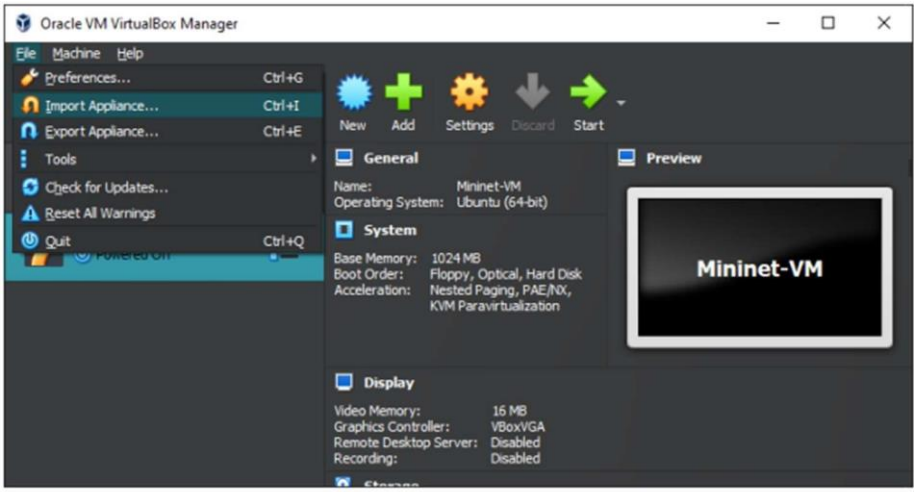
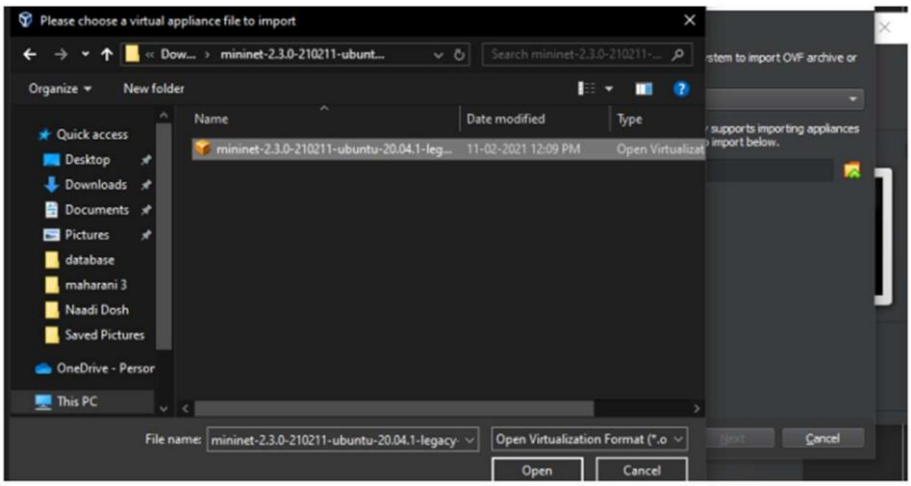
BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Name	Manish Shashikant Jadhav
UID	2023301005
Subject	Computer Communication and Networks (CCN)
Experiment No.	10
Aim	The objective of this lab exercise is to create a realistic virtual network usingMininet, a tool for emulating network environments.
Procedure	**Step 1: Introduction to Mininet** Mininet is a popular open-source network emulator used for creating virtual networks for testing and development purposes. It allows users to create complex network topologies using virtualized network devices such as switches, routers, and hosts. Mininet runs on a single machine and utilizes lightweight virtualization techniques to simulate a network environment. **Step 2**: **Installation and Setup**  

Step 3: Sample Workflow

1. After starting Mininet, you will be presented with a Mininet prompt (`mininet>`).
2. Create a simple network topology using the following

command:mininet> h1 = net.addHost('h1')

mininet> h2 = net.addHost('h2')

mininet> s1 = net.addSwitch('s1')

mininet> net.addLink(h1, s1)

mininet> net.addLink(h2, s1)

3. Start the network: mininet> net.start()
4. Test connectivity between hosts: mininet> h1 ping h2

```
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> net.start()
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Step 4: Walkthrough Follow a walkthrough tutorial provided on the Mininet website or otheronline resources to understand more complex network topologies and configurations.

```
mininet> h1 ping 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=9.07 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.330 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.079 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3030ms
rtt min/avg/max/mdev = 0.079/2.394/9.067/3.853 ms
```

- **Creating custom topologies:**

```
mininet@mininet-vm: ~
  GNU nano 4.8                                   mininet/custom/topo-2sw-2host.py
"""Custom topology example

Two directly connected switches plus a host for each switch:

   host --- switch --- switch --- host

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify   ^C Cur Pos   M-U Undo
```

Step 5: Overview

- To See Configuration of Hosts Used ifconfig command

```
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 3e:21:78:cf:65:f7  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
```

- To ping all hosts(Broadcast) we can use Pingall command

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

- Dump information about all nodes:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=683>
<Host h2: h2-eth0:10.0.0.2 pid=685>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=690>
<Controller c0: 127.0.0.1:6653 pid=676>
mininet>
```

```
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 1.778 seconds
```

- Display nodes:

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

- To print Process list use ps command:

```
mininet> h1 ps -a
    PID TTY          TIME CMD
    668 pts/0    00:00:00 sudo
    671 pts/0    00:00:00 mn
    713 pts/1    00:00:00 controller
    745 pts/2    00:00:00 ps
mininet> s1 ps -a
    PID TTY          TIME CMD
    668 pts/0    00:00:00 sudo
    671 pts/0    00:00:00 mn
    713 pts/1    00:00:00 controller
    747 pts/4    00:00:00 ps
mininet>
```

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

➢ sudo mn --link tc,bw=10,delay=10ms

➢ h1 ping -c10 h

```
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> 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=90.8 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=44.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=44.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=43.6 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=42.0 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=42.9 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=42.3 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=41.6 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=43.2 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=42.2 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 41.621/47.676/90.793/14.396 ms
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:

➢ link s1 h1 down You should see an OpenFlow Port Status Change notification get generated.To bring the link back up:

➢ link s1 h1 up

```
mininet> link s1 h1 down
mininet> h1 ping -c 1 h2
ping: connect: Network is unreachable
mininet> link s1 h1 up
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=18.3 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 18.295/18.295/18.295/0.000 ms
mininet>
```

Conclusion

Hence, by completing this experiment I came to know about Mininet.