WINTER – SEMESTER
Course Code: MCSE505L
Course-Title: – Computer Network
DIGITAL ASSIGNMENT – 1

(LAB) Slot- L35+L36

Faculty: - SRIMATHI C (SCOPE)

Name: Nidhi Singh
Reg. No: 22MAI0015

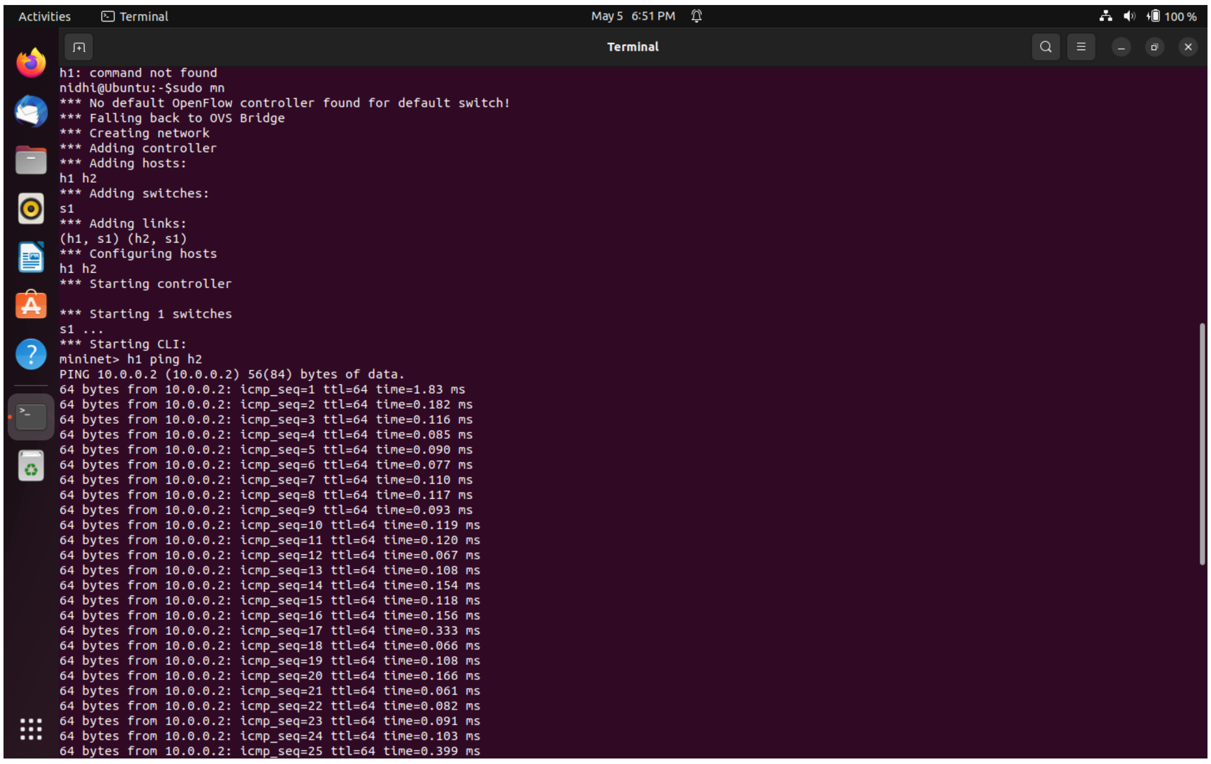
Question 1:- Create simple network in mininet.

Answer :-

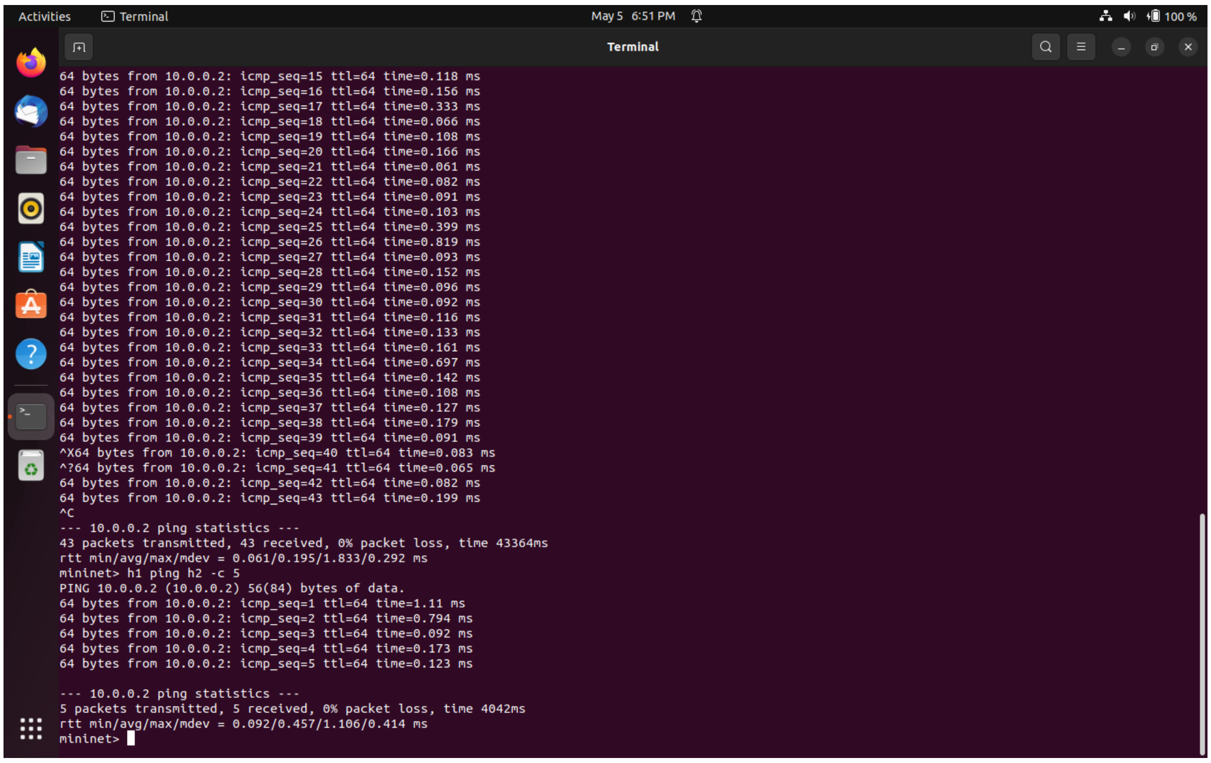
Test default network :-

In this example, we are creating a network with one switch and two hosts using the sudo mn.

here's a simple example of how to create a network in Mininet. In the code bellow, ping command is used to connect h1 and h2 nodes together.

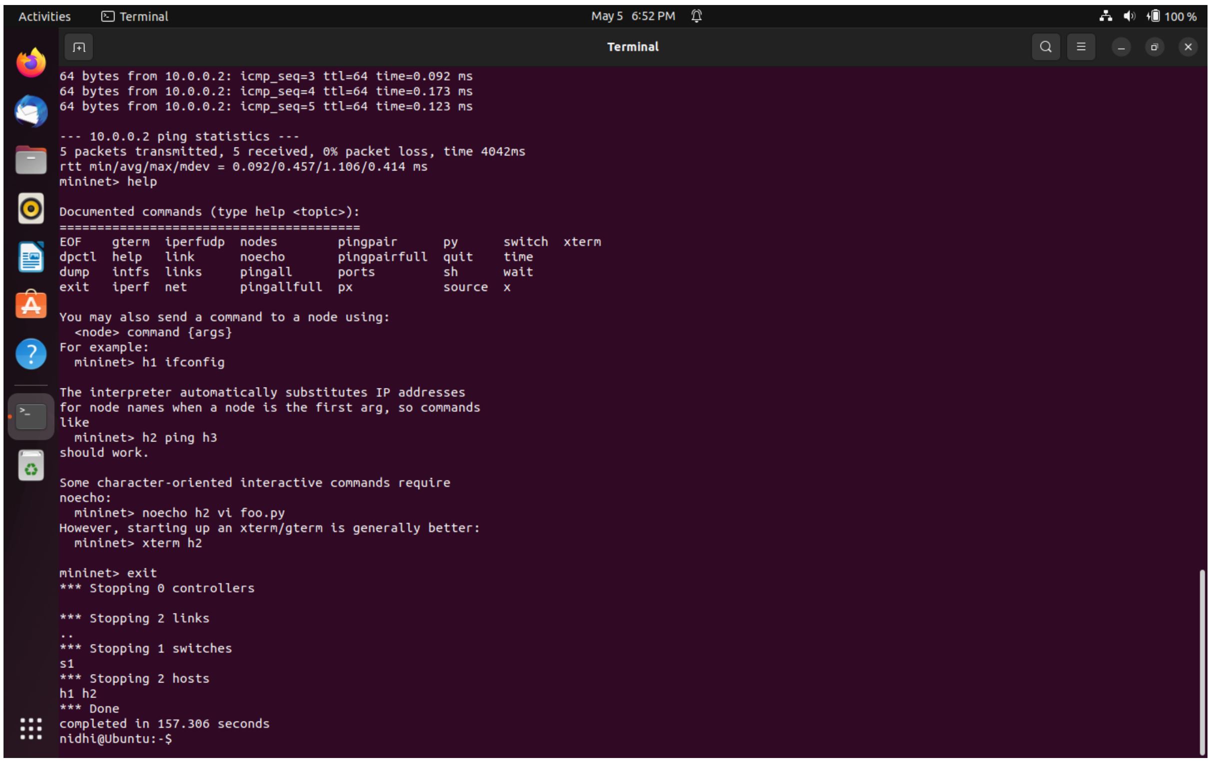


```
h1: command not found
nidhi@Ubuntu:~$sudo mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 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=1.83 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.182 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.116 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.117 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.154 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.156 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.333 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.166 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.399 ms
```



```
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.156 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.333 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.166 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.399 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.819 ms
64 bytes from 10.0.0.2: icmp_seq=27 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=0.152 ms
64 bytes from 10.0.0.2: icmp_seq=29 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2: icmp_seq=30 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=31 ttl=64 time=0.116 ms
64 bytes from 10.0.0.2: icmp_seq=32 ttl=64 time=0.133 ms
64 bytes from 10.0.0.2: icmp_seq=33 ttl=64 time=0.161 ms
64 bytes from 10.0.0.2: icmp_seq=34 ttl=64 time=0.697 ms
64 bytes from 10.0.0.2: icmp_seq=35 ttl=64 time=0.142 ms
64 bytes from 10.0.0.2: icmp_seq=36 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=37 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=38 ttl=64 time=0.179 ms
64 bytes from 10.0.0.2: icmp_seq=39 ttl=64 time=0.091 ms
^X64 bytes from 10.0.0.2: icmp_seq=40 ttl=64 time=0.083 ms
^?64 bytes from 10.0.0.2: icmp_seq=41 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=42 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=43 ttl=64 time=0.199 ms
^C
--- 10.0.0.2 ping statistics ---
43 packets transmitted, 43 received, 0% packet loss, time 43364ms
rtt min/avg/max/mdev = 0.061/0.195/1.833/0.292 ms
mininet> h1 ping h2 -c 5
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=1.11 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.794 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.173 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.123 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4042ms
rtt min/avg/max/mdev = 0.092/0.457/1.106/0.414 ms
mininet> 
```

In the code above, we transfer 5 packets from node h1 to node h2 using command h1 ping h2 –c 5 .

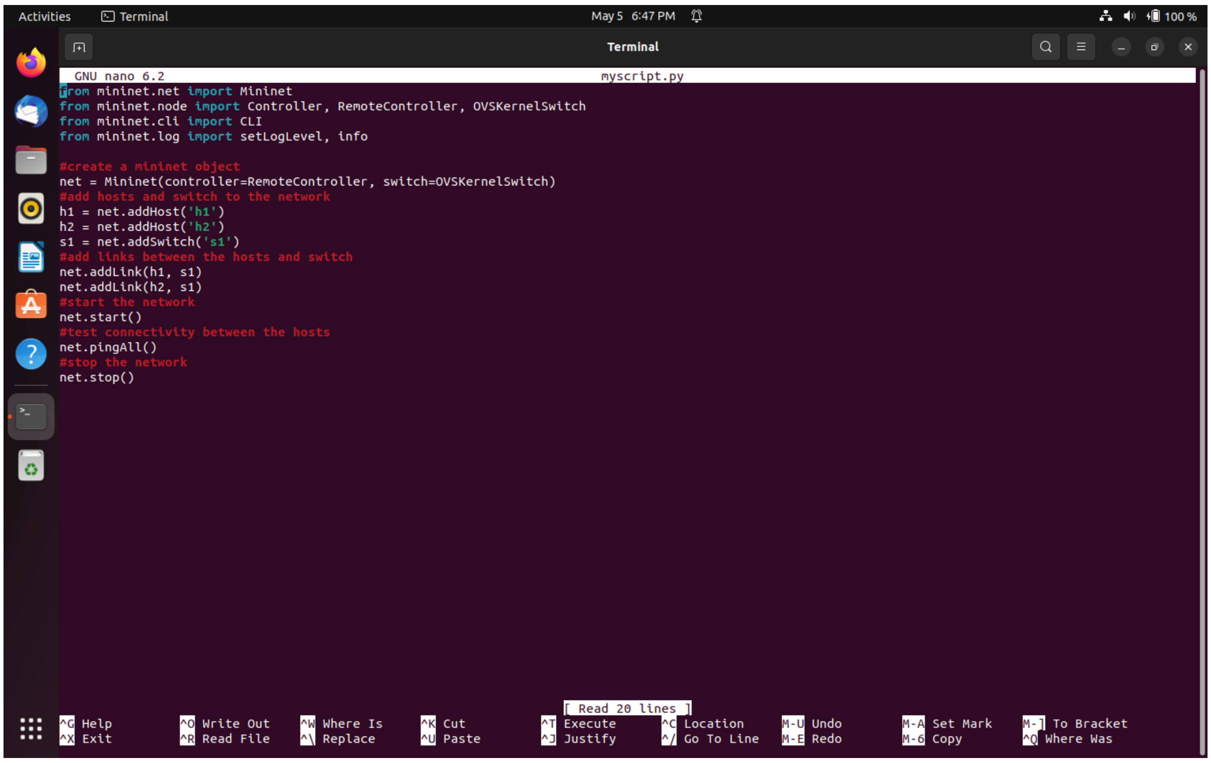


```
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.173 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.123 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4042ms
rtt min/avg/max/mdev = 0.092/0.457/1.106/0.414 ms
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> exit
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 157.306 seconds
nidhi@Ubuntu:-$
```

In the above code, help command is used to know about command documented.

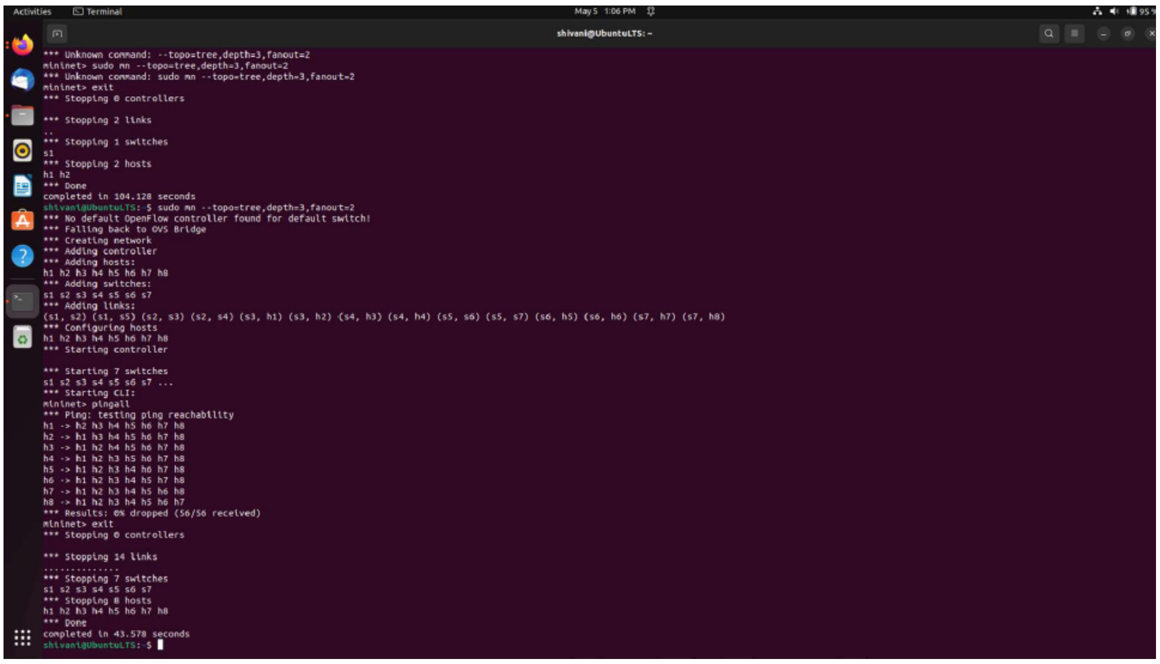
Creation of custom simple network using python :

In Mininet, you can create a simple client-server network by defining a topology with multiple hosts and using the Mininet API to run a server process on one host and a client process on another host. Here's an example of how to do this:



```
  GNU nano 6.2                                   myscript.py
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info

#create a mininet object
net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)
#add hosts and switch to the network
h1 = net.addHost('h1')
h2 = net.addHost('h2')
s1 = net.addSwitch('s1')
#add links between the hosts and switch
net.addLink(h1, s1)
net.addLink(h2, s1)
#start the network
net.start()
#test connectivity between the hosts
net.pingAll()
#stop the network
net.stop()

                                        [ Read 20 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^Q Where Was
```

Tree Topology Creation using Mininet:



```
*** Unknown command: --topo=tree,depth=3,fanout=2
mininet> sudo mn --topo=tree,depth=3,fanout=2
*** Unknown command: sudo mn --topo=tree,depth=3,fanout=2
mininet> exit
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 104.128 seconds
shivani@UbuntuLTS:~$ sudo mn --topo=tree,depth=3,fanout=2
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller

*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> exit
*** Stopping 0 controllers

*** Stopping 14 links
..............
*** Stopping 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Stopping 8 hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Done
completed in 43.578 seconds
shivani@UbuntuLTS:~$
```

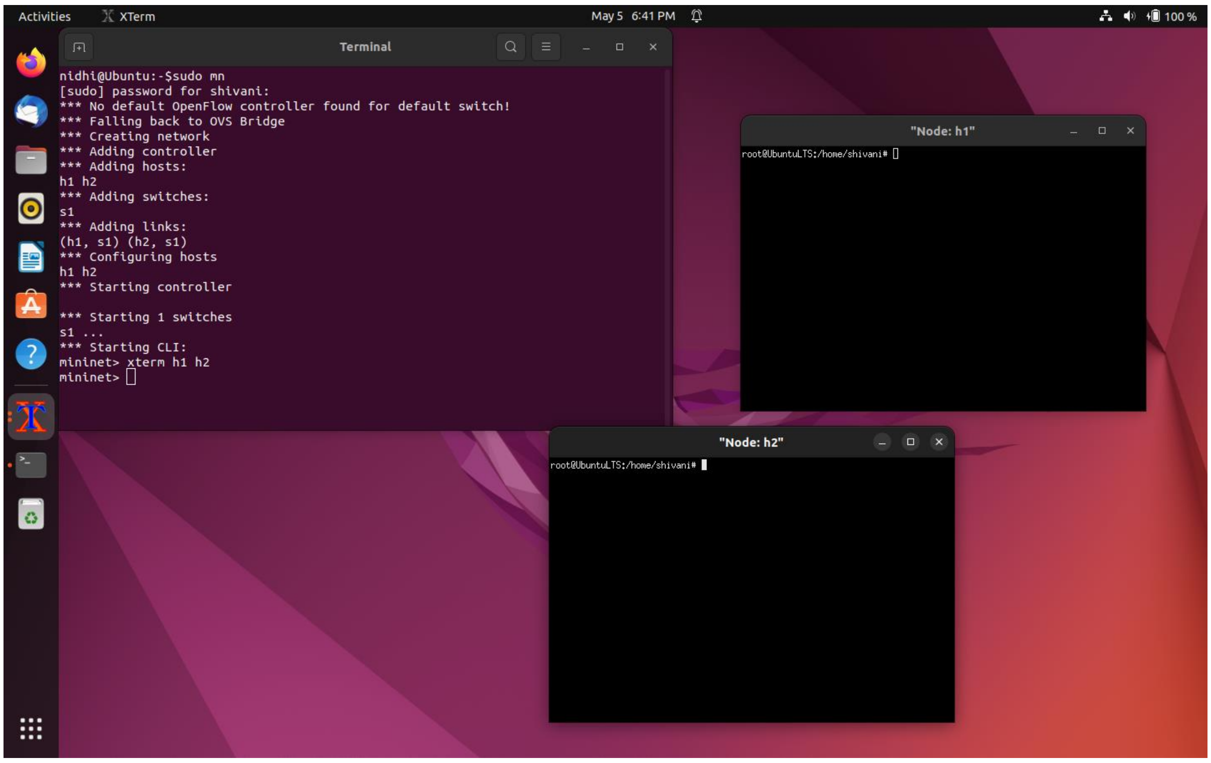
Transfer of data between client and server :-

When you run this code, you should see two xterm windows open up, one for each host in the network. The client process running on host h2 should download the index.html file from the server process running on host h1, demonstrating a simple client-server interaction.

Here is the screenshot of the process to transfer of data between client(Node :h1) and server(Node:h2).

In Mininet, you can use the xterm command to open a terminal window for each host in your network. This can be useful for interacting with the command line of individual hosts and running commands and programs within their respective namespaces.

Here's an example of how to use xterm in Mininet:

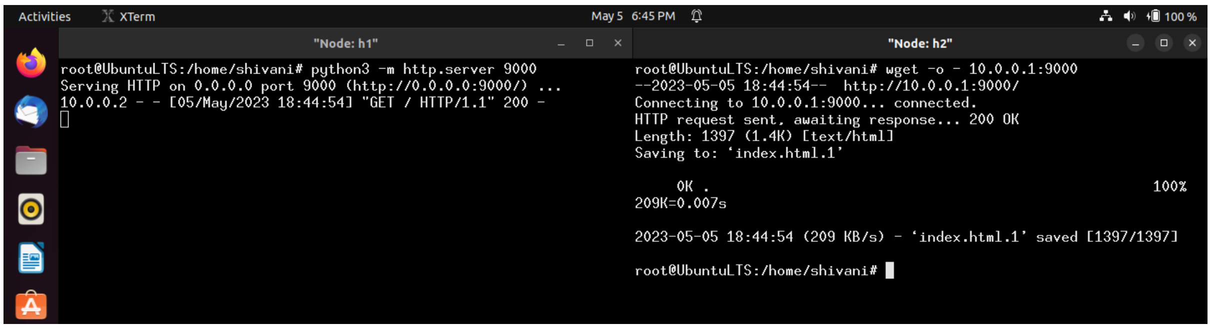


```
nidhi@Ubuntu:-$sudo mn
[sudo] password for shivani:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1 h2
mininet> 
```

```
root@UbuntuLTS:/home/shivani# 
```

```
root@UbuntuLTS:/home/shivani# 
```

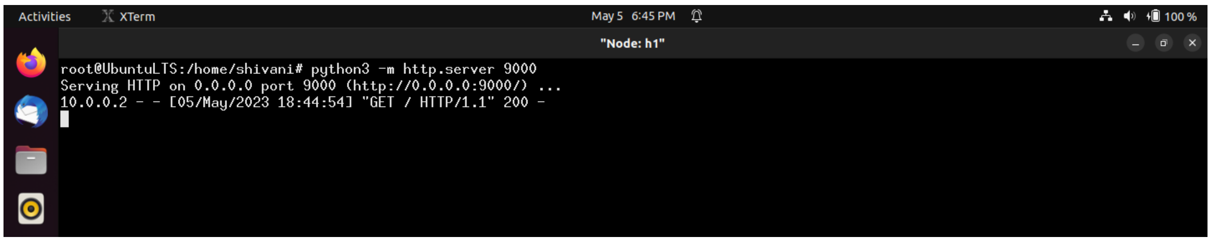
Next, we start a simple HTTP server on host h1 by running the command python –m http.server 9000. This will start a web server listening on port 9000, allowing us to serve static files over HTTP.



```
root@UbuntuLTS:/home/shivani# python3 -m http.server 9000
Serving HTTP on 0.0.0.0 port 9000 (http://0.0.0.0:9000/) ...
10.0.0.2 - - [05/May/2023 18:44:54] "GET / HTTP/1.1" 200 -
```

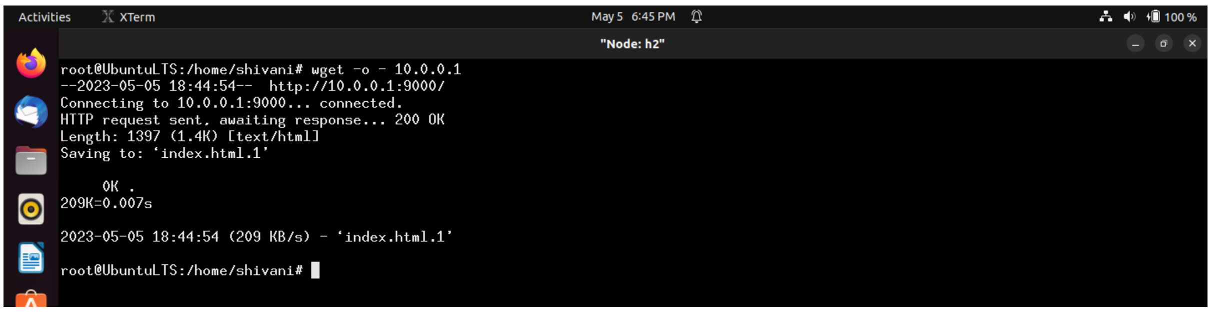
```
root@UbuntuLTS:/home/shivani# wget -o - 10.0.0.1:9000
--2023-05-05 18:44:54--  http://10.0.0.1:9000/
Connecting to 10.0.0.1:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1397 (1.4K) [text/html]
Saving to: ‘index.html.1’

     0K .                                                    100%
209K=0.007s

2023-05-05 18:44:54 (209 KB/s) - ‘index.html.1’ saved [1397/1397]

root@UbuntuLTS:/home/shivani# 
```

Next, we start a simple HTTP server on host h1 by running the command python –m http.server 9000. This will start a web server listening on port 9000, allowing us to serve static files over HTTP.



```
root@UbuntuLTS:/home/shivani# python3 -m http.server 9000
Serving HTTP on 0.0.0.0 port 9000 (http://0.0.0.0:9000/) ...
10.0.0.2 - - [05/May/2023 18:44:54] "GET / HTTP/1.1" 200 -
```

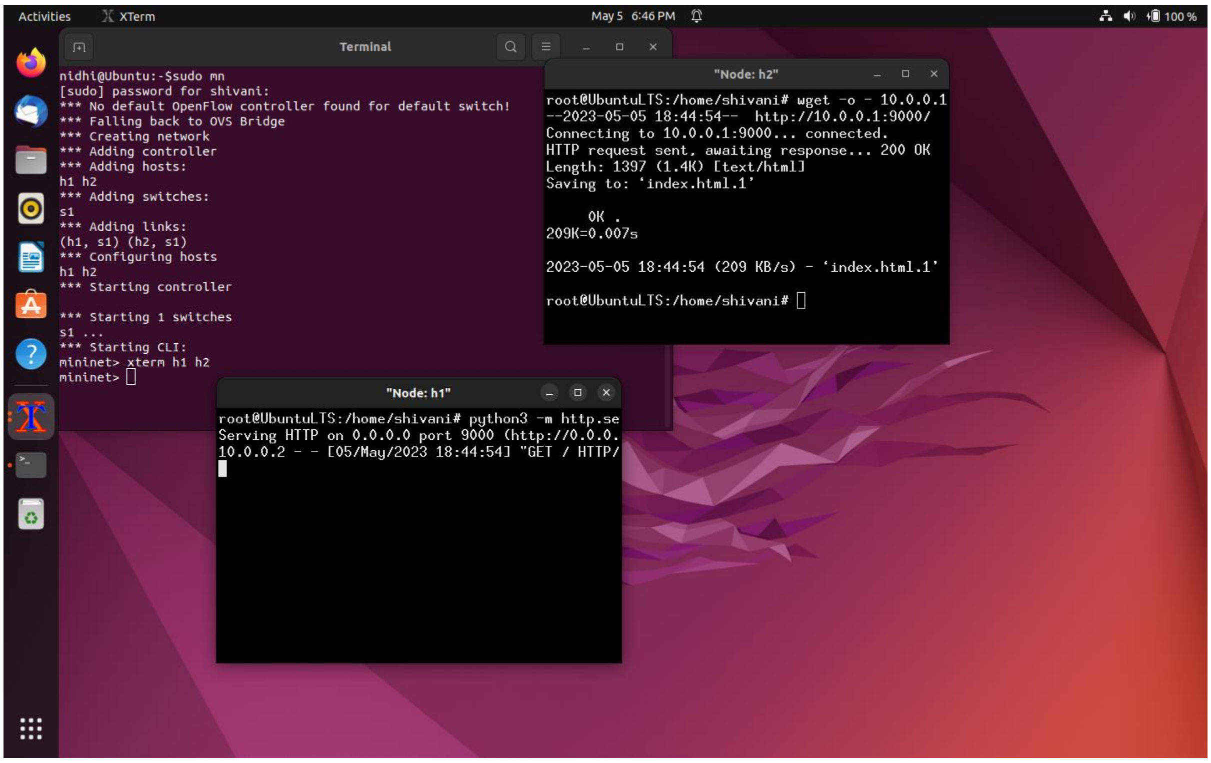
We then send a request from host h2 to the server on host h1 using the wget command, which downloads the contents of a URL to a file. In this case, we use the wget command to download the index.html.1 file from the server running on host h1.



```
root@UbuntuLTS:/home/shivani# wget -o - 10.0.0.1
--2023-05-05 18:44:54--  http://10.0.0.1:9000/
Connecting to 10.0.0.1:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1397 (1.4K) [text/html]
Saving to: ‘index.html.1’

     0K .
209K=0.007s

2023-05-05 18:44:54 (209 KB/s) - ‘index.html.1’

root@UbuntuLTS:/home/shivani# 
```

In Mininet, you can use the xterm command to open a terminal window for each host in your network. This can be useful for interacting with the command line of individual hosts and running commands and programs within their respective namespaces.

Here's an example of how to use xterm in Mininet:



```
nidhi@Ubuntu:-$sudo mn
[sudo] password for shivani:
*** 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> xterm h1 h2
mininet>
```

```
root@UbuntuLTS:/home/shivani# wget -o - 10.0.0.1
--2023-05-05 18:44:54--  http://10.0.0.1:9000/
Connecting to 10.0.0.1:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1397 (1.4K) [text/html]
Saving to: ‘index.html.1’

     OK .
209K=0.007s

2023-05-05 18:44:54 (209 KB/s) - ‘index.html.1’

root@UbuntuLTS:/home/shivani#
```

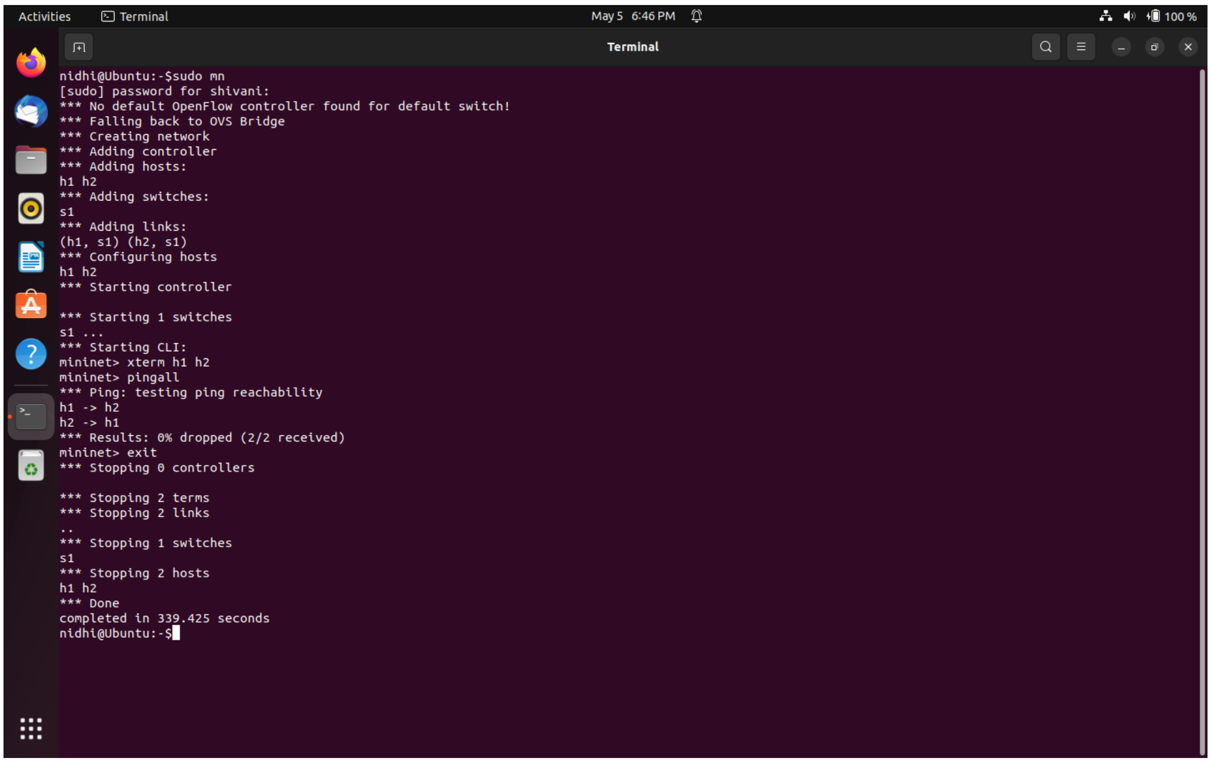
```
root@UbuntuLTS:/home/shivani# python3 -m http.se
Serving HTTP on 0.0.0.0 port 9000 (http://0.0.0.
10.0.0.2 - - [05/May/2023 18:44:54] "GET / HTTP/
```

pingall is a command in Mininet that allows you to test the connectivity between all hosts in your network. When you run the pingall command, Mininet sends ICMP echo request packets from each host to every other host in the network, and reports whether each packet was successfully received or not.

Here's an example of how to use pingall in Mininet:

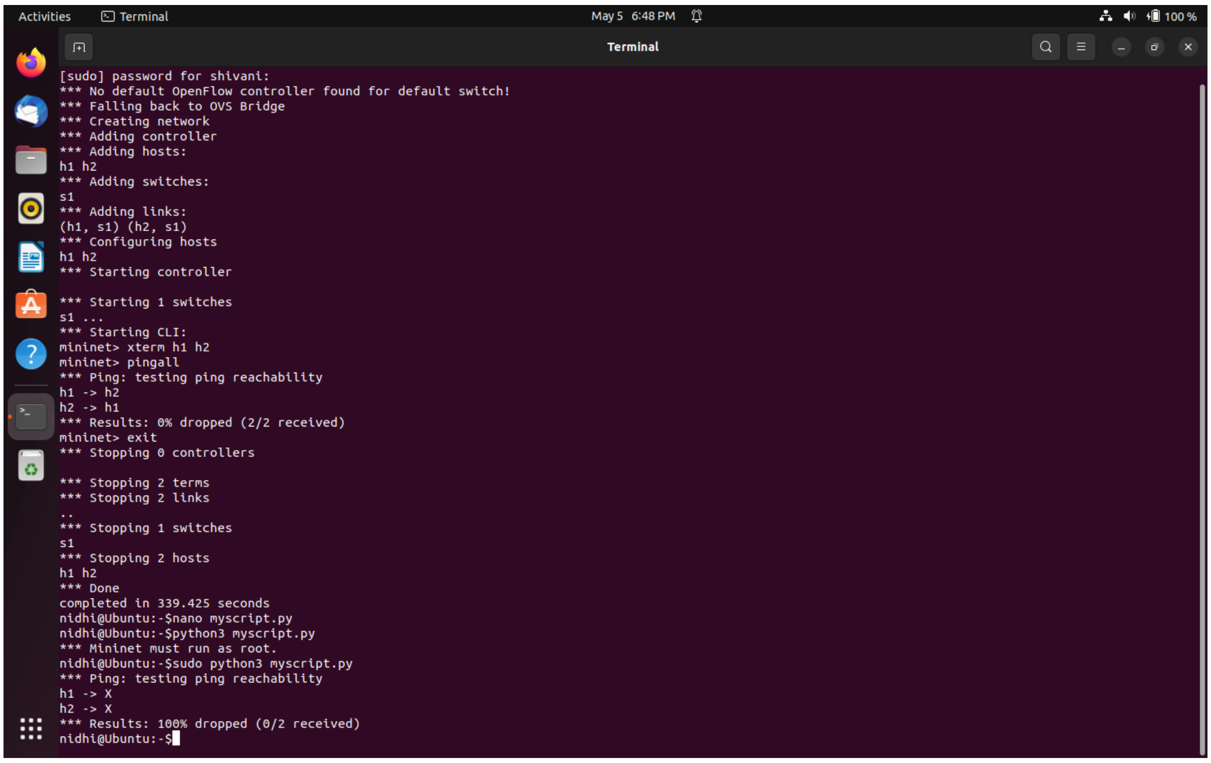
In Mininet, you can use the xterm command to open a terminal window for specific hosts in your network. This can be useful for interacting with the command line of individual hosts and running commands and programs within their respective namespaces.

To open xterm windows for hosts h1 and h2, you can run the following commands:



```
nidhi@Ubuntu:~$sudo mn
[sudo] password for shivani:
*** 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> xterm h1 h2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> exit
*** Stopping 0 controllers

*** Stopping 2 terms
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 339.425 seconds
nidhi@Ubuntu:~$
```

Finally, we open an xterm window for each host in the network, open the Mininet CLI, and stop the network when the CLI is exited.



```
[sudo] password for shivani:
*** 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> xterm h1 h2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> exit
*** Stopping 0 controllers

*** Stopping 2 terms
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 339.425 seconds
nidhi@Ubuntu:~$nano myscript.py
nidhi@Ubuntu:~$python3 myscript.py
*** Mininet must run as root.
nidhi@Ubuntu:~$sudo python3 myscript.py
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
nidhi@Ubuntu:~$
```