MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY


# LAB-REPORT

Report no : 04

Report Name: SDN Controllers and Mininet

Course code : ICT-3208

Course title : : Network Planning and Designing Lab.

Date of Submission : 04-September-2020

**Submitted By**

Name: Md. Nayan Ali
ID: IT-16062
3rd year 2nd semester
Session: 2015-2016
Dept. of ICT
MBSTU.

**Submitted To**

Mr. Nazrul Islam
Assistant Professor,
Dept. of ICT, MBSTU

## Objectives:

i) Install and use traffic generators as powerful tools for testing network performance.
ii) Install and configure SDN Controller.
iii) Install and understand how the mininet simulator works.
iv) Implement and run basic examples for understanding the role of the controller and how it interact with mininet.

# SDN Controllers and Mininet Lab

## Part 1: Everyday Mininet Usage

**Display Startup Options:** Type the following command "sudo mn -h" to display a help message describing Mininet's startup options:

```
--custom=CUSTOM       read custom classes or params from .py file(s)
--test=TESTS          cli|build|pingall|pingpair|iperf|all|iperfudp|none|pin
                      gpair|iperfudp|pingall|iperfUDP
-x, --xterms          spawn xterms for each node
-i IPBASE, --ipbase=IPBASE
                      base IP address for hosts
--mac                 automatically set host MACs
--arp                 set all-pairs ARP entries
-v VERBOSITY, --verbosity=VERBOSITY
                      info|warning|critical|error|debug|output
--innamespace         sw and ctrl in namespace?
--listenport=LISTENPORT
                      base port for passive switch listening
--nolistenport        don't use passive listening port
--pre=PRE             CLI script to run before tests
--post=POST           CLI script to run after tests
--pin                 pin hosts to CPU cores (requires --host cfs or --host
                      rt)
--nat                 [option=val...] adds a NAT to the topology that
                      connects Mininet hosts to the physical network.
                      Warning: This may route any traffic on the machine
                      that uses Mininet's IP subnet into the Mininet
                      network. If you need to change Mininet's IP subnet,
                      see the --ipbase option.
--version             prints the version and exits
--cluster=server1,server2...
                      run on multiple servers (experimental!)
--placement=block|random
                      node placement for --cluster (experimental!)
ininet@mininet-vm:~$
```

**Interact with Hosts and Switches:** Start a minimal topology and enter the CLI:

```
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

**i. Display Mininet CLI commands:**

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

mininet>
```

**ii. Display nodes:**

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
```

**iii. Display links:**

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

**iv. Dump information about all nodes:**

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1291>
<Host h2: h2-eth0:10.0.0.2 pid=1293>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1298>
<Controller c0: 127.0.0.1:6653 pid=1284>
mininet>
```

- If the first string typed into the Mininet CLI is a host, switch or controller name, the command is executed on that node. Run a command on a host process:

```
mininet> h1 ifconfig -a
h1-eth0   Link encap:Ethernet  HWaddr aa:9a:e5:63:da:d9
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

- In contrast, the switch by default runs in the root network namespace, so running a command on the “switch” is the same as running it from a regular terminal:

```
ovs-system Link encap:Ethernet  HWaddr 1a:cf:18:f5:ce:0d
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s1        Link encap:Ethernet  HWaddr 96:75:b3:ad:03:4b
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s1-eth1   Link encap:Ethernet  HWaddr 02:cf:8b:1b:7d:6d
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

s1-eth2   Link encap:Ethernet  HWaddr 7e:11:92:9a:0d:c0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

- For other examples highlighting that the hosts have isolated network state, run arp and route on both s1 and h1.
- It would be possible to place every host, switch and controller in its own isolated network namespace, but there's no real advantage to doing so, unless you want to replicate a complex multiple-controller network. Mininet does support this; see the --innamespace option.
- Note that only the network is virtualized; each host process sees the same set of processes and directories. For example, print the process list from a host process:

```
mininet> h1 ps -a
  PID TTY          TIME CMD
 1256 tty1     00:00:00 bash
 1276 tty1     00:00:00 sudo
 1277 tty1     00:00:00 mn
 1337 pts/2    00:00:00 controller
 1605 pts/3    00:00:00 ps
mininet>
```

- This should be the exact same as that seen by the root network namespace:

```
mininet> s1 ps -a
  PID TTY          TIME CMD
 1256 tty1     00:00:00 bash
 1276 tty1     00:00:00 sudo
 1277 tty1     00:00:00 mn
 1337 pts/2    00:00:00 controller
 1621 pts/5    00:00:00 ps
mininet>
```

**Test connectivity between hosts:** Now, verify that you can ping from host 0 to host 1:

If a string appears later in the command with a node name, that node name is replaced by its IP address; this happened for h2.

You should see OpenFlow control traffic. The first host ARPs for the MAC address of the second, which causes a packet_in message to go to the controller. The controller then sends a packet_out message to flood the broadcast packet to other ports on the switch (in this example, the only other data port). The second host sees the ARP request and sends a reply. This reply goes to the controller, which sends it to the first host and pushes down a flow entry.

Now the first host knows the MAC address of the second, and can send its ping via an ICMP Echo Request. This request, along with its corresponding reply from the second host, both go the controller and result in a flow entry pushed down (along with the actual packets getting sent out).

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.85 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.852/1.852/1.852/0.000 ms
mininet>
```

Repeat the last ping:

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.00 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.002/1.002/1.002/0.000 ms
mininet>
```

An easier way to run this test is to use the Mininet CLI built-in pingall command, which does an all-pairs ping:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

**Run a simple web server and client:** Remember that ping isn't the only command you can run on a host! Mininet hosts can run any command or application that is available to the underlying Linux system (or VM) and its file system. You can also enter any bash command, including job control (&, jobs, kill, etc.).

Next, try starting a simple HTTP server on h1, making a request from h2, then shutting down the web server:

mininet> h1 python -m SimpleHTTPServer 80 &

mininet> h2 wget -O - h1

...

```
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gitconfig">.gitconfig</a>
<li><a href=".profile">.profile</a>
<li><a href=".rnd">.rnd</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href="install-mininet-vm.sh">install-mininet-vm.sh</a>
<li><a href="loxigen/">loxigen/</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
</ul>
<hr>
</body>
</html>
100%[======================================>] 750         --.-K/s   in 0s

2020-09-03 00:14:43 (98.7 MB/s) - written to stdout [750/750]

mininet>
```

mininet> h1 kill %python

```
mininet> h1 kill %python
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [03/Sep/2020 00:14:43] "GET / HTTP/1.1" 200 -
mininet>
```

Exit the CLI:

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 2318.761 seconds
mininet@mininet-vm:~$
```

**Cleanup:** If Mininet crashes for some reason, clean it up:

```
mininet@mininet-vm:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
 ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openfl
owd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$
```

## Part 2: Advanced Startup Options

**Run a Regression Test:** You don't need to drop into the CLI; Mininet can also be used to run self-contained regression tests. Run a regression test:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.577 seconds
mininet@mininet-vm:~$ _
```

This command created a minimal topology, started up the OpenFlow reference controller, ran an all-pairs-ping test, and tore down both the topology and the controller.

Another useful test is iperf (give it about 10 seconds to complete):

```
mininet@mininet-vm:~$ sudo mn --test iperf
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['28.9 Gbits/sec', '29.0 Gbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 10.993 seconds
mininet@mininet-vm:~$
```

**Changing Topology Size and Type:** The default topology is a single switch connected to two hosts. You could change this to a different topo with --topo, and pass parameters for that topology's creation. For example, to verify all-pairs ping connectivity with one switch and three hosts:

Run a regression test:

```
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 5.483 seconds
mininet@mininet-vm:~$
```

Another example, with a linear topology (where each switch has one host, and all switches connect in a line):

```
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Waiting for switches to connect
s1 s2 s3 s4
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
c0
*** Stopping 7 links
.......
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 5.554 seconds
mininet@mininet-vm:~$
```

**Link variations:** Mininet 2.0 allows you to set link parameters, and these can even be set automatially from the command line:

```
$ sudo mn --link tc,bw=10,delay=10ms
```

```
mininet@mininet-vm:~$ sudo mn --link tc,bw=10,delay=10ms
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h1, s1) (10.00Mbit 10ms delay) (1
0.00Mbit 10ms delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay)
*** Starting CLI:
mininet>
```

```
mininet> iperf
```

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.51 Mbits/sec', '12.0 Mbits/sec']
mininet>
```

```
mininet> h1 ping -c10 h2
```

```
mininet> h1 ping -c10 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=42.0 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=41.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=42.3 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=40.8 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=41.1 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=40.9 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=41.5 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=40.4 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=41.5 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=41.2 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 40.444/41.322/42.348/0.568 ms
mininet>
```

**Adjustable Verbosity:** The default verbosity level is info, which prints what Mininet is doing during startup and teardown. Compare this with the full debug output with the -v param:

```
$ sudo mn -v debug
...
mininet> exit
```

```
*** h2 : ('ifconfig', 'h2-eth0', 'up')

added intf s1-eth2 (2) to node s1
*** s1 : ('ifconfig', 's1-eth2', 'up')
(h2, s1)
*** Configuring hosts
h1 *** h1 : ('ifconfig', 'h1-eth0', '10.0.0.1/8', 'up')
*** h1 : ('ifconfig lo up',)
h2 *** h2 : ('ifconfig', 'h2-eth0', '10.0.0.2/8', 'up')
*** h2 : ('ifconfig lo up',)

*** Starting controller
c0 *** errRun: ['which', 'controller']
/usr/local/bin/controller
  0*** c0 : ('controller -v ptcp:6653 1>/tmp/c0.log 2>/tmp/c0.log &',)

*** Starting 1 switches
s1 ...*** errRun: ovs-vsctl -- --id=@s1c0 create Controller target=\"tcp:127.0.0
.1:6653\" max_backoff=1000 -- --id=@s1-listen create Controller target=\"ptcp:66
54\" max_backoff=1000 -- --if-exists del-br s1 -- add-br s1 -- set bridge s1 con
troller=[@s1c0,@s1-listen] other_config:datapath-id=0000000000000001 fail_mode=s
ecure other-config:disable-in-band=true -- add-port s1 s1-eth1 -- set Interface
s1-eth1 ofport_request=1 -- add-port s1 s1-eth2 -- set Interface s1-eth2 ofport_
request=2
3465b793-20ce-4306-b950-df0ac62455a1
d9283f92-f6d3-49fb-a3cc-c4a69fbb7e0f
  0
*** Starting CLI:
*** errRun: ['stty', 'echo', 'sane', 'intr', '^C']
  0mininet>
```

```
$ sudo mn -v output
mininet> exit
```

```
mininet@mininet-vm:~$ sudo mn -v output
mininet> exit
mininet@mininet-vm:~$
```

**Custom Topologies:** Custom topologies can be easily defined as well, using a simple Python API, and an example is provided in custom/topo-2sw-2host.py. This example connects two switches directly, with a single host off each switch:

```
$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test
pingall
```

```
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Waiting for switches to connect
s3 s4
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s3 s4
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.708 seconds
mininet@mininet-vm:~$
```

**ID = MAC:** By default, hosts start with randomly assigned MAC addresses. This can make debugging tough, because every time the Mininet is created, the MACs change, so correlating control traffic with specific hosts is tough.The --mac option is super-useful, and sets the host MAC and IP addrs to small, unique, easy-to-read IDs.

Before:

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr be:95:6c:57:a6:f3
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

After:

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

**X-Term Display:** For more complex debugging, you can start Mininet so that it spawns one or more xterms. To start an xterm for every host and switch, pass the -x option:

```
$ sudo mn -x
```

```
mininet@mininet-vm:~$ sudo mn -x
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
Error starting terms: Cannot connect to display
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

**Other Switch Types:** Other switch types can be used. For example, to run the user-space switch:

```
$ sudo mn --switch ovsk --test iperf
```

```
mininet@mininet-vm:~$ sudo mn --switch ovsk --test iperf
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['26.5 Gbits/sec', '26.5 Gbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 10.884 seconds
mininet@mininet-vm:~$
```

**Mininet Benchmark:** To record the time to set up and tear down a topology, use test ‘none’:

```
$ sudo mn --test none
```

```
mininet@mininet-vm:~$ sudo mn --test none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.202 seconds
mininet@mininet-vm:~$
```

**Everything in its own Namespace (user switch only):** By default, the hosts are put in their own namespace, while switches and the controller are in the root namespace. To put switches in their own namespace, pass the --innamespace option:

```
$ sudo mn --innamespace --switch user
```

```
mininet@mininet-vm:~$ sudo mn --innamespace --switch user
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
c0 <-> s1
*** Testing control network
s1 -> c0
c0 -> s1
*** Results: 0% dropped (2/2 received)

*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

## Part 3: Mininet Command-Line Interface (CLI) Commands

**Display Options:** To see the list of Command-Line Interface (CLI) options, start up a minimal topology and leave it running. Build the Mininet:

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2
```

**Python Interpreter:** If the first phrase on the Mininiet command line is py, then that command is executed with Python. This might be useful for extending Mininet, as well as probing its inner workings. Each host, switch, and controller has an associated Node object. At the Mininet CLI, run:

```
mininet> py 'hello ' + 'world'
hello world
mininet>
```

Print the accessible local variables:

```
mininet> py locals()
{'h2': <Host h2: h2-eth0:10.0.0.2 pid=2026> , 'net': <mininet.net.Mininet object
 at 0x7f3c014b8190>, 'h1': <Host h1: h1-eth0:10.0.0.1 pid=2024> , 'c0': <Control
ler c0: 127.0.0.1:6653 pid=2017> , 's1': <OVSSwitch s1: lo:127.0.0.1,s1-eth1:Non
e,s1-eth2:None pid=2031> }
mininet>
```

Next, see the methods and properties available for a node, using the dir() function:

```
mininet> py dir(s1)
['IP', 'MAC', 'OVSVersion', 'TCReapply', '__class__', '__delattr__', '__dict__',
 '__doc__', '__format__', '__getattribute__', '__hash__', '__init__', '__module_
_', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__size
of__', '__str__', '__subclasshook__', '__weakref__', '_popen', '_uuids', 'addInt
f', 'argmax', 'attach', 'batch', 'batchShutdown', 'batchStartup', 'bridgeOpts',
'checkSetup', 'cleanup', 'cmd', 'cmdPrint', 'cmds', 'commands', 'config', 'confi
gDefault', 'connected', 'connectionsTo', 'controlIntf', 'controllerUUIDs', 'data
path', 'defaultDpid', 'defaultIntf', 'deleteIntfs', 'detach', 'dpctl', 'dpid', '
dpidLen', 'execed', 'failMode', 'fdToNode', 'inNamespace', 'inToNode', 'inband',
 'intf', 'intfIsUp', 'intfList', 'intfNames', 'intfOpts', 'intfs', 'isOldOVS', '
isSetup', 'lastCmd', 'lastPid', 'linkTo', 'listenPort', 'monitor', 'mountPrivate
Dirs', 'name', 'nameToIntf', 'newPort', 'opts', 'outToNode', 'params', 'pexec',
'pid', 'pollOut', 'popen', 'portBase', 'ports', 'privateDirs', 'protocols', 'rea
d', 'readbuf', 'readline', 'reconnectms', 'sendCmd', 'sendInt', 'setARP', 'setDe
faultRoute', 'setHostRoute', 'setIP', 'setMAC', 'setParam', 'setup', 'shell', 's
tart', 'startShell', 'stdin', 'stdout', 'stop', 'stp', 'terminate', 'unmountPriv
ateDirs', 'vsctl', 'waitOutput', 'waitReadable', 'waiting', 'write']
mininet>
```

You can read the on-line documentation for methods available on a node by using the help() function:

```
Help on Host in module mininet.node object:

class Host(Node)
 |  A host is simply a Node
 |
 |  Method resolution order:
 |      Host
 |      Node
 |      __builtin__.object
 |
 |  Data and other attributes defined here:
 |
 |  isSetup = True
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Node:
 |
 |  IP(self, intf=None)
 |      Return IP address of a node or specific interface.
 |
 |  MAC(self, intf=None)
 |      Return MAC address of a node or specific interface.
 |
 |  __init__(self, name, inNamespace=True, **params)
 |      name: name of node
 |      inNamespace: in network namespace?
 |      privateDirs: list of private directory strings or tuples
 |      params: Node parameters (see config() for details)
 |
:
```

You can also evaluate methods of variables:

```
mininet> py h1.IP()
10.0.0.1
mininet>
```

**Link Up/Down:** For fault tolerance testing, it can be helpful to bring links up and down.To disable both halves of a virtual ethernet pair:

```
mininet> link s1 h1 down
mininet> link s1 h1 up
mininet>
```

**XTerm Display:** To display an xterm for h1 and h2:

```
mininet> xterm h1 h2
Error: Cannot connect to display
Error: Cannot connect to display
mininet>
```

## Part 4: Python API Examples

**SSH daemon per host:** One example that may be particularly useful runs an SSH daemon on every host:

```
*** Waiting for ssh daemons to start
.
*** Hosts are running sshd at the following addresses:

h1 10.0.0.1
h2 10.0.0.2
h3 10.0.0.3
h4 10.0.0.4

*** Type 'exit' or control-D to shut down network
*** Starting CLI:
mininet>
```

```
mininet@mininet-vm:~$ ssh 10.0.0.1
The authenticity of host '10.0.0.1 (10.0.0.1)' can't be established.
ECDSA key fingerprint is bc:4d:9b:1b:ce:55:3d:0a:25:23:53:39:be:62:8e:5a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.0.1' (ECDSA) to the list of known hosts.
mininet@10.0.0.1's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Thu Sep  3 15:38:06 2020
mininet@mininet-vm:~$ _
```

**Conclusion:** In this lab, I come to learn that the walkthrough of Mininet. This walkthrough demonstrates most Mininet commands, as well as its typical usage in concert with the Wireshark dissector. Specially thanks to my course teacher who give us very much inspiration to learn this lab.