.215.240.52  133.243.238.164  2 u   45  128  377    4.291    5.400   4.
462
+91.189.94.4     131.188.3.220    2 u   75  128  367  229.564    4.604   6.
896
```

Ensure that correct time zones are configured.

```
# date
Thu Mar 26 13:24:34 JST 2015
```

Overlay Network

The overlay network, i.e. the physical network, should be the first starting point to check for any connectivity issues:

Tunnel Zone

Ensure that the hosts running the MidoNet Agent have been added to the tunnel zone and are alive.

```
# midonet-cli
midonet> list tunnel-zone
tzone tzone0 name tz type vxlan
midonet> tunnel-zone tzone0 list member
zone tzone0 host host0 address 192.168.0.1
zone tzone0 host host1 address 192.168.0.2
zone tzone0 host host2 address 192.168.0.3
zone tzone0 host host3 address 192.168.0.4
midonet> list host
host host0 name host-a alive true
host host1 name host-b alive true
host host2 name host-c alive true
host host3 name host-d alive true
```

Check if packets are transmitted over the tunnel interface, and ensure that there are no errors or dropped packets.

Check the `tngre-overlay` port in case of GRE protocol, or the `tnvxlan-overlay` port in case of VXLAN protocol.

```
# mm-dpctl --show-dp midonet | grep overlay
Port #1 "tngre-overlay"  Gre
 Stats{rxPackets=508157678, txPackets=398704120, rxBytes=291245619484,
 txBytes=318474308439, rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
```

```
Port #2 "tnvxlan-overlay"  VXLan  Stats{rxPackets=0, txPackets=0,
 rxBytes=0, txBytes=0, rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
```

MidoNet Datapath

Check the MidoNet Datapath.

```
# mm-dpctl --show-dp midonet
Datapath name   : midonet
Datapath index : 11
Datapath Stats:
  Flows :1340066
  Hits  :1111802509
  Lost  :0
  Misses:17302163
Port #0 "midonet"  Internal  Stats{rxPackets=0, txPackets=0, rxBytes=0,
 txBytes=0, rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
Port #1 "tngre-overlay"  Gre  Stats{rxPackets=508157678,
 txPackets=398704120, rxBytes=291245619484, txBytes=318474308439,
 rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
Port #2 "tnvxlan-overlay"  VXLan  Stats{rxPackets=0, txPackets=0,
 rxBytes=0, txBytes=0, rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
Port #3 "tnvxlan-vtep"  VXLan  Stats{rxPackets=0, txPackets=0, rxBytes=0,
 txBytes=0, rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
Port #4 "tapa0164c42-dd"  NetDev  Stats{rxPackets=389426272,
 txPackets=342761506, rxBytes=1128206548338, txBytes=241007949600,
 rxErrors=0, txErrors=0, rxDropped=0, txDropped=0}
Port #5 "tap19ccc069-f1"  NetDev  Stats{rxPackets=0, txPackets=54640,
 rxBytes=0, txBytes=2347034, rxErrors=0, txErrors=0, rxDropped=0,
 txDropped=0}
Port #6 "tape3055fc6-cc"  NetDev  Stats{rxPackets=21375, txPackets=42911,
 rxBytes=3573207, txBytes=4607633, rxErrors=0, txErrors=0, rxDropped=0,
 txDropped=0}
```

```
# mm-dpctl --dump-dp midonet
1340149 flows
  Flow:
    match keys:
      Tunnel{tun_id=4360, ipv4_src=10.11.0.16, ipv4_dst=10.11.0.15,
 tun_flag=0, ipv4_tos=0, ipv4_ttl=-3}
      InPort{1}
      Ethernet{src=02:13:38:97:08:f3, dst=fa:16:3f:92:53:60}
      EtherType{0x800}
      KeyIPv4{src=8.8.8.8, dst=10.17.3.14, proto=17, tos=0, ttl=55, frag=0}
      UDP{src=53, dst=56975}
    actions:
      Output{port=21}
[...]
```

```
# mm-ctl --list-hosts
Host: id=17ef018f-de8b-431b-89f0-b5472f176769
  name=hostname
  isAlive=true
  addresses:
  vport-host-if-bindings:
    VirtualPortMapping{virtualPortId=ac0c2557-9fa0-4009-9e18-dc62ea65052a,
 localDeviceName='tapac0c2557-9f'}
    VirtualPortMapping{virtualPortId=c37d8bf2-d008-464e-a688-0627f2da342f,
 localDeviceName='f58b0880_MN_dp'}
    VirtualPortMapping{virtualPortId=7aa08012-d06c-4c78-aee8-1fff7c063fed,
 localDeviceName='tap7aa08012-d0'}
    VirtualPortMapping{virtualPortId=5aa6a752-57f2-4749-b160-9e632e0a16bb,
 localDeviceName='f58b0880_MN_dp'}
[...]
```

MTU

The MTU of VM instances has to account for the tunnel protocol's overhead to avoid fragmentation in the underlay network.

This adjusted MTU is advertised automatically by MidoNet via DHCP, but may not be applied depending on the VM's operating system being used.

Ensure that the VM's MTU is set accordingly to the underlay's MTU.

Underlay MTU	Tunnel Protocol	Protocol Overhead	VM's MTU
1500 bytes	VxLAN	50 bytes	1450 bytes
1500 bytes	GRE	46 bytes	1455 bytes
9000 bytes	VxLAN	50 bytes	8950 bytes
9000 bytes	GRE	46 bytes	8955 bytes

Topology Simulation

Topology simulation is done by the MidoNet Agent (Midolman), which retrieves the virtual topology data from the Network State Database (NSDB).

Check the `/var/log/midolman.log` file for errors or warnings.

Ensure that the connection to the NSDB works properly. The NSDB consists of two components, ZooKeeper and Cassandra.

You can verify network accessibility manually by pinging the NSDB hosts and telneting to the appropriate service ports.

Service	Port
ZooKeeper	2181
Cassandra	9042

Virtual Topology

The virtual topology is stored in the Neutron database and MidoNet's Network State Database (NSDB).

Below are some common thing to check.

Security Groups

Are the Neutron Security Groups configured to let desired traffic pass?

Check if appropriate rules for the protocols (e.g. ICMP, SSH, HTTP) and ports being used exist.

2. Common Topics

Table of Contents

MidoNet Agent

Debugging

During troubleshooting periods, you can increase the logging level via the `mn-conf(1)` utility.

The available log levels are: DEBUG, INFO, WARN, ERROR

To check the currently configured level, execute the following command:

```
$ mn-conf get agent.loggers.root
agent.loggers.root = INFO
```

To increase the log level, execute the following command:

```
$ echo "agent.loggers.root : DEBUG" | mn-conf set -t default
```

This change does not require an Agent restart.

MidoNet API

Debugging

During troubleshooting periods, you can increase the logging level via the `/usr/share/midonet-api/WEB-INF/classes/logback.xml` file.

To do so, change the root loggers level to one of the following values: DEBUG, INFO, WARN, ERROR

```
<logger name="org.midonet.event" level="DEBUG">
    <appender-ref ref="EVENT-FILE"/>
</logger>

<root level="DEBUG">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="LOG-FILE" />
</root>
```

This change does require a Tomcat restart.

Border Gateway Protocol (BGP)

This section covers some of the most common BGP related topics.

Find BGP namespace:

```
# ip netns list
mbgpX_ns
```

View interfaces in the BGP namespace:

```
# ip netns exec <mbgpX_ns> ip link show
```

Ensure BGP peers are REACHABLE on far end:

```
# ip netns exec <mbgpX_ns> ip neigh show
```

Sniff on BGP peering:

```
# ip netns exec <mbgpX_ns> tcpdump -i <ns_itf>
```

Enter Quagga's VTY shell:

```
# ip netns exec <mbgpX_ns> vtysh
```

Inside Quagga's VTY shell, show the running configuration:

```
# show run
Building configuration...

Current configuration:
!
hostname bgpd
log file /var/log/quagga/bgpd.2609.log
!
password zebra_password
!
router bgp 65535
 bgp router-id 192.168.107.29
 network 42.159.202.0/24
 neighbor 192.168.107.30 remote-as 65534
 neighbor 192.168.107.30 timers 5 15
 neighbor 192.168.107.30 timers connect 10
!
line vty
!
end
```

Inside Quagga's VTY shell, show the BGP summary:

```
# show ip bgp summary
BGP router identifier 192.168.107.29, local AS number 65535
RIB entries 19, using 2128 bytes of memory
Peers 1, using 4560 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/
PfxRcd
192.168.107.30  4 65534    5512    5523        0    0    0 07:40:09
 10
```

Inside Quagga's VTY shell, show BGP routing information:

```
# show ip bgp
BGP table version is 0, local router ID is 192.168.107.29
Status codes: s suppressed, d damped, h history, * valid, > best, i -
 internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete
```

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          192.168.107.30      0             0 65534 ?
*> 1.1.1.0/30       192.168.107.30      0             0 65534 ?
*> 42.159.202.0/24  0.0.0.0             0         32768 i
*> 192.168.0.0      192.168.107.30      0             0 65534 ?
*> 192.168.1.0      192.168.107.30      0             0 65534 ?
*> 192.168.2.0      192.168.107.30      0             0 65534 ?
*> 192.168.49.0     192.168.107.30      0             0 65534 ?
*> 192.168.107.4/30 192.168.107.30      0             0 65534 ?
*> 192.168.107.12/30 192.168.107.30     0             0 65534 ?
*> 192.168.107.20/30 192.168.107.30     0             0 65534 ?
*> 192.168.107.28/30 192.168.107.30     0             0 65534 ?

Total number of prefixes 11
```

ZooKeeper

Test if ZooKeeper is running in a non-error state. The server will respond with `imok` if it is running. Otherwise it will not respond at all.

```
$ echo ruok | nc zk-host 2181
imok
```

Lists statistics about performance and connected clients:

```
$ echo stat | nc zk-host 2181
```

Dump the contents of the ZooKeeper database into a pretty-printed text file:

```
$ zkdump -z zk-host:2181 -d -p -o zkdump.txt
```

Dump the contents of the ZooKeeper database into a machine readable JSON file:

```
$ zkdump -z zk-host:2181 -d -o zkdump.json
```

VM Interconnectivity

Scenario

VM1 can not send TCP traffic to VM2.

We want to determine how far the packet is reaching before being lost.

Determine physical compute hosts

To find out on which physical compute hosts these VMs live on, log into Horizon as administrative user and navigate to the instances page.

Find the VMs in the list and note the down the compute hosts, Internal IPs and Floating IPs:

VM1: compute1, 192.168.0.1, 172.16.0.1

VM2: compute2, 192.168.0.2, 172.16.0.2

Determine TAP interfaces

TAP interface names consists of the string "tap", followed by the first 11 characters of the VM's port UUID.

To determine the VM’s port UUID, navigate to the VM’s network in Horizon and search the port list for the VM’s internal IP. Construct the TAP interface name from it like in the following example:

Port UUID: **7aa08012-d0**6c-4c78-aee8-1fff7c063fed

TAP interface: **tap7aa08012-d0**

Examine the traffic on the TAP interfaces

In order to verify if the traffic is being seen on a VM’s virtual NIC without logging into the guest host, you can use `tcpdump` on the VM’s TAP interface on the compute host.

```
# tcpdump -n -i tap7aa08012-d0
```

Watch packet counters on the TAP interface:

```
# watch -d ip -s link show tap7aa08012-d0
```

3. Tools and Commands

Table of Contents

This section gives an overview of helpful tools and commands.

midonet-cli

The `midonet-cli` command can be run on any host which has the `python-midonet-client` package installed and connectivity to the MidoNet API.

mm-dpctl

The `mm-dpctl` command can be run on any MidoNet Agent node and will display the datapath information, such as the current flows.

Available options:

```
$ mm-dpctl
usage: mm-dpctl
    --add-dp <arg>      Add a new datapath.
    --delete-dp <arg>   Delete a datapath.
    --dump-dp <arg>     Show all the flows installed for a given datapath.
    --list-dps          List all the installed datapaths
    --show-dp <arg>     Show all the information related to a given
 datapath.
    --timeout <arg>     Specifies a timeout in seconds. If the program is
 not able to get the results in less than this amount of time it will stop
 and return with an error code
```

Examples:

```
$ mm-dpctl --show-dp midonet # shows datapath and interfaces
$ mm-dpctl --dump-dp midonet # shows current flows
```

mm-trace

The `mm-trace` command allows the MidolMan Agent to capture a particular traffic flow and to log each stage of the simulation.

It's settings are not persistent across MidolMan restarts.

Outputs are written to the `/var/log/midolman/mm-trace.log` file.

Available options:

```
$ 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
```

Examples:

```
$ 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)
```

ip

The `ip` command can be used to show / manipulate routing, devices, policy routing and tunnels.

See the man page for detailed information: http://linux.die.net/man/8/ip

List interfaces:

```
# ip link show
```

List namespaces:

```
# ip netns list
```

List interfaces within a namespace:

```
# ip netns exec namespace ip link show
```

4. Directories and Files

Table of Contents

This section gives an overview of frequently used configuration and log files.

Note that file names and paths may slightly differ depending on the used operating system and OpenStack distribution.

Cassandra

File	Type
/etc/cassandra/conf/cassandra.yaml	CONF
/var/log/cassandra/cassandra.log	LOG

MidoNet Agent

File	Type
/etc/midolman/midolman-akka.conf	CONF
/etc/midolman/midolman.conf	CONF
/etc/midolman/midolman-env.sh	CONF
/var/log/midolman/midolman.event.log	LOG
/var/log/midolman/midolman.log	LOG
/var/log/midolman/mm-trace.log	LOG
/var/log/midolman/upstart-stderr.log	LOG

MidoNet API

File	Type
/usr/share/midonet-api/WEB-INF/web.xml	CONF
/var/log/tomcat/catalina.out	LOG
/var/log/tomcat/midonet-api.log	LOG

Quagga (BGPD)

File	Type
/var/log/quagga/bgpd.log	LOG

ZooKeeper

File	Type
/etc/zookeeper/zoo.cfg	CONF

File	Type
/var/log/zookeeper/zookeeper.out	LOG

5. Processes

This section gives an overview of common processes.

Note that names and paths may slightly differ depending on the used operating system and OpenStack distribution.

Program	Process
Cassandra	java [...] org.apache.cassandra.service.CassandraDaemon
MidoNet Agent	java [...] org.midonet.midolman.Midolman
MidoNet Agent (Watchdog)	/usr/bin/python /usr/bin/wdog [...] org.midonet.midolman.Midolman
MidoNet API (Tomcat)	java [...] org.apache.catalina.startup.Bootstrap
ZooKeeper	java [...] org.apache.zookeeper.server.quorum.QuorumPeerMain