CCN EXPERIMENT 10

UID: 2022300057

Name:Piyush Mehta

CLASS: Comps-A

Batch: A-4

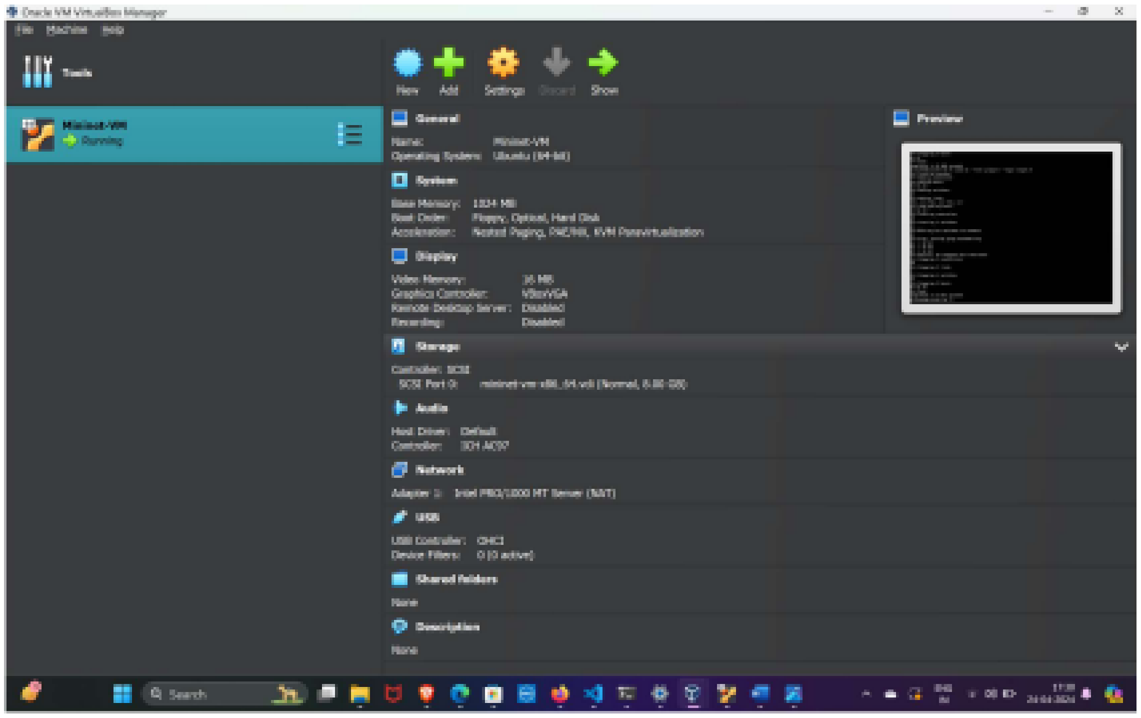
Aim: Seng up a Virtual Network using Mininet

Objecve:

The objecve of this lab exercise is to create a realisc virtual network using Mininet, a tool for emulang network environments. By the end of this exercise, students should be able to set up a virtual network, run real kernel, switch, and applicaon code, and understand the basic workflow of Mininet.

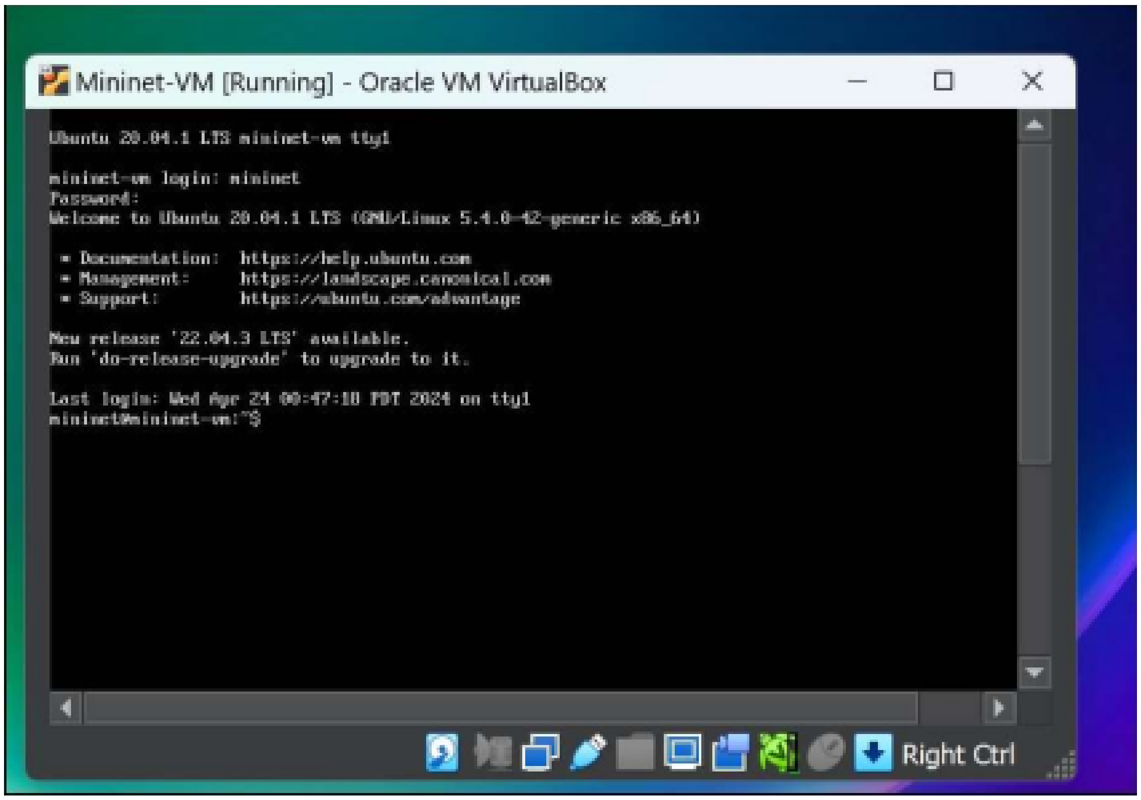
Step 1: Installaon and setup of Mininet and running it in Virtual

Box. A] Installaon



```
General
Name:               Mininet-VM
Operating System:   Ubuntu (64-bit)

System
Base Memory:  1024 MB
Boot Order:   Floppy, Optical, Hard Disk
Acceleration: Nested Paging, PAE/NX, KVM Paravirtualization

Display
Video Memory:            16 MB
Graphics Controller:     VBoxVGA
Remote Desktop Server:   Disabled
Recording:               Disabled

Storage
Controller: SCSI
 SCSI Port 0:    mininet-vm-x86_64.vdi (Normal, 8.00 GB)

Audio
Host Driver:  Default
Controller:   ICH AC97

Network
Adapter 1:  Intel PRO/1000 MT Server (NAT)

USB
USB Controller:  OHCI
Device Filters:  0 (0 active)

Shared folders
None

Description
None
```

B] Login



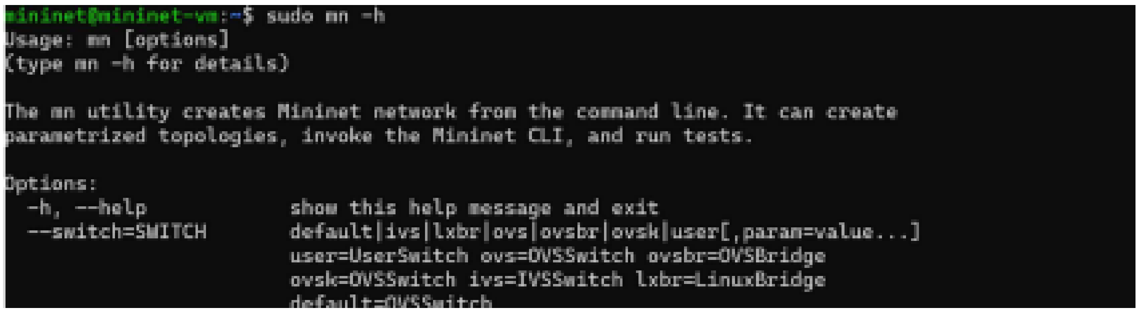
```
Ubuntu 20.04.1 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Apr 24 00:47:18 PDT 2024 on tty1
mininet@mininet-vm:~$
```

Step 2: Walkthrough

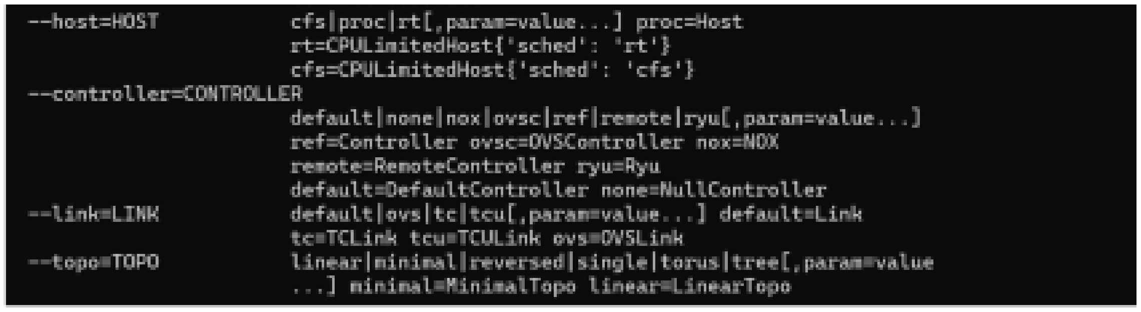
Part 1: Everyday Mininet Usage

Display Startup Options

$ sudo mn -h



```
mininet@mininet-vm:~$ sudo mn -h
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...]
                        user=UserSwitch ovs=OVSSwitch ovsbr=OVSBridge
                        ovsk=OVSSwitch ivs=IVSSwitch lxbr=LinuxBridge
                        default=OVSSwitch
```

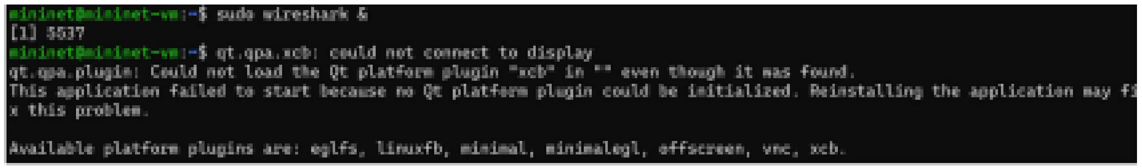


```
  --host=HOST           cfs|proc|rt[,param=value...] proc=Host
                        rt=CPULimitedHost{'sched': 'rt'}
                        cfs=CPULimitedHost{'sched': 'cfs'}
  --controller=CONTROLLER
                        default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ref=Controller ovsc=OVSController nox=NOX
                        remote=RemoteController ryu=Ryu
                        default=DefaultController none=NullController
  --link=LINK           default|ovs|tc|tcu[,param=value...] default=Link
                        tc=TCLink tcu=TCULink ovs=OVSLink
  --topo=TOPO           linear|minimal|reversed|single|torus|tree[,param=value
                        ...] minimal=MinimalTopo linear=LinearTopo
```

Start Wireshark

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

$ sudo wireshark &



```
mininet@mininet-vm:~$ sudo wireshark &
[1] 5537
mininet@mininet-vm:~$ qt.qpa.xcb: could not connect to display
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found.
This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fi
x this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc, xcb.
```

Fixing error: “Could not load the Qt platform plugin "xcb"”

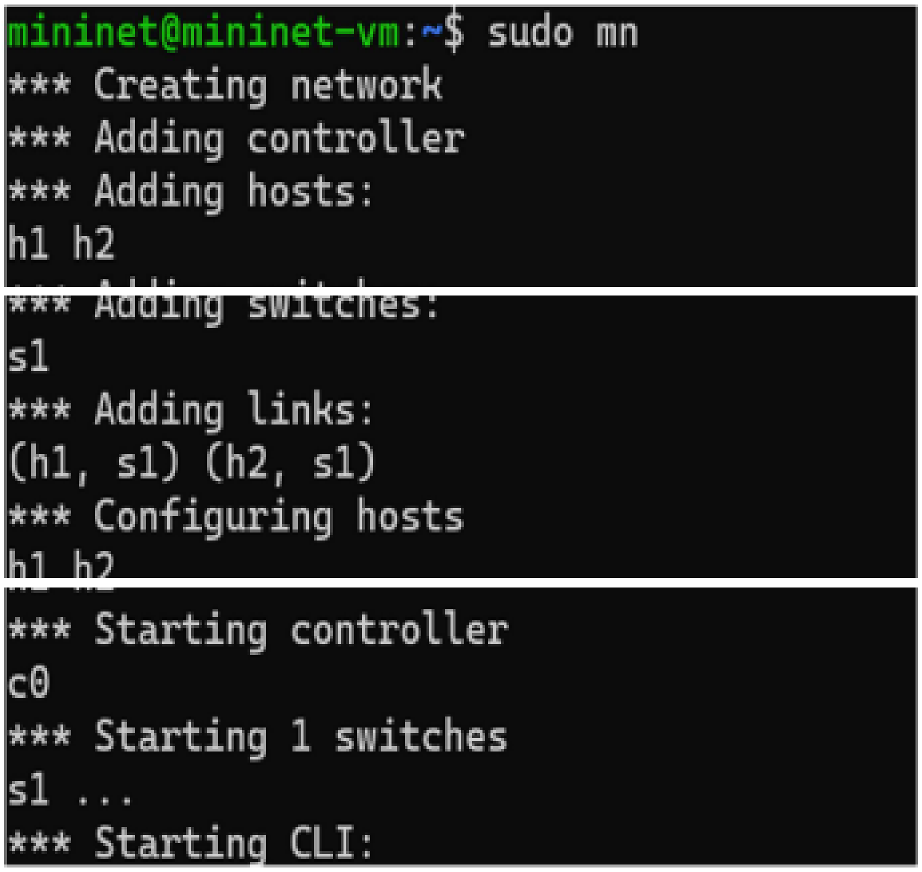
$ dpkg -l | grep libdouble-conversion # to see which version you have $ sudo apt remove libdouble-conversion3 # be sure to specify the right version
$ sudo apt autoremove
$ sudo apt install wireshark

Still error persists.

Interact with Hosts and Switches

Start a minimal topology and enter the CLI:

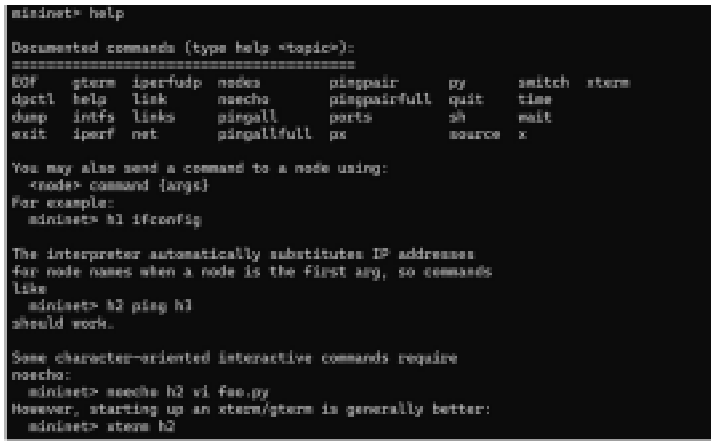
$ sudo mn



```
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:
```

Display Mininet CLI commands:

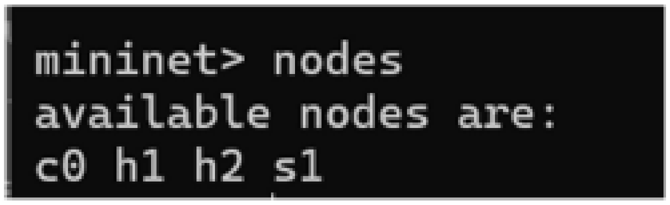
mininet> help



```
mininet> help

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

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
```

Display nodes:

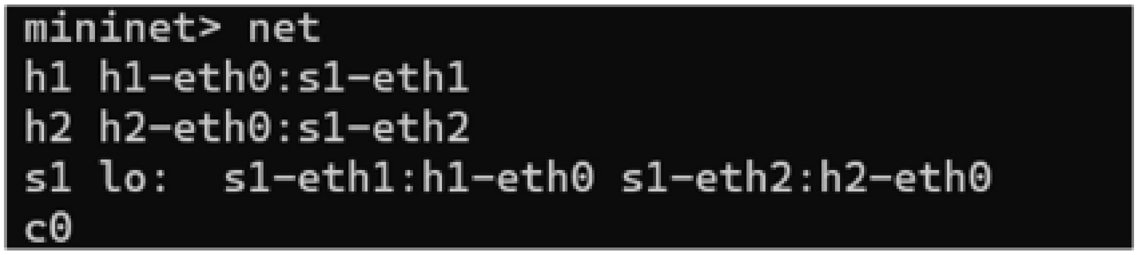
mininet> nodes



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

Display links:

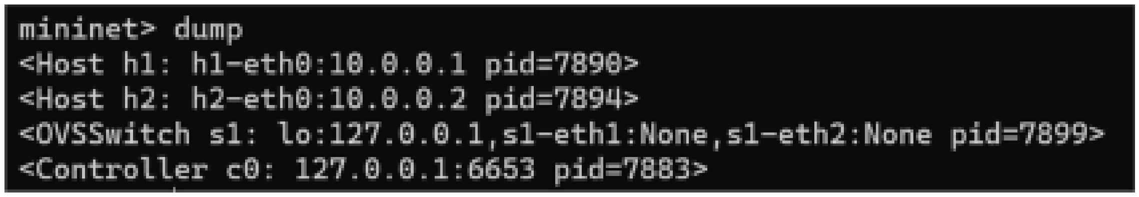
mininet> net



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

Dump information about all nodes:

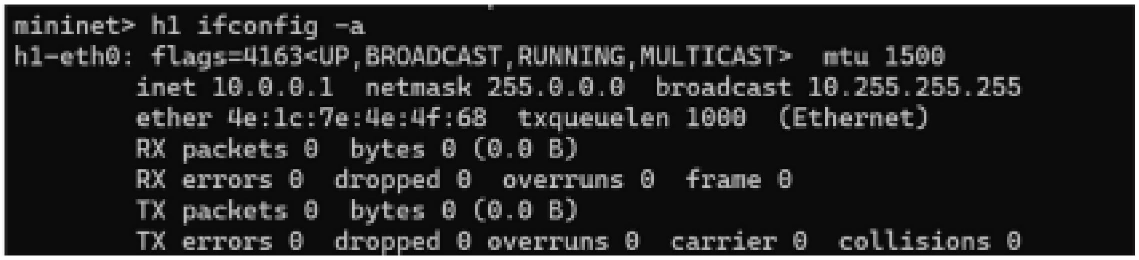
mininet> dump



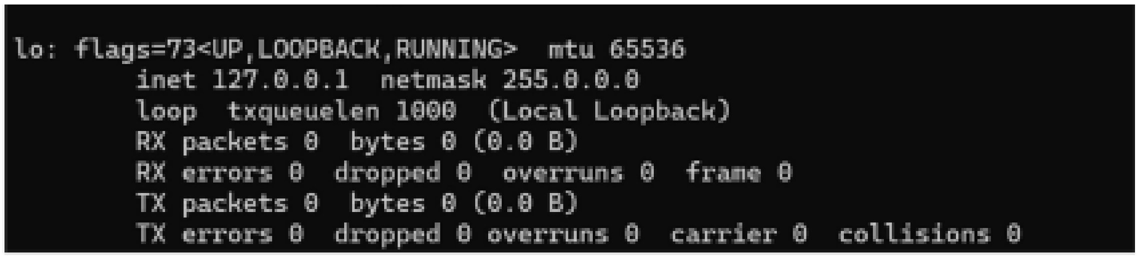
```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=7890>
<Host h2: h2-eth0:10.0.0.2 pid=7894>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=7899>
<Controller c0: 127.0.0.1:6653 pid=7883>
```

Run command on host:

mininet> h1 ifconfig -a



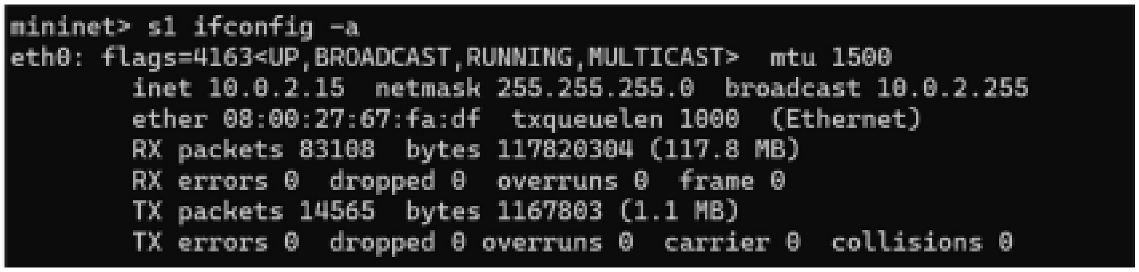
```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 4e:1c:7e:4e:4f:68  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        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
```

Run command on switch:

mininet> s1 ifconfig -a



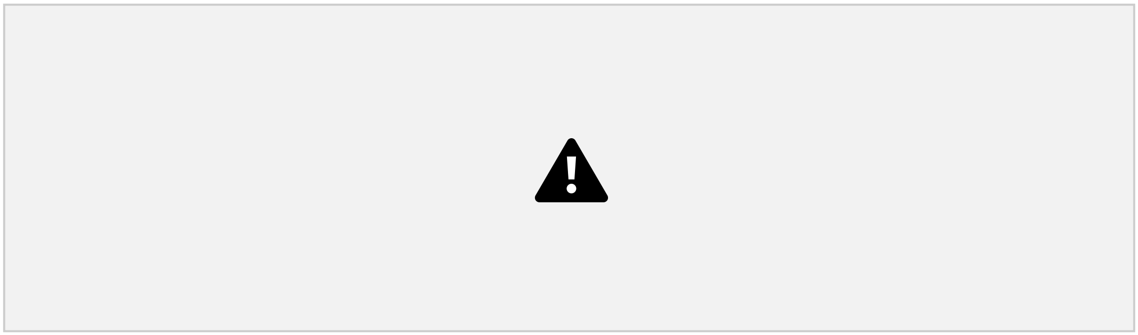
```
mininet> s1 ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 08:00:27:67:fa:df  txqueuelen 1000  (Ethernet)
        RX packets 83108  bytes 117820304 (117.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14565  bytes 1167803 (1.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Print the process list from a host process:

```
mininet> h1 ps -a
```

For switch:

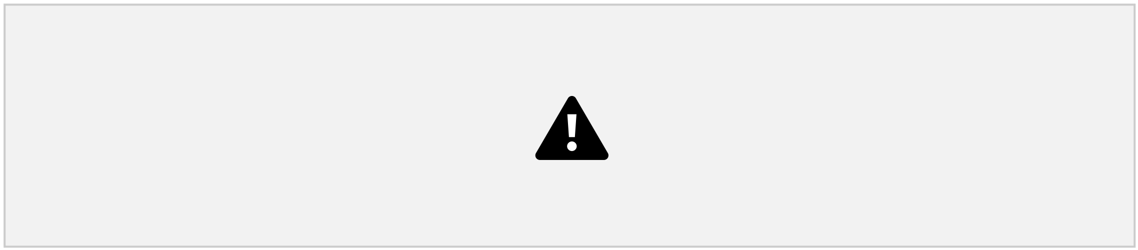
```
mininet> s1 ps -a
```

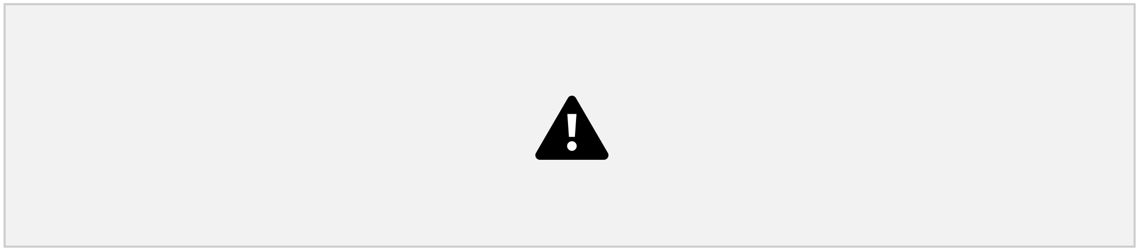


Test connectivity between hosts

Ping from host 0 to host 1:

```
mininet> h1 ping -c 1 h2
```



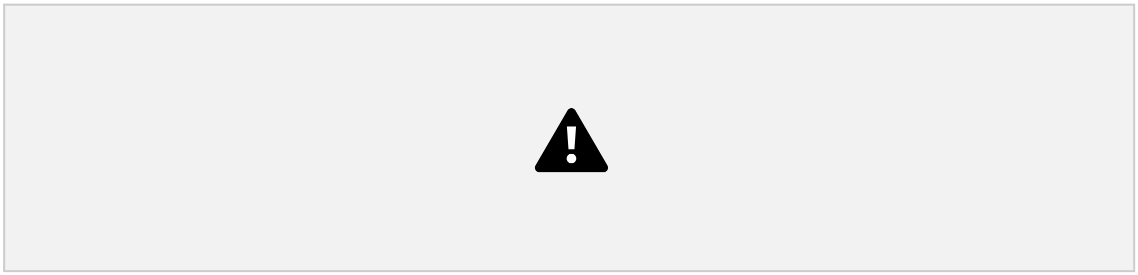


```
mininet> pingall
```

Run a simple web server and client

Start a simple HTTP server on `h1`, making a request from `h2`, then shut down the web server:

```
mininet> h1 python -m http.server 80 &
mininet> h2 wget -O - h1
```



Close Server
mininet> h1 kill %python

Exit the CLI:

```
mininet> exit
```

Cleanup

If Mininet crashes for some reason, clean it up:

```
$ sudo mn -c
```

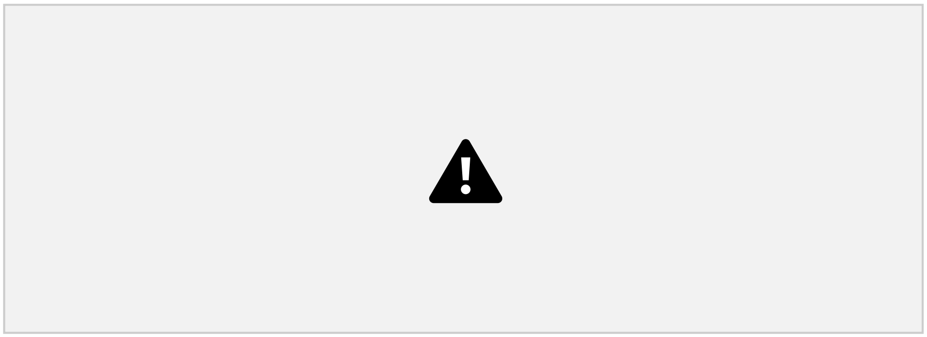
Not needed as it did not crash

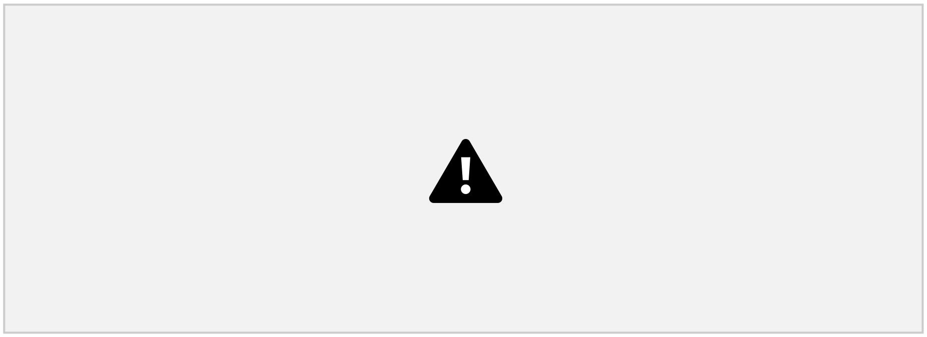
Part 2: Advanced Startup Opons

Run a Regression Test

```
$ sudo mn --test pingpair
```

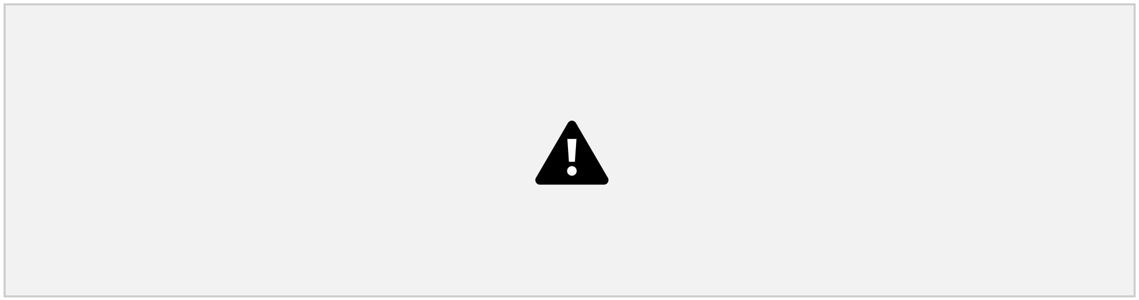
This command created a minimal topology, started up the OpenFlow reference controller, ran an all-pairs-ping test, and tore down both the topology and the controller.

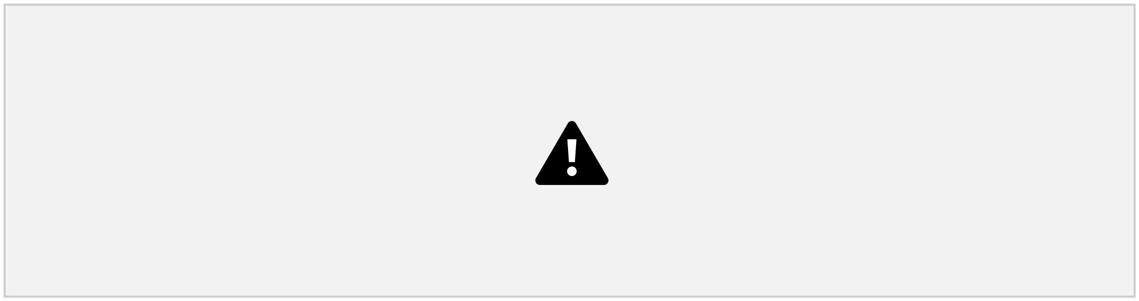




```
$ sudo mn --test iperf
```

This command created the same Mininet, ran an iperf server on one host, ran an iperf client on the second host, and parsed the bandwidth achieved.

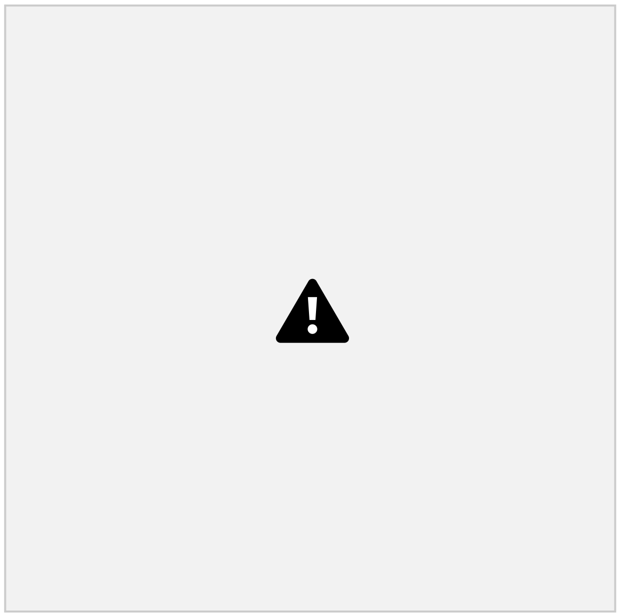




Changing Topology Size and Type

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

```
$ sudo mn --test pingall --topo single,3
```



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

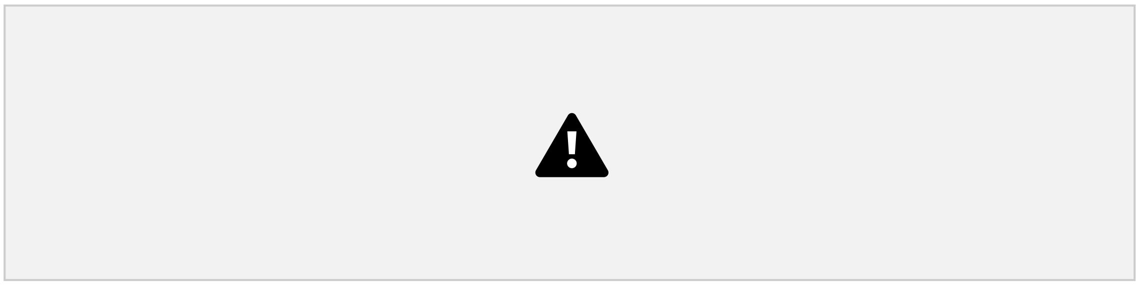
```
$ sudo mn --test pingall --topo linear,4
```

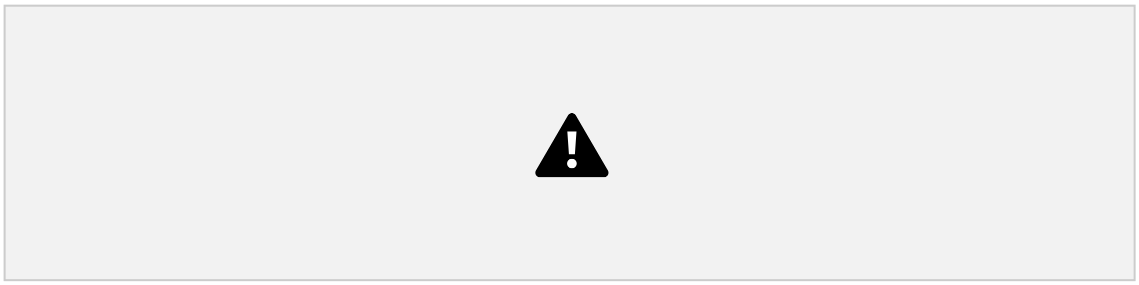
Link variations

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

If the delay for each link is 10 ms, the round trip time (RTT) should be about 40 ms, since the ICMP request traverses two links (one to the switch, one to the destination) and the ICMP reply traverses two links coming back.





Adjustable Verbosity

The default verbosity level is info, which prints what Mininet is doing during startup and teardown. Verbosity can be adjusted by using debug, output etc type.

```
$ sudo mn -v debug
```

Part 3: Mininet Command-Line Interface (CLI)

Commands 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'
```

Print the accessible local variables:

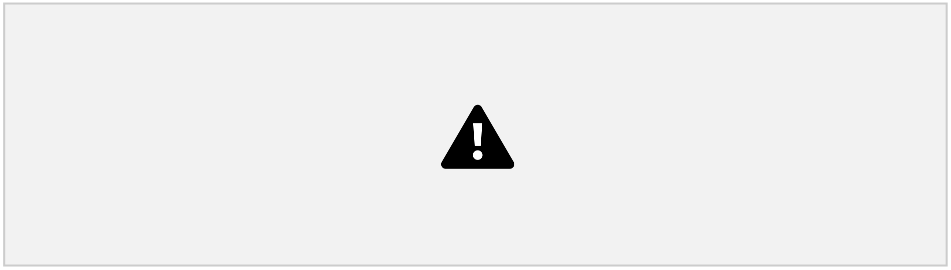
```
mininet> py locals()
```

Next, see the methods and properties available for a node, using the dir() function: `mininet> py dir(s1)`

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:

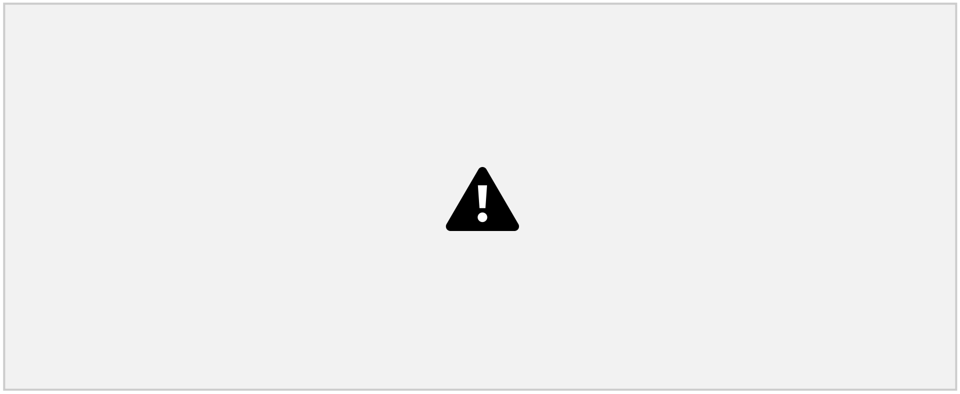
```
mininet> link s1 h1 down and mininet> link s1 h1 up
```

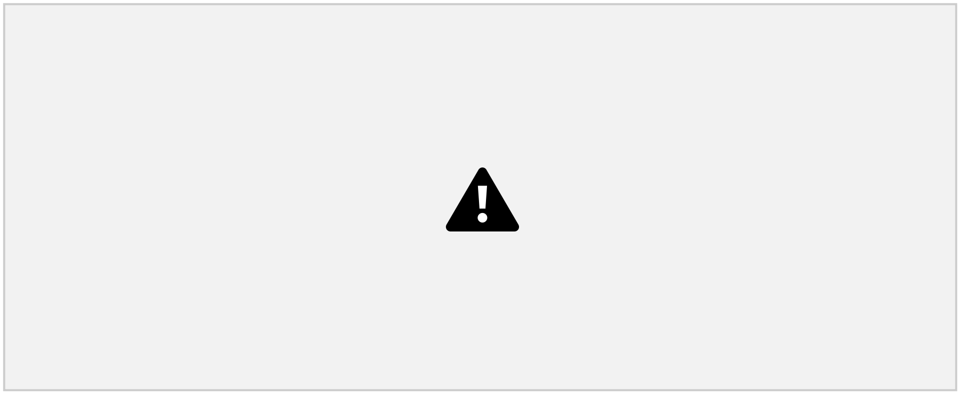


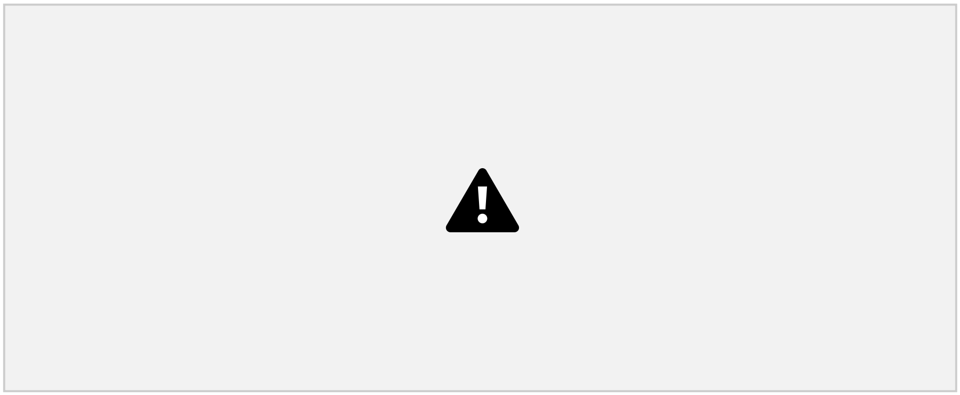
XTerm Display

To display an xterm for h1 and h2:

```
mininet> xterm h1 h2
```







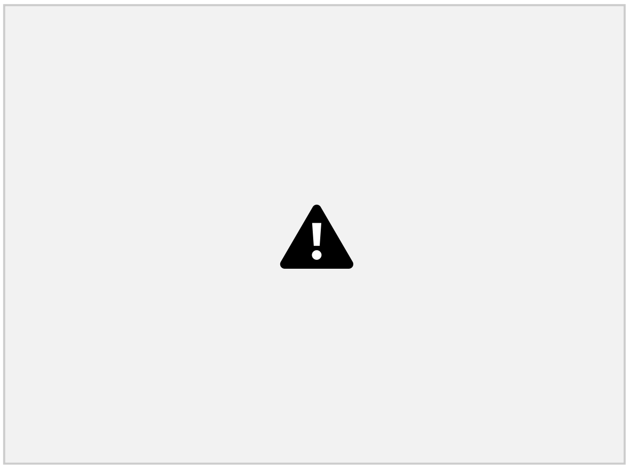
Part 4: Python API Examples

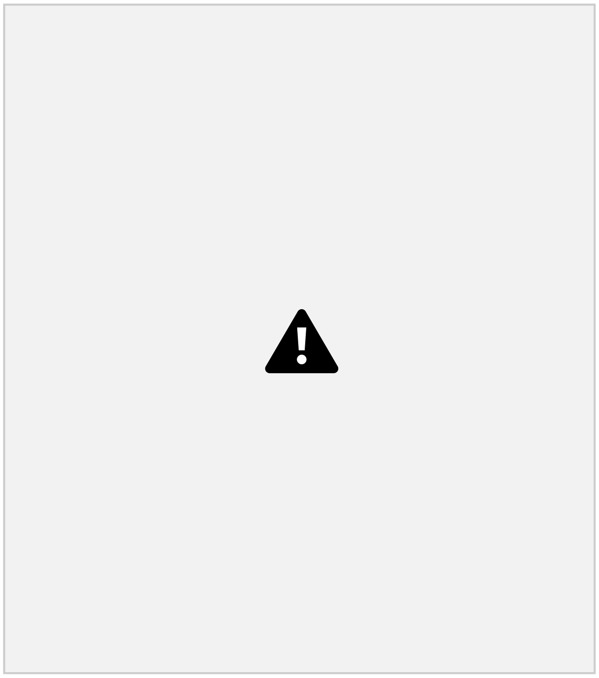
SSH daemon per host

```
$ sudo ~/mininet/examples/sshd.py
```

From another terminal, you can ssh into any host and run interactive commands:

```
$ ssh 10.0.0.1
$ ping 10.0.0.2
...
$ exit
```





CONCLUSION: I this experiment I learnt how to use Mininet to generate different topologies, customize topologies and also establish communicaon between different elements of the topology. Furthermore, I learnt to use Python API and also establish ssh connecon with another remote server running Mininet.

I also learnt to run tests like regression test to check connecvity between components of a topology. I did face difficulty in running Wireshark from Mininet as an error persisted even aer several tries to overcome it.