Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Report No: 01
Report Name: Mininet Walkthrough
Course Title: Network Planning and designing Lab.

Course Code: ICT-3208

Submitted By	Submitted To
Name: **Zafrul Hasan Khan** ID: **IT-18003** Session: 2017-18 3rd Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	**Nazrul Islam** **Assistant Professor,** Dept. of Information & Communication Technology, MBSTU.

Submission Date: 26-10-2020

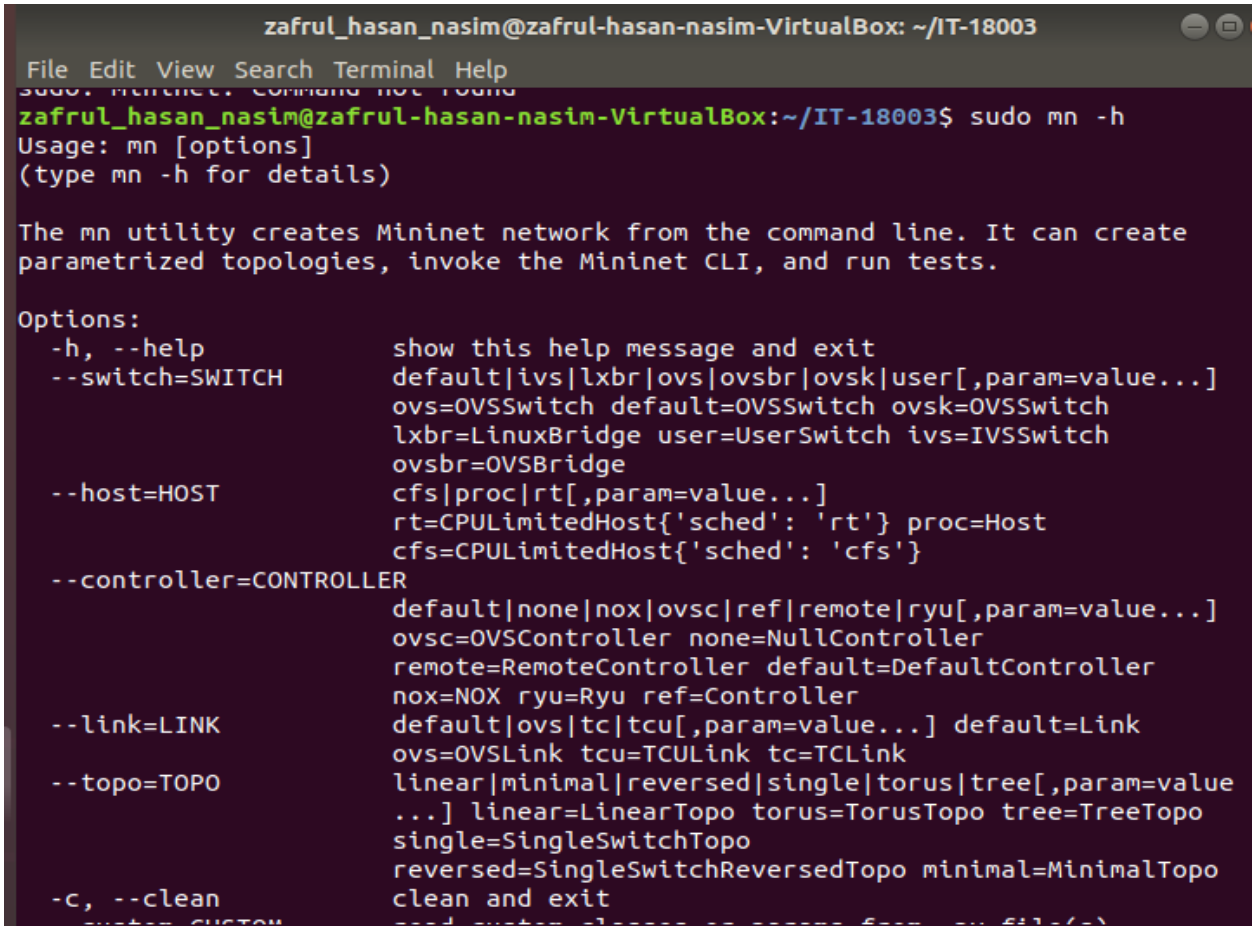
Objectives : Install and understand how to mininet emulator works . Beside , to leran how to mininet networks creates a network of virtual hosts, switches, controllers, and links and hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined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

Explanation :

Mininet: Mininet is a *network emulator* which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Everyday Mininet Usage

Display startup options: 'sudo mn -h' command to display a help message describing Mininet's startup options.



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

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~/IT-18003$ sudo wireshark &
[1] 19599
```

1f 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:

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~/IT-18003$ cd ~
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ cd ~
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ git clone https://github.co
m/mininet/mininet
Cloning into 'mininet'...
remote: Enumerating objects: 9752, done.
remote: Total 9752 (delta 0), reused 0 (delta 0), pack-reused 9752
Receiving objects: 100% (9752/9752), 3.03 MiB | 41.00 KiB/s, done.
Resolving deltas: 100% (6470/6470), done.
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ mininet/util/install.sh -w
Detected Linux distribution: Ubuntu 18.04 bionic amd64
sys.version_info(major=2, minor=7, micro=17, releaselevel='final', serial=0)
Detected Python (python) version 2
Installing Wireshark
[sudo] password for zafrul_hasan_nasim:
Reading package lists...
Building dependency tree...
Reading state information...
The following packages were automatically installed and are no longer required:
  efibootmgr gir1.2-geocodeglib-1.0 libegl1-mesa libfwup1 libllvm8
  libwayland-egl1-mesa ubuntu-web-launchers
```

Interact with Hosts and Switches:

Start a minimal topology and enter the CLI

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~/mininet$ sudo mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

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>
```

Display nodes:

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

Display links:

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

Dump information about all nodes:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=22626>
<Host h2: h2-eth0:10.0.0.2 pid=22628>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=22633>
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::9412:55ff:fe70:ea67  prefixlen 64  scopeid 0x20<link>
        ether 96:12:55:70:ea:67  txqueuelen 1000  (Ethernet)
        RX packets 33  bytes 3735 (3.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

Run a command on a host process:

```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::9412:55ff:fe70:ea67  prefixlen 64  scopeid 0x20<link>
        ether 96:12:55:70:ea:67  txqueuelen 1000  (Ethernet)
        RX packets 33  bytes 3735 (3.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

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

```
mininet> s1 ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::153b:cca1:e7bc:dd1c  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:44:19:2e  txqueuelen 1000  (Ethernet)
        RX packets 44368  bytes 59205949 (59.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22499  bytes 1483760 (1.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 951  bytes 86810 (86.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 951  bytes 86810 (86.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 72:a0:56:ba:3c:cf  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
```

print the process list from a host process:

```
mininet> h1 ps -a
  PID TTY          TIME CMD
  860 tty1     00:00:40 Xorg
  884 tty1     00:00:00 gnome-session-b
 1008 tty1     00:01:51 gnome-shell
 1056 tty1     00:00:07 ibus-daemon
 1061 tty1     00:00:00 ibus-dconf
 1066 tty1     00:00:00 ibus-x11
 1140 tty1     00:00:00 gsd-power
 1142 tty1     00:00:00 gsd-print-notif
 1144 tty1     00:00:00 gsd-rfkill
 1145 tty1     00:00:00 gsd-screensaver
 1147 tty1     00:00:00 gsd-sharing
 1151 tty1     00:00:00 gsd-sound
 1158 tty1     00:00:00 gsd-xsettings
 1167 tty1     00:00:00 gsd-wacom
 1170 tty1     00:00:00 gsd-smartcard
 1182 tty1     00:00:00 gsd-clipboard
 1183 tty1     00:00:00 gsd-a11y-settin
 1184 tty1     00:00:00 gsd-datetime
 1188 tty1     00:00:01 gsd-color
 1192 tty1     00:00:00 gsd-keyboard
```

Host processseen by the root network namespace:

```
mininet> s1 ps -a
  PID TTY          TIME CMD
  860 tty1     00:00:41 Xorg
  884 tty1     00:00:00 gnome-session-b
 1008 tty1     00:01:52 gnome-shell
 1056 tty1     00:00:07 ibus-daemon
 1061 tty1     00:00:00 ibus-dconf
 1066 tty1     00:00:00 ibus-x11
 1140 tty1     00:00:00 gsd-power
 1142 tty1     00:00:00 gsd-print-notif
 1144 tty1     00:00:00 gsd-rfkill
 1145 tty1     00:00:00 gsd-screensaver
 1147 tty1     00:00:00 gsd-sharing
 1151 tty1     00:00:00 gsd-sound
 1158 tty1     00:00:00 gsd-xsettings
 1167 tty1     00:00:00 gsd-wacom
 1170 tty1     00:00:00 gsd-smartcard
 1182 tty1     00:00:00 gsd-clipboard
 1183 tty1     00:00:00 gsd-a11y-settin
 1184 tty1     00:00:00 gsd-datetime
 1188 tty1     00:00:01 gsd-color
 1192 tty1     00:00:00 gsd-keyboard
 1193 tty1     00:00:00 gsd-housekeepin
 1195 tty1     00:00:00 gsd-mouse
```

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=0.365 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.365/0.365/0.365/0.000 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 me

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

:

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 &
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [25/Oct/2020 12:58:17] "GET / HTTP/1.1" 200 -
mininet> h2 wget -o - h1
--2020-10-25 12:58:39--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 920 [text/html]
Saving to: ‘index.html.1’

     0K                                                       100%  133M=0s

2020-10-25 12:58:39 (133 MB/s) - ‘index.html.1’ saved [920/920]

mininet>
```

```
mininet> h1 kill %python
Traceback (most recent call last):
  File "/usr/lib/python2.7/runpy.py", line 174, 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 235, in <module>
    test()
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 231, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 606, in test
    httpd = ServerClass(server_address, HandlerClass)
  File "/usr/lib/python2.7/SocketServer.py", line 420, 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 434, in server_bind
    self.socket.bind(self.server_address)
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 98] Address already in use
10.0.0.2 - - [25/Oct/2020 12:58:39] "GET / HTTP/1.1" 200 -
bash: kill: python: ambiguous job spec
```

Advanced Startup Options:

Run a regression test:

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ sudo mn --test pingpair
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
```

```
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=0.748 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.113 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=0.130 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.142 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.109 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9217ms
rtt min/avg/max/mdev = 0.103/0.182/0.748/0.188 ms
mininet>
```

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

```
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=0.748 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.113 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=0.130 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.142 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.109 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9217ms
rtt min/avg/max/mdev = 0.103/0.182/0.748/0.188 ms
mininet>
```

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: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::2c26:4dff:feda:90a7  prefixlen 64  scopeid 0x20<link>
        ether 2e:26:4d:da:90:a7  txqueuelen 1000  (Ethernet)
        RX packets 44  bytes 4712 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 1860 (1.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

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

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ sudo mn --switch ovsk --tes
t iperf
*** ------------------------------------------------------------------------
-
Caught exception. Cleaning up...

Exception: Could not find a default controller for switch ovsk
*** ------------------------------------------------------------------------
-
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow
d ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openf
lowd 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
```

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

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ sudo mn --test none
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.266 seconds
```

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

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ sudo mn --innamespace --swi
tch user
-------------------------------------------------------------------------------
-
Caught exception. Cleaning up...

Exception: Could not find a default controller for switch user
-------------------------------------------------------------------------------
-
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow
d ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openf
lowd 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
```

At the Mininet CLI, run:

```
mininet> py 'hello' + ' ' + 'nasim'
hello nasim
mininet>
```

Print the accessible local variables

```
mininet> py locals()
{'h2': <Host h2: h2-eth0:10.0.0.2 pid=2302> , 'net': <mininet.net.Mininet objec
t at 0x7f25176f0690>, 'h1': <Host h1: h1-eth0:10.0.0.1 pid=2300> , 's1': <OVSBr
idge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=2307> }
mininet>
```

I can also evaluate methods of variables:

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

In one window, and in another window, start up Mininet to connect to the “remote” controller (which is actually running locally, but outside of Mininet’s control):

```
zafrul_hasan_nasim@zafrul-hasan-nasim-VirtualBox:~$ sudo mn --controller=remote
 ,ip=10.0.0.1,port =6633
Usage: mn [options]
(type mn -h for details)

The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help            show this help message and exit
  --switch=SWITCH       default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
                        ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch
                        lxbr=LinuxBridge user=UserSwitch ivs=IVSSwitch
                        ovsbr=OVSBridge
  --host=HOST           cfs|proc|rt[,param=value...]
                        rt=CPULimitedHost{'sched': 'rt'} proc=Host
                        cfs=CPULimitedHost{'sched': 'cfs'}
  --controller=CONTROLLER
                        default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ovsc=OVSController none=NullController
                        remote=RemoteController default=DefaultController
                        nox=NOX ryu=Ryu ref=Controller
  --link=LINK           default|ovs|tc|tcu[,param=value...] default=Link
                        ovs=OVSLink tcu=TCULink tc=TCLink
  --topo=TOPO           linear|minimal|reversed|single|torus|tree[,param=value
                        ...] linear=LinearTopo torus=TorusTopo tree=TreeTopo
                        single=SingleSwitchTopo
                        reversed=SingleSwitchReversedTopo minimal=MinimalTopo
```

Conclusion: From this lab , I have known that the walkthrough of mininet . I also known that mininet how to creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. In future , Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks.