

MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY



DEPARTMENT OF ICT

ASSIGNMENT NO : 01

Course Code : ICT-3108

Course Title : Network Planning and designing Lab.

AssignmentName :Mininet Walkthrough

Submitted by

Md. Md Sohanur Islam

ID : IT-18011

Session :2017-18

Year : 3rdSemester : 1st

Submitted to

Nazrul Islam

AssistantProfessor,

Department of ICT,MBSTU

Santosh,Tangail-1902

Date of Submission :26-10-2020

Objectives : This experiment is about Install and understand how to mininet emulator works . Beside , to learn how to mininet networks creates a network of virtual hosts, switches, controllers, and links and hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking

Explanation :

Mininet: Mininet is an emulator for deploying large networks on the limited resources of a simple single Computer or Virtual Machine. Mininet has been created for enabling research in Software Defined Networking (SDN) and OpenFlow. Mininet emulator allows running unmodified code interactively on virtual hardware on a simple PC. It provides convenience and realism at very low cost. The alternative to Mininet is hardware test beds which are fast, accurate but very expensive and shared. The other option is to use simulator which is very cheap but sometimes slow and require code modification. Mininet offers ease of use, performance accuracy and scalability.

Everyday Mininet Usage:

- (1) **Display startup options:** '`sudo mn -h`' command to display a help message describing Mininet's startup options

```

--preserve-env=list      preserve specific environment variables
-e, --edit               edit files instead of running a command
-g, --group=group        run command as the specified group name or ID
-H, --set-home           set HOME variable to target user's home dir
-h, --help               display help message and exit
-h, --host=host           run command on host (if supported by plugin)
-i, --login               run login shell as the target user; a command
                        may also be specified
-K, --remove-timestamp   remove timestamp file completely
-k, --reset-timestamp    invalidate timestamp file
-l, --list                list user's privileges or check a specific
                        command; use twice for longer format
-n, --non-interactive    non-interactive mode, no prompts are used
-P, --preserve-groups     preserve group vector instead of setting to
                        target's
-p, --prompt=prompt      use the specified password prompt
-r, --role=role           create SELinux security context with specified
                        role
-S, --stdin              read password from standard input
-s, --shell               run shell as the target user; a command may
                        also be specified
-t, --type=type           create SELinux security context with specified
                        type
-T, --command-timeout=timeout terminate command after the specified time
                        limit
-U, --other-user=user     in list mode, display privileges for user
-u, --user=user           run command (or edit file) as specified user
                        name or ID
-V, --version             display version information and exit
-v, --validate            update user's timestamp without running a
                        command
--                        stop processing command line arguments
sohan18011@sohan18011-VirtualBox:~$

```

(2)Start Wireshark : To view control traffic using the OpenFlow Wireshark dissector, first open wireshark in the background

```

sohan18011@sohan18011-VirtualBox:~$ sudo wireshark &
[1] 4675
sohan18011@sohan18011-VirtualBox:~$ sudo wireshark: command not found

```

If the system you are using does not have Wireshark and the OpenFlow plugin installed, you may be able to install both of them using Mininet's install.sh script as follows:

Interact with Hosts and Switches:

Start a minimal topology and enter the CLI

```

--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!)
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>

```

1.Display Mininet CLI commands:

```

*** Starting CLI:
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>

```

2.Display nodes:

```

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

```

3.Display links:

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> _
```

4.Dump information about all nodes:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1860>
<Host h2: h2-eth0:10.0.0.2 pid=1862>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1867>
<Controller c0: 127.0.0.1:6653 pid=1853>
mininet>
```

5.Run a command on a host process:

```
mininet> h1 ifconfig -a
h1-eth0  Link encap:Ethernet  HWaddr 9a:4d:63:e7:d7: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>
```

6. Running a command on the “switch” is the same as running it from a regular terminal:

```
File Edit View Help
Mininet-VM [Running] - Oracle VM VirtualBox

ovs-system Link encap:Ethernet HWaddr 16:81:d9:14:20:a3
          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 12:c7:f9:7c:4a:45
   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 8e:05:aa:2c:9e:38
        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 66:93:b6:bd:40:86
        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>
```

7. print the process list from a host process:

```
mininet> h1 ps -a
  PID TTY          TIME CMD
 1314 tty1        00:00:00 bash
 1847 tty1        00:00:00 sudo
 1848 tty1        00:00:00 mn
 1921 pts/1      00:00:00 controller
 2203 pts/2      00:00:00 ps
mininet>
```


8. Host processes seen by the root network namespace:

```
mininet> s1 ps -a
  PID TTY          TIME CMD
 1314 tty1        00:00:00 bash
 1847 tty1        00:00:00 sudo
 1848 tty1        00:00:00 mn
 1921 pts/1        00:00:00 controller
 2221 pts/5        00:00:00 ps
mininet>
```

9. Now, verify that you can ping from host 0 to host 1

```
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=4.15 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.153/4.153/4.153/0.000 ms
mininet>
```

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

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

11. 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> h1 kill %python
Traceback (most recent call last):
  File "/usr/lib/python2.7/runpy.py", line 162, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "/usr/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 230, in <module>
    test()
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 226, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 595, in test
    httpd = ServerClass(server_address, HandlerClass)
  File "/usr/lib/python2.7/SocketServer.py", line 419, in __init__
    self.server_bind()
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 108, in server_bind
    SocketServer.TCPServer.server_bind(self)
  File "/usr/lib/python2.7/SocketServer.py", line 430, in server_bind
    self.socket.bind(self.server_address)
  File "/usr/lib/python2.7/socket.py", line 224, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 98] Address already in use
bash: kill: python: ambiguous job spec
mininet>

```

SocketServer.TCPServer.server_bind(self)

File "/usr/lib/python2.7/SocketServer.py", line 434, in server_bind

Create, edit and sign PDF

Advanced Startup Options:

A. Run a regression test:

```
ovs-vsctl --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_\.[:alnum:]]+-eth[[:digit:]]+)'
( ip link del s1-eth1; ip link del s1-eth2 ) 2> /dev/null
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:~$ sudo mn --test plngpalr
*** 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
```

```
Mininet-VM [Running] - Oracle VM VirtualBox
AttributeError: 'Mininet' object has no attribute 'plngpalr'

*** 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 --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_[:alnum:]]+-eth[[:digit:]]+)'
( ip link del s1-eth1; ip link del s1-eth2 ) 2> /dev/null
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:~$ sudo mn --test plngpalr_
```

B. Mininet 2.0 allows you to set link parameters, and these can even be set automatically from the command line

```
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=1.62 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3.14 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.303 ms  
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.074 ms  
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.120 ms  
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.116 ms  
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=1.17 ms  
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.044 ms  
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.044 ms  
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.036 ms  
  
--- 10.0.0.2 ping statistics ---  
10 packets transmitted, 10 received, 0% packet loss, time 9002ms  
rtt min/avg/max/mdev = 0.036/0.668/3.149/0.980 ms  
mininet>
```

C.The --mac option is super-useful, and sets the host MAC and IP addrs to small, unique, easy-to-read IDs

```
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr c2:81:79:a1:c4:07
         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:14 errors:0 dropped:0 overruns:0 frame:0
         TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:1204 (1.2 KB)  TX bytes:1204 (1.2 KB)

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> _
```

D.The iperf-reported TCP bandwidth should be similar to the OpenFlow kernel module, and possibly faster

```
Mininet-VM [Running] - Oracle VM VirtualBox
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: ['10.1 Gbits/sec', '10.1 Gbits/sec']
**** Stopping 1 controllers
c0
**** Stopping 2 links
..
**** Stopping 1 switches
s1
**** Stopping 2 hosts
h1 h2
**** Done
completed in 11.284 seconds
mininet@mininet-vm:~$
```

F.To record the time to set up and tear down a topology, use test 'none'


```
mininet@mininet-vm:~$ sudo mn --switch ovsk --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 0.515 seconds
mininet@mininet-vm:~$ _
```

G.To put switches in their own namespace, pass the innamespace option:

```

completed in 11.201 seconds
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> _

```

H. At the Mininet CLI, run:

```

*** Starting CLI:
mininet> py 'hello' + ' ' + 'sohan-it18011'
hello sohan-it18011
mininet>

```

I.. Print the accessible local variables

```

*** Starting CLI:
mininet> py 'hello' + ' ' + 'sohan-it18011'
hello sohan-it18011
mininet> py locals()
{'h2': <Host h2: h2-eth0:10.0.0.2 pid=2888> , 'net': <mininet.net.MininetWithCon
trolNet object at 0x7f89e6515190>, 'h1': <Host h1: h1-eth0:10.0.0.1 pid=2886> ,
'c0': <Controller c0: 192.168.123.1:6653 pid=2879> , 's1': <UserSwitch s1: s1-et
h0:192.168.123.2,s1-eth1:None,s1-eth2:None pid=2890> }
mininet>
mininet>

```

J. I can also evaluate methods of variables:

```

mininet> py h1.IP()
10.0.0.1
mininet>

```

Conclusion:

This is the command of mininet walkthrough. I have known that the walkthrough of mininet, from this labM. I install mininet in my virtual box then I run the command. This is very interesting for me for the first time. I take help from my senior brothers to complete my work. I also know that how to create virtual hosts, switches, controllers, and link. Mininet supports research, development, learning, testing, and many other tasks. Alhamdulillah, I completed my work without any problem.