Subject	**Computer Communications and Network**
Name	**Krrish Nichanii**
UID no.	**2022300069 , Batch-D**
Experiment No.	**10**
Date of Submission	**25-04-2024**

AIM :	The objective of this lab exercise is to create a realistic virtual network using Mininet, a tool for emulating network environments.

PROCEDURE:	PART 1 : EVERYDAY MININET USAGE
	
	i. Command : sudo mn
	Ouput:
	`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>`
	ii. Command : help
	Output:

```
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

mininet>
```

iii. Command : nodes

Output:

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

iv. Command : net

Output:

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

v. Command : a) net.addLink(h1,s1) b) net.addLink(h2,s2)

Output:

```
mininet> net.addLink(h1,s1)
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> net.addLink(h2,s2)
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

vi. Command : dump = dump information about all nodes

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=16441>
<Host h2: h2-eth0:10.0.0.2 pid=16445>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=16450>
<Controller c0: 127.0.0.1:6653 pid=16434>
mininet>
```

vii) Command : h1 ifconfig -a : Run ifconfig on host “h1”

```
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 c2:eb:82:8b:d0:cc  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

mininet> _
```

viii. s1 ifconfig -a

Output:

```
s1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 06:8c:df:08:b1:4f  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

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether ea:bc:fd:a6:3f:f0  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

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether c6:24:9e:a2:82:48  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

sw0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether fa:82:b7:82:a2:42  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

mininet>
```

ix. h1 ps -a : Print the list of all process of a host

```
mininet> h1 ps -a
    PID TTY          TIME CMD
  16380 tty1     00:00:00 bash
  16427 tty1     00:00:00 sudo
  16429 tty1     00:00:01 mn
  16485 pts/0    00:00:00 controller
  16540 pts/1    00:00:00 ps
mininet>
```

x. s1 ps -a: Print the list of processes as seen by root network namespace

```
mininet> s1 ps -a
    PID TTY          TIME CMD
  16380 tty1     00:00:00 bash
  16427 tty1     00:00:00 sudo
  16429 tty1     00:00:01 mn
  16485 pts/0    00:00:00 controller
  16542 pts/3    00:00:00 ps
mininet> _
```

Note : It is exactly same as the host process.

xi. h1 ping -c 1 h2

```
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=14.7 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.739/14.739/14.739/0.000 ms
mininet> _
```

xii. pingall

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

xiii. Starting a simple HTTP server on h1, making a request from h2, and then shutting down the web server.

Starting Webserver command : h1 python -m http.server 80 &

Making a request command: h2 wget -O – h1

Shutting down the web server: h1 kill %python

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso8859-1">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a></li>
<li><a href=".bash_logout">.bash_logout</a></li>
<li><a href=".bashrc">.bashrc</a></li>
<li><a href=".cache/">.cache/</a></li>
<li><a href=".gitconfig">.gitconfig</a></li>
<li><a href=".profile">.profile</a></li>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a></li>
<li><a href=".wget-hsts">.wget-hsts</a></li>
<li><a href=".wireshark/">.wireshark/</a></li>
<li><a href="cs144_lab3/">cs144_lab3/</a></li>
<li><a href="mininet/">mininet/</a></li>
<li><a href="oflops/">oflops/</a></li>
<li><a href="oftest/">oftest/</a></li>
<li><a href="openflow/">openflow/</a></li>
<li><a href="pox/">pox/</a></li>
</ul>
<hr>
</body>
</html>
-                   100%[===================>]     974  --.-KB/s    in 0s

2024-04-24 07:54:32 (339 MB/s) - written to stdout [974/974]

mininet> h1 kill %python
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.2 - - [24/Apr/2024 07:54:32] "GET / HTTP/1.1" 200 -
mininet> _
```

xiv. Check the python version : py sys.version

and exit the CLI : exit

```
mininet> py sys.version
3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0]
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 1566.498 seconds
mininet@mininet-vm:~$
mininet@mininet-vm:~$
```

PART 2 : ADVANCED STARTUP OPTIONS

RUN A REGRESSION TEST

Command : sudo mn –test pingpair

```
mininet@mininet-vm:~$ sudo mn --test pingpair
*** 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 3.516 seconds
mininet@mininet-vm:~$ _
```

Command : sudo mn –test iperf

```
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: ['44.1 Gbits/sec', '44.2 Gbits/sec']
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 9.646 seconds
mininet@mininet-vm:~$ _
```

CHANGING TOPOLOGY SIZE AND TYPE

The default topology is a single switch connected to two host. This can be changed. Ex. : To verify all pairs ping connectivity with one switch and three hosts.

Running a regression test:

Sudo mn –test pingall –topo single,3

```
mininet@mininet-vm:~$ sudo mn --test pingall --topo single,3
*** Creating network
*** Adding controller
*** 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 1.984 seconds
mininet@mininet-vm:~$
```

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
```

Parametrized topologies are one of Mininet's most useful and powerful features.

```
mininet@mininet-vm:~$ sudo mn --test pingall --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
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 4.547 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> iperf
...
mininet> h1 ping -c10 h2
```

```
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) (10.00Mbit 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> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.44 Mbits/sec', '11.8 Mbits/sec']
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=43.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=41.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=41.7 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=42.8 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=41.3 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=42.5 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=41.6 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9019ms
rtt min/avg/max/mdev = 41.121/41.980/43.447/0.682 ms
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:

```
$ 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:
mininet>
```

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)
```

You can also evaluate methods of variables:

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

Part 4: Python API Examples

SSH daemon per host

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

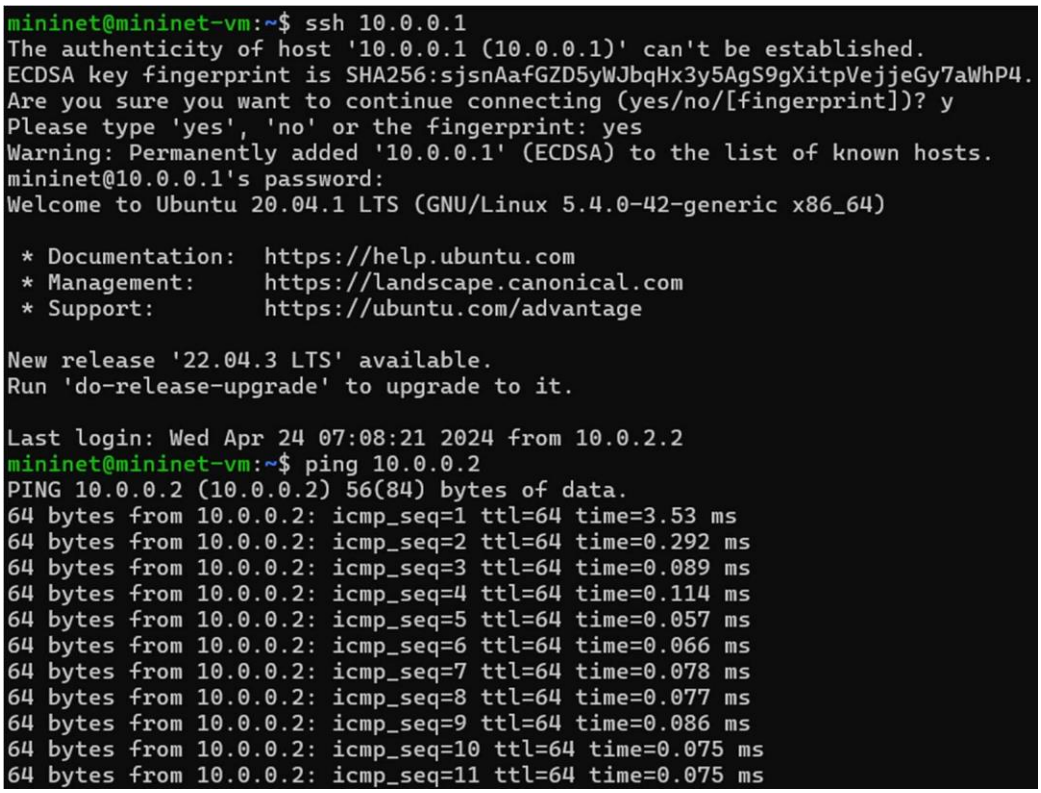
```
$ sudo ~/mininet/examples/sshd.py
```

```
mininet@mininet-vm:~$ sudo ~/mininet/examples/sshd.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** 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>
```

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

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

Exit SSH example mininet:

	
CONCLUSION:	Through this experiment, I have learned the basics of Mininet. I have also understood its functionalities and the way it operates.