



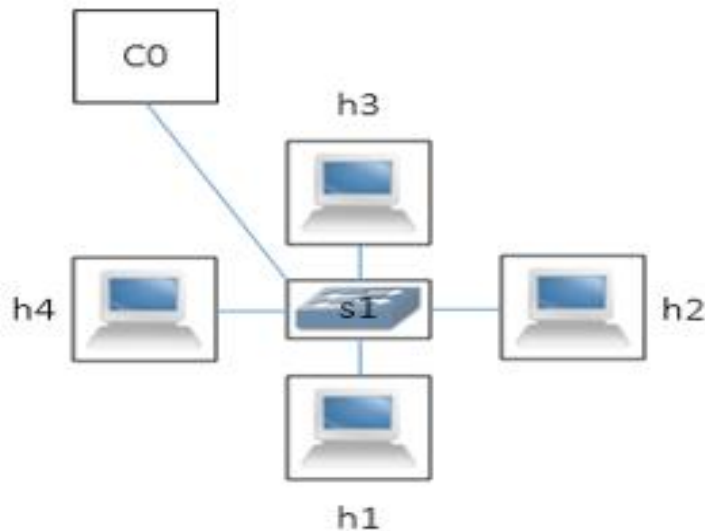
TCET-747 NEXT GENERATION NETWORKS

Prof. Yossi Nygate



Omkar Bhalekar | Abhishek Sakpal

SDN MININET ASSIGNMENT

Q.1. Warm-Up questions solve these first.

- a. What command can be used to obtain the network pictured above
- i. `sudo mn --topo linear,4`
 - ii. `sudo mn --topo star,4`
 - iii. `sudo mn --topo tree,depth=2,fanout=4`
 - iv. `sudo mn --topo single,4`

➤ The command `sudo mn --topo single,4` is used to obtain the above network.

```
root@mininet-vm: /home/mininet
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo su
root@mininet-vm:/home/mininet# sudo mn --topo single,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

b. What three commands do you run to check the network configuration, elements present and connectivity between all hosts?

- 1) The command **nodes** is used to check the network configuration such as all the network devices present in the network
- 2) The command **net** is used to show all the connections between the hosts and switches or other network devices
- 3) The command **pingall** is used to test the connectivity between all the hosts

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
mininet> pingall
*** 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)
mininet>
```

c. Why are there 3 values of RTT provided after completing a Connectivity test? What is the significance of the max RTT?

- The three values of RTT represent the minimum RTT, Average RTT and Maximum RTT. The significance of max RTT is to represent the highest time taken between the transmission of a packet and reception of its acknowledgement

```
root@mininet-vm:/home/mininet# ping -c 10.0.0.2
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
          [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
          [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
          [-w deadline] [-W timeout] [hop1 ...] destination
root@mininet-vm:/home/mininet# ping -c 10 10.0.0.2
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=1.74 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.37 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.224 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.757 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.045 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9003ms
rtt min/avg/max/mdev = 0.033/0.440/1.748/0.604 ms
root@mininet-vm:/home/mininet#
```

d. What command manually assigns an IP address to an interface?

- h1 ifconfig h1-eth0 192.168.1.0 netmask 255.255.255.0

```
mininet> h1 ifconfig h1-eth0 10.1.1.1 netmask 255.0.0.0
```

- e. What is the command to check the IP address and other details of an interface such as Rx/Tx bytes/packages of a specific host?

➤ h1 ifconfig

```
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr ba:fb:14:1c:8b:51
         inet addr:10.1.1.1  Bcast:10.255.255.255  Mask:255.0.0.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:40 errors:0 dropped:0 overruns:0 frame:0
         TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:9516 (9.5 KB)  TX bytes:840 (840.0 B)

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         UP LOOPBACK RUNNING  MTU:65536  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

- f. How do you clear previous running processes and topology to start a new simulation? Why do we do it?

➤ Ctrl+Z or type exit

Sudo mn -c

➤ The reason to clear a running process is to avoid overwriting a new process over a currently existing process. When we directly try to create a new topology, it first deletes the current topology. This means it deletes all the running processes. Hence, we need to delete the process using *sudo mn -c*

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 777.849 seconds
```

```

root@mininet-vm:/home/mininet# sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openfl
owd 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
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.

```

g. What is tcpdump used for?

- Tcpdump is a packet analyzing utility is used to capture, analyze and save the packets for network monitoring. It basically dumps TCP packet headers and displays captured/dropped packets.

```

root@mininet-vm:/home/mininet# ping -c 10 10.0.0.2
Usage: ping [-aAbBdDfhLnOqrRUvW] [-c count] [-i interval] [-I interface]
          [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
          [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
          [-w deadline] [-W timeout] [hop1 ...] destination
root@mininet-vm:/home/mininet# ping -c 10 10.0.0.2
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=1.74 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.37 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.224 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.757 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.045 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9003ms
rtt min/avg/max/mdev = 0.033/0.440/1.748/0.604 ms
root@mininet-vm:/home/mininet#

```



```

root@mininet-vm:/home/mininet# sudo tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
22:15:09.482592 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4088, seq 1, length 64
22:15:09.482664 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4088, seq 1, length 64
22:15:10.485127 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4088, seq 2, length 64
22:15:10.485140 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4088, seq 2, length 64
22:15:11.486357 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4088, seq 3, length 64
22:15:11.486387 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4088, seq 3, length 64
22:15:12.485552 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4088, seq 4, length 64
22:15:12.485574 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4088, seq 4, length 64
22:15:13.484995 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4088, seq 5, length 64
22:15:13.485013 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4088, seq 5, length 64

```

h. Can we bring the link down in a running network? If yes how?

➤ Yes. By using the command *link h4 s1 down*

```

mininet> link h4 s1 down
mininet> h1 ping -c 1 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

```

```

mininet> h4 ping -c 1 h2
connect: Network is unreachable
mininet>

```

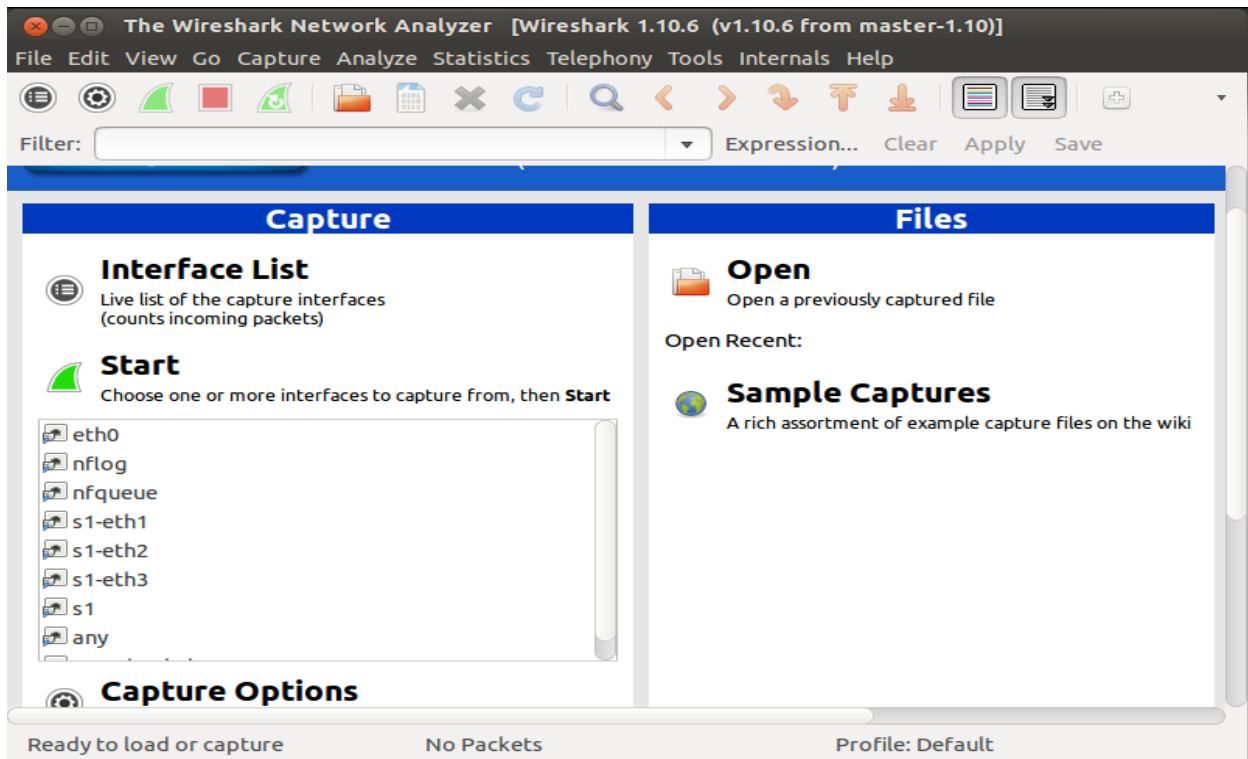
i. How do you start Wireshark and check Openflow messages? (Include necessary Screenshots)

➤ We can start wireshark using command *sudo wireshark &*

```

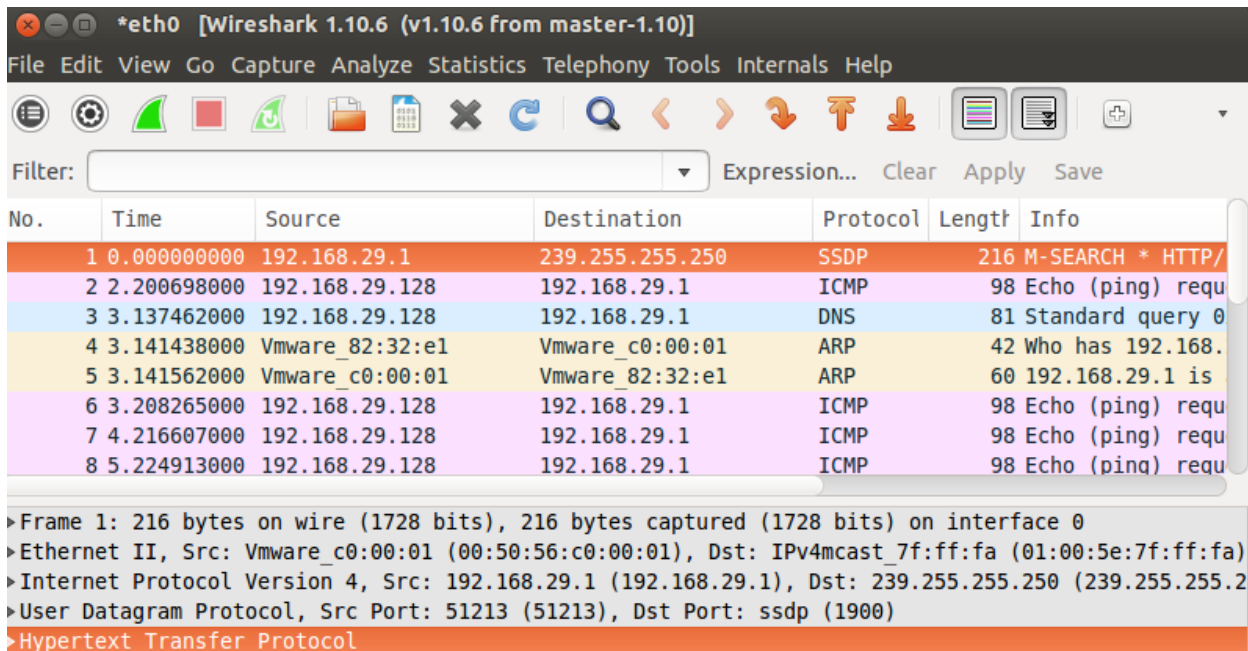
root@mininet-vm:/home/mininet# sudo wireshark &
[1] 4257

```



Openflow Messages

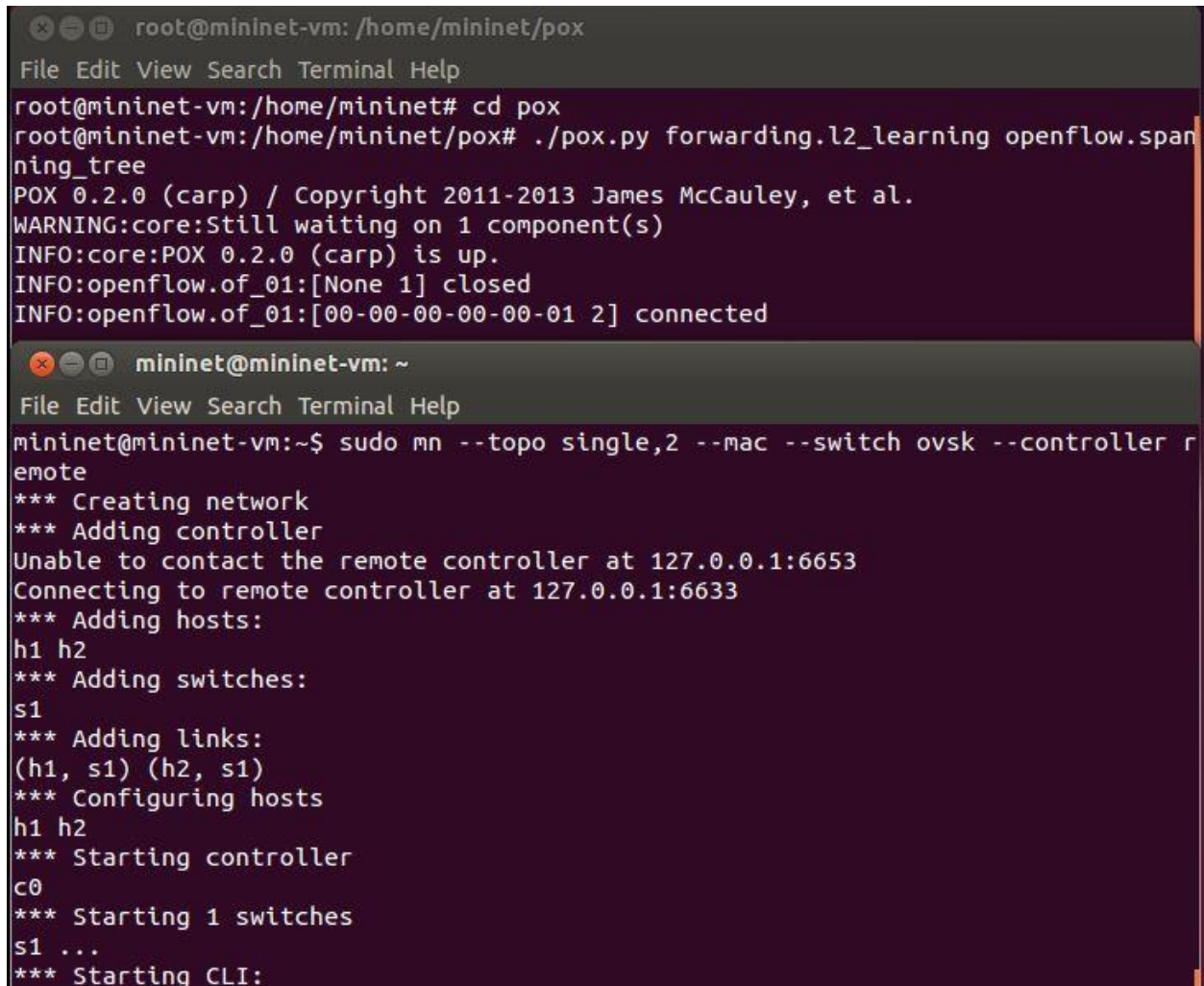
```
root@mininet-vm:/home/mininet# ping 192.168.29.1
PING 192.168.29.1 (192.168.29.1) 56(84) bytes of data.
```



j. How do you start the pox controller with forwarding (as a l2 learning switch), with spanning-tree?

➤ cd pox

/pox/pox.py forwarding.l2_learning openflow.spanning_tree



```
root@mininet-vm: /home/mininet/pox
File Edit View Search Terminal Help
root@mininet-vm:/home/mininet# cd pox
root@mininet-vm:/home/mininet/pox# ./pox.py forwarding.l2_learning openflow.spanning_tree
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
WARNING:core:Still waiting on 1 component(s)
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo mn --topo single,2 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** 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:
```

Q.2. Create a Tree topology network with Depth = 3 and Fanout = 4, with default link configuration (which is 0 msec delay and unlimited link bandwidth). Check the performance between hosts h1 and h64.

➤ `sudo mn --topo tree,depth=3,fanout=4`

```
mininet@mininet-vm:~$ sudo mn --switch ovs --controller ref --topo tree,depth=3,fanout=4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h
28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h
53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21
*** Adding links:
(s1, s2) (s1, s7) (s1, s12) (s1, s17) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s3, h1) (s3, h2) (s3, h3)
(s3, h4) (s4, h5) (s4, h6) (s4, h7) (s4, h8) (s5, h9) (s5, h10) (s5, h11) (s5, h12) (s6, h13) (s6,
h14) (s6, h15) (s6, h16) (s7, s8) (s7, s9) (s7, s10) (s7, s11) (s8, h17) (s8, h18) (s8, h19) (s8, h2
0) (s9, h21) (s9, h22) (s9, h23) (s9, h24) (s10, h25) (s10, h26) (s10, h27) (s10, h28) (s11, h29) (s
11, h30) (s11, h31) (s11, h32) (s12, s13) (s12, s14) (s12, s15) (s12, s16) (s13, h33) (s13, h34) (s1
3, h35) (s13, h36) (s14, h37) (s14, h38) (s14, h39) (s14, h40) (s15, h41) (s15, h42) (s15, h43) (s15
, h44) (s16, h45) (s16, h46) (s16, h47) (s16, h48) (s17, s18) (s17, s19) (s17, s20) (s17, s21) (s18,
h49) (s18, h50) (s18, h51) (s18, h52) (s19, h53) (s19, h54) (s19, h55) (s19, h56) (s20, h57) (s20,
h58) (s20, h59) (s20, h60) (s21, h61) (s21, h62) (s21, h63) (s21, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h
28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h
53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 21 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 ...
*** Starting CLI:
```

a). Perform ping 10 times and report the 3 values of RTT between the h1 and h64. After you open xterms for h1 and h64, run tcpdump on h64 and 'ping -c 10 10.0.0.64' from h1 (Take screen shot of both h1 and h64).

➤ Ping 10 Times

```
mininet>
mininet>
mininet>
mininet>
mininet> h1 ping -c 10 h64
PING 10.0.0.64 (10.0.0.64) 56(84) bytes of data.
64 bytes from 10.0.0.64: icmp_seq=1 ttl=64 time=1.04 ms
64 bytes from 10.0.0.64: icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from 10.0.0.64: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.64: icmp_seq=4 ttl=64 time=0.066 ms
64 bytes from 10.0.0.64: icmp_seq=5 ttl=64 time=0.068 ms
64 bytes from 10.0.0.64: icmp_seq=6 ttl=64 time=0.087 ms
64 bytes from 10.0.0.64: icmp_seq=7 ttl=64 time=0.067 ms
64 bytes from 10.0.0.64: icmp_seq=8 ttl=64 time=0.072 ms
64 bytes from 10.0.0.64: icmp_seq=9 ttl=64 time=0.219 ms
64 bytes from 10.0.0.64: icmp_seq=10 ttl=64 time=0.086 ms

--- 10.0.0.64 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9021ms
rtt min/avg/max/mdev = 0.066/0.187/1.041/0.288 ms
mininet>
```

3 values of RTT:

Min RTT= 0.066 ms

Average RTT= 0.187 ms

Max RTT= 1.041 ms

Ping from H1 to H64

```
root@mininet-vm:~# ping -c 10 10.0.0.64
PING 10.0.0.64 (10.0.0.64) 56(84) bytes of data.
64 bytes from 10.0.0.64: icmp_seq=1 ttl=64 time=0.513 ms
64 bytes from 10.0.0.64: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 10.0.0.64: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.64: icmp_seq=4 ttl=64 time=0.123 ms
64 bytes from 10.0.0.64: icmp_seq=5 ttl=64 time=0.092 ms
64 bytes from 10.0.0.64: icmp_seq=6 ttl=64 time=0.055 ms
64 bytes from 10.0.0.64: icmp_seq=7 ttl=64 time=0.099 ms
64 bytes from 10.0.0.64: icmp_seq=8 ttl=64 time=0.101 ms
64 bytes from 10.0.0.64: icmp_seq=9 ttl=64 time=0.074 ms
64 bytes from 10.0.0.64: icmp_seq=10 ttl=64 time=0.065 ms

--- 10.0.0.64 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 0.055/0.125/0.513/0.131 ms
```

Run tcpdump on H64

```
"Node: h64"
th 64
15:40:17.995999 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 3462, seq 1, length 64
15:40:19.015139 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 3462, seq 2, length 64
15:40:19.015178 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 3462, seq 2, length 64
15:40:20.010676 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 3462, seq 3, length 64
15:40:20.010710 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 3462, seq 3, length 64
15:40:21.012553 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 3462, seq 4, length 64
15:40:21.012584 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 3462, seq 4, length 64
15:40:22.012776 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 3462, seq 5, length 64
15:40:22.012806 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 3462, seq 5, length 64
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
root@mininet-vm:~#
```

b. Make h64 as the server and h1 as the client and perform iperf for 1 minute (Iperf takes in time parameter in seconds, which in this case would be 60 seconds. Anytime you want to check details about any command, type the command on Linux terminal with the parameter -h, e.g. to checks the parameters accepted by iperf, type “iperf -h”). What is the bandwidth?

- Make h1 as client
- Perform iperf for 1 minute

```
"Node: h1"
root@mininet-vm:~# iperf -t60 -c 10.0.0.64
-----
Client connecting to 10.0.0.64, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[177] local 10.0.0.1 port 47960 connected with 10.0.0.64 port 5001
[ ID] Interval      Transfer    Bandwidth
[177] 0.0-60.0 sec  45.3 GBytes  6.49 Gbits/sec
```

- Make h64 as server
- Analyzing iperf on server h64

```

"Node: h64"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[178] local 10.0.0.64 port 5001 connected with 10.0.0.1 port 47960
[ ID] Interval      Transfer    Bandwidth
[178]  0.0-60.0 sec  45.3 GBytes 6.49 Gbits/sec

```

As it is observed the Bandwidth is 6.49 Gbps between both the node h1 and node h64

c. Iperf gives us the time duration, data transmitted and the bandwidth, by controlling the amount of time we send the data. Suppose we want to control the amount of data transmitted and check the time it takes to transmit, we would need to use Iperf3. Use iperf3 to send 10000 packets from h1 to h64. What is the bandwidth?

- Iperf3 to send 10000 packets from h1 to h64

```

root@mininet-vm:~# iperf3 -k10000 -c 10.0.0.64
Connecting to host 10.0.0.64, port 5201
[178] local 10.0.0.1 port 37452 connected to 10.0.0.64 port 5201
[ ID] Interval      Transfer    Bandwidth    Retr  Cwnd
[178]  0.00-0.80    sec  1.22 GBytes 13.0 Gbits/sec    0   5.44 MBytes
-----
[ ID] Interval      Transfer    Bandwidth    Retr
[178]  0.00-0.80    sec  1.22 GBytes 13.0 Gbits/sec    0
[178]  0.00-0.80    sec  1.22 GBytes 13.0 Gbits/sec    0
iperf Done.

```



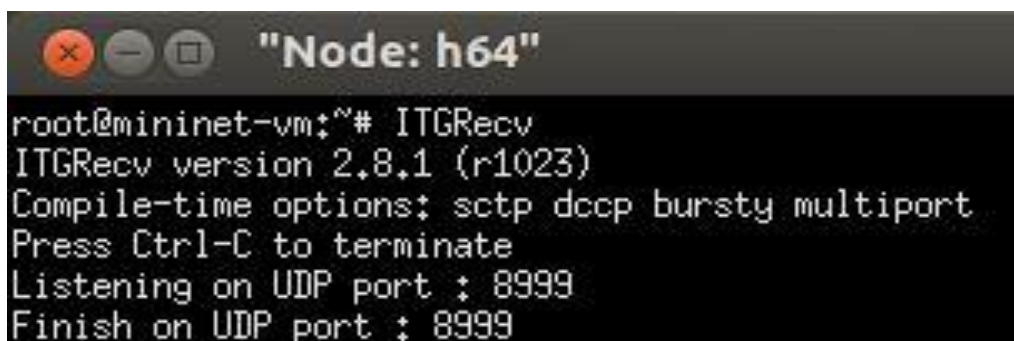
```

root@mininet-vm:~# iperf3 -s
warning: this system does not seem to support IPv6 - trying IPv4
-----
Server listening on 5201
-----
Accepted connection from 10.0.0.1, port 37450
[179] local 10.0.0.64 port 5201 connected to 10.0.0.1 port 37452
[ ID] Interval      Transfer    Bandwidth
[179]  0.00-0.81    sec  1.22 GBytes  12.9 Gbits/sec
-----
[ ID] Interval      Transfer    Bandwidth    Retr
[179]  0.00-0.81    sec  1.22 GBytes  13.0 Gbits/sec    0
[179]  0.00-0.81    sec  1.22 GBytes  12.9 Gbits/sec
-----
Server listening on 5201
-----

```

We can observe that using iperf3, the bandwidth between h1 and h64 is 13.0 Gbps

d). While Iperf and Iperf3 are used to get performance in terms of data transmitted and the bandwidth and the time performed, we need to use D-ITG to get more details such as latency, packet loss, number of packets, number of bytes, minimum delay, average delay, maximum delay, jitter, average bit rate etc. Use DITG to get the average delay and packet loss between h1 and h64. Note to enable logging when sending the flows. The results generated are encoded and need to be decoded by using ITGDec.



```

"Node: h64"
root@mininet-vm:~# ITGRecv
ITGRecv version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Press Ctrl-C to terminate
Listening on UDP port : 8999
Finish on UDP port : 8999

```



```

Node: h1
Finished sending packets of flow ID: 1

root@mininet-vm:~# ITGDec receiver.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
\-----
Flow number: 1
From 10.0.0.1:37861
To 10.0.0.64:8999
-----
Total time = 1.976273 s
Total packets = 734
Minimum delay = 0.000032 s
Maximum delay = 0.022430 s
Average delay = 0.000251 s
Average jitter = 0.000122 s
Delay standard deviation = 0.001247 s
Bytes received = 375808
Average bitrate = 1521.279702 Kbit/s
Average packet rate = 371.406177 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
***** TOTAL RESULTS *****
-----
Number of flows = 1
Total time = 1.976273 s
Total packets = 734
Minimum delay = 0.000032 s
Maximum delay = 0.022430 s
Average delay = 0.000251 s
Average jitter = 0.000122 s
Delay standard deviation = 0.001247 s
Bytes received = 375808
Average bitrate = 1521.279702 Kbit/s
Average packet rate = 371.406177 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
-----

```

Between h1 and h64:

Average delay: 0.000251 s

Packet Loss: 0 (0.00%)

e. Repeat above sub questions (a. through d.) with the same network topology but with a link that has a delay of 10ms and a bandwidth limit of 10MBits/sec. What differences do you observe?

A section

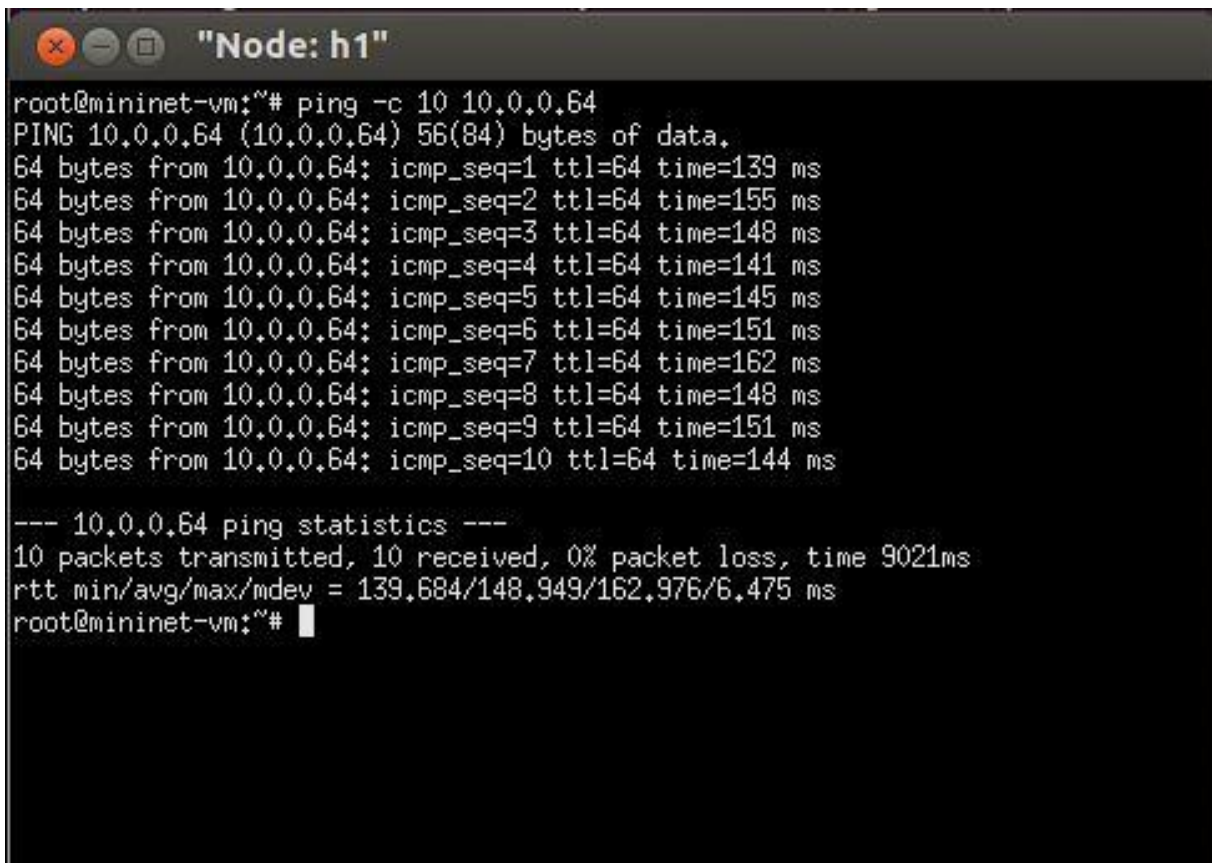
➤ Node h64

```

Node: h64
gth 64
21:47:09.337461 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 1, length 64
21:47:10.341036 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 2, length 64
gth 64
21:47:10.353206 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 2, length 64
21:47:11.339876 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 3, length 64
gth 64
21:47:11.351374 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 3, length 64
21:47:12.334778 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 4, length 64
gth 64
21:47:12.347299 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 4, length 64
21:47:13.343593 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 5, length 64
gth 64
21:47:13.355035 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 5, length 64
21:47:14.344747 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 6, length 64
gth 64
21:47:14.356654 ARP, Request who-has 10.0.0.1 tell 10.0.0.64, length 28
21:47:14.356699 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 6, length 64
21:47:14.522644 ARP, Reply 10.0.0.1 is-at 8e:13:43:7f:b1:39 (oui Unknown), length 28
21:47:15.343394 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 7, length 64
gth 64
21:47:15.356474 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 7, length 64
21:47:16.347379 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 8, length 64
gth 64
21:47:16.361144 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 8, length 64
21:47:17.355139 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 9, length 64
gth 64
21:47:17.366918 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 9, length 64
21:47:18.351940 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 10, length 64
21:47:18.363504 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 10, length 64

```

➤ Node h1



```
root@mininet-vm:~# ping -c 10 10.0.0.64
PING 10.0.0.64 (10.0.0.64) 56(84) bytes of data:
64 bytes from 10.0.0.64: icmp_seq=1 ttl=64 time=139 ms
64 bytes from 10.0.0.64: icmp_seq=2 ttl=64 time=155 ms
64 bytes from 10.0.0.64: icmp_seq=3 ttl=64 time=148 ms
64 bytes from 10.0.0.64: icmp_seq=4 ttl=64 time=141 ms
64 bytes from 10.0.0.64: icmp_seq=5 ttl=64 time=145 ms
64 bytes from 10.0.0.64: icmp_seq=6 ttl=64 time=151 ms
64 bytes from 10.0.0.64: icmp_seq=7 ttl=64 time=162 ms
64 bytes from 10.0.0.64: icmp_seq=8 ttl=64 time=148 ms
64 bytes from 10.0.0.64: icmp_seq=9 ttl=64 time=151 ms
64 bytes from 10.0.0.64: icmp_seq=10 ttl=64 time=144 ms

--- 10.0.0.64 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9021ms
rtt min/avg/max/mdev = 139.684/148.949/162.976/6.475 ms
root@mininet-vm:~#
```

- The average RTT is 148.949 which is more than previous RTT with more delay and bandwidth limitation.

B Section

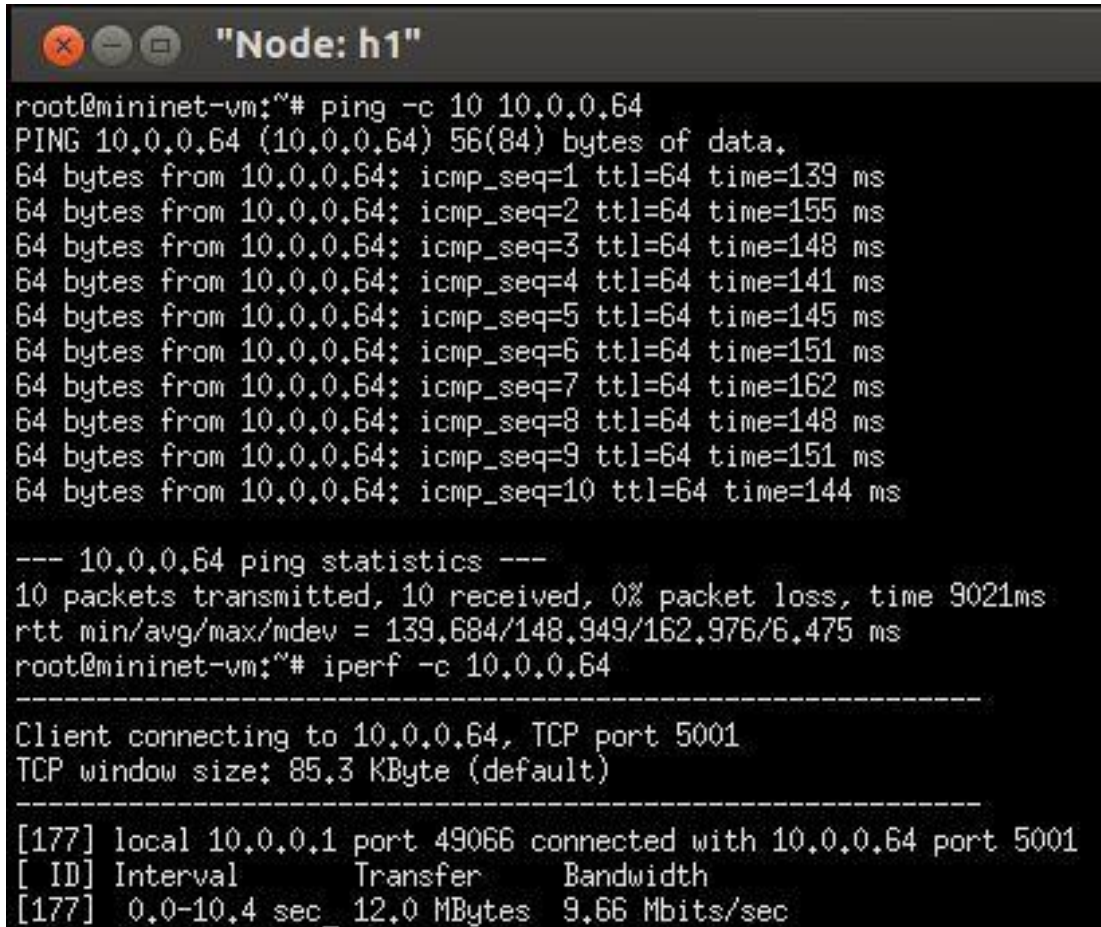
➤ Node h64

```

Node: h64
21:47:12.347299 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 4, length 64
21:47:13.343593 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 5, length 64
21:47:13.355035 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 5, length 64
21:47:14.344747 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 6, length 64
21:47:14.356654 ARP, Request who-has 10.0.0.1 tell 10.0.0.64, length 28
21:47:14.356699 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 6, length 64
21:47:14.522644 ARP, Reply 10.0.0.1 is-at 8e:13:43:7f:b1:39 (oui Unknown), length 28
21:47:15.343394 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 7, length 64
21:47:15.356474 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 7, length 64
21:47:16.347379 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 8, length 64
21:47:16.361144 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 8, length 64
21:47:17.355139 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 9, length 64
21:47:17.366918 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 9, length 64
21:47:18.351940 IP 10.0.0.1 > 10.0.0.64: ICMP echo request, id 26598, seq 10, length 64
21:47:18.363504 IP 10.0.0.64 > 10.0.0.1: ICMP echo reply, id 26598, seq 10, length 64
^C
22 packets captured
22 packets received by filter
0 packets dropped by kernel
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[178] local 10.0.0.64 port 5001 connected with 10.0.0.1 port 49066
[ ID] Interval      Transfer      Bandwidth
[178] 0.0-12.3 sec    12.0 MBytes   8.16 Mbits/sec

```


➤ Node h1



```
root@mininet-vm:~# ping -c 10 10.0.0.64
PING 10.0.0.64 (10.0.0.64) 56(84) bytes of data:
64 bytes from 10.0.0.64: icmp_seq=1 ttl=64 time=139 ms
64 bytes from 10.0.0.64: icmp_seq=2 ttl=64 time=155 ms
64 bytes from 10.0.0.64: icmp_seq=3 ttl=64 time=148 ms
64 bytes from 10.0.0.64: icmp_seq=4 ttl=64 time=141 ms
64 bytes from 10.0.0.64: icmp_seq=5 ttl=64 time=145 ms
64 bytes from 10.0.0.64: icmp_seq=6 ttl=64 time=151 ms
64 bytes from 10.0.0.64: icmp_seq=7 ttl=64 time=162 ms
64 bytes from 10.0.0.64: icmp_seq=8 ttl=64 time=148 ms
64 bytes from 10.0.0.64: icmp_seq=9 ttl=64 time=151 ms
64 bytes from 10.0.0.64: icmp_seq=10 ttl=64 time=144 ms

--- 10.0.0.64 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9021ms
rtt min/avg/max/mdev = 139.684/148.949/162.976/6.475 ms
root@mininet-vm:~# iperf -c 10.0.0.64
-----
Client connecting to 10.0.0.64, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[177] local 10.0.0.1 port 49066 connected with 10.0.0.64 port 5001
[ ID] Interval          Transfer      Bandwidth
[177] 0.0-10.4 sec      12.0 MBytes  9.66 Mbits/sec
```

- The amount of data transfer is low (12.0MB) compared to previous condition (45.3 MB) when we limit the bandwidth and set delay.

C Section

➤ Node h64

```

Node: h64
[179] 1154.00-1155.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1155.00-1156.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1156.00-1157.00 sec 1.14 MBytes 9.54 Mbits/sec
[179] 1157.00-1158.00 sec 1.14 MBytes 9.53 Mbits/sec
[179] 1158.00-1159.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1159.00-1160.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1160.00-1161.00 sec 1.14 MBytes 9.54 Mbits/sec
[179] 1161.00-1162.00 sec 1.14 MBytes 9.54 Mbits/sec
[179] 1162.00-1163.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1163.00-1164.00 sec 1.14 MBytes 9.53 Mbits/sec
[179] 1164.00-1165.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1165.00-1166.00 sec 1.13 MBytes 9.47 Mbits/sec
[179] 1166.00-1167.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1167.00-1168.00 sec 1.14 MBytes 9.52 Mbits/sec
[179] 1168.00-1169.00 sec 1.14 MBytes 9.57 Mbits/sec
[179] 1169.00-1170.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1170.00-1171.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1171.00-1172.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1172.00-1173.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1173.00-1174.00 sec 1.14 MBytes 9.51 Mbits/sec
[179] 1174.00-1175.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1175.00-1176.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1176.00-1177.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1177.00-1178.00 sec 1.14 MBytes 9.53 Mbits/sec
[179] 1178.00-1179.00 sec 1.12 MBytes 9.43 Mbits/sec
[179] 1179.00-1180.00 sec 1.14 MBytes 9.54 Mbits/sec
[179] 1180.00-1181.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1181.00-1182.00 sec 1.14 MBytes 9.57 Mbits/sec
[179] 1182.00-1183.00 sec 1.14 MBytes 9.55 Mbits/sec
[179] 1183.00-1184.00 sec 1.14 MBytes 9.52 Mbits/sec
[179] 1184.00-1185.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1185.00-1186.00 sec 1.14 MBytes 9.52 Mbits/sec
[179] 1186.00-1187.00 sec 1.14 MBytes 9.56 Mbits/sec
[179] 1187.00-1187.05 sec 58.0 KBytes 9.12 Mbits/sec
-----
[ ID] Interval          Transfer      Bandwidth    Retr
[179] 0.00-1187.05 sec 1.22 GBytes  8.80 Mbits/sec    1
[179] 0.00-1187.05 sec 1.21 GBytes  8.74 Mbits/sec
-----
Server listening on 5201
-----

```


➤ Node h1

```

"Node: h1"
[178] 1150.00-1151.00 sec 0.00 Bytes 0.00 bits/sec 0 2.78 MBytes
[178] 1151.00-1152.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.86 MBytes
[178] 1152.00-1153.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.92 MBytes
[178] 1153.00-1154.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.96 MBytes
[178] 1154.00-1155.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.00 MBytes
[178] 1155.00-1156.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.02 MBytes
[178] 1156.00-1157.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.04 MBytes
[178] 1157.00-1158.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.05 MBytes
[178] 1158.00-1159.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.06 MBytes
[178] 1159.00-1160.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.07 MBytes
[178] 1160.00-1161.00 sec 0.00 Bytes 0.00 bits/sec 0 3.08 MBytes
[178] 1161.00-1162.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.10 MBytes
[178] 1162.00-1163.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.05 MBytes
[178] 1163.00-1164.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.71 MBytes
[178] 1164.00-1165.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.37 MBytes
[178] 1165.00-1166.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.23 MBytes
[178] 1166.00-1167.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.36 MBytes
[178] 1167.00-1168.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.49 MBytes
[178] 1168.00-1169.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.61 MBytes
[178] 1169.00-1170.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.72 MBytes
[178] 1170.00-1171.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.82 MBytes
[178] 1171.00-1172.00 sec 0.00 Bytes 0.00 bits/sec 0 2.89 MBytes
[178] 1172.00-1173.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.96 MBytes
[178] 1173.00-1174.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.00 MBytes
[178] 1174.00-1175.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.04 MBytes
[178] 1175.00-1176.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.06 MBytes
[178] 1176.00-1177.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.08 MBytes
[178] 1177.00-1178.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.09 MBytes
[178] 1178.00-1179.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.10 MBytes
[178] 1179.00-1180.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.11 MBytes
[178] 1180.00-1181.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.13 MBytes
[178] 1181.00-1182.00 sec 0.00 Bytes 0.00 bits/sec 0 3.14 MBytes
[178] 1182.00-1183.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.85 MBytes
[178] 1183.00-1184.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.51 MBytes
[178] 1184.00-1185.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.21 MBytes
[178] 1185.00-1185.08 sec 1.25 MBytes 142 Mbits/sec 0 2.22 MBytes
-----
[ ID] Interval          Transfer      Bandwidth    Retr
[178] 0.00-1185.08 sec 1.22 GBytes 8.82 Mbits/sec 1
[178] 0.00-1185.08 sec 1.21 GBytes 8.76 Mbits/sec
iperf Done.

```

- While the data transfer is same for previous and current scenario but there is reduction in bandwidth.

D section

➤ Receiver

```

Node: h1
Error lines = 0
-----
root@mininet-vm:~# ITGDec sender.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
-----
Flow number: 1
From 10.0.0.1:55484
To 10.0.0.64:8999
-----
Total time = 1.998824 s
Total packets = 1010
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 517120
Average bitrate = 2069.696982 Kbit/s
Average packet rate = 505.297115 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0.000000 pkt
-----
***** TOTAL RESULTS *****
-----
Number of flows = 1
Total time = 1.998824 s
Total packets = 1010
Minimum delay = 0.000000 s
Maximum delay = 0.000000 s
Average delay = 0.000000 s
Average jitter = 0.000000 s
Delay standard deviation = 0.000000 s
Bytes received = 517120
Average bitrate = 2069.696982 Kbit/s
Average packet rate = 505.297115 pkt/s
Packets dropped = 0 (0.00 %)
Average loss-burst size = 0 pkt
Error lines = 0
-----

```

➤ Sender

```

"Node: h1"
Finished sending packets of flow ID: 1

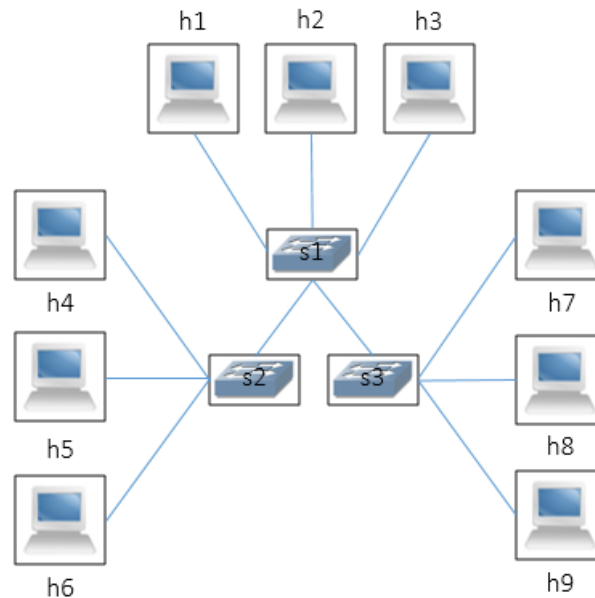
root@mininet-vm:~# ITGDec receiver.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
/-----
Flow number: 1
From 10.0.0.1:58638
To   10.0.0.64:8999
-----
Total time           =      1.983311 s
Total packets        =           836
Minimum delay        =      0.061953 s
Maximum delay        =      0.081412 s
Average delay        =      0.067515 s
Average jitter       =      0.001825 s
Delay standard deviation =    0.004056 s
Bytes received       =      428032
Average bitrate      =    1726.535072 Kbit/s
Average packet rate  =    421.517352 pkt/s
Packets dropped      =           0 (0.00 %)
Average loss-burst size =    0.000000 pkt
-----

***** TOTAL RESULTS *****
-----
Number of flows      =           1
Total time           =      1.983311 s
Total packets        =           836
Minimum delay        =      0.061953 s
Maximum delay        =      0.081412 s
Average delay        =      0.067515 s
Average jitter       =      0.001825 s
Delay standard deviation =    0.004056 s
Bytes received       =      428032
Average bitrate      =    1726.535072 Kbit/s
Average packet rate  =    421.517352 pkt/s
Packets dropped      =           0 (0.00 %)
Average loss-burst size =           0 pkt
Error lines          =           0
-----

```

- The average delay is increased compared to previous condition as we are setting delay and bandwidth.

Q.3). Use the provided 'incomplete' code (in the box below) to create the topology below by adding the links using the corresponding code snippets. To add a link, use `"self.addLink('s1', 's2')"` This command creates a link between s1 to s2.



a) Verify the network configuration, elements, and connectivity between all the hosts.

➤ Create topology

```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo python ./ques3.py
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3
*** Adding links:
(10ms delay) (10ms delay) (s1, h1) (10ms delay) (10ms delay) (s1, h2) (10ms delay) (10ms delay) (s1, h3) (10ms delay) (10ms delay) (s1, s2) (10ms delay) (10ms delay) (s1, s3) (10ms delay) (10ms delay) (s2, h4) (10ms delay) (10ms delay) (s2, h5) (10ms delay) (10ms delay) (s2, h6) (10ms delay) (10ms delay) (s3, h7) (10ms delay) (10ms delay) (s3, h8) (10ms delay) (10ms delay) (s3, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ... (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay)
*** Starting CLI:
mininet>
  
```

➤ Start Pox controller

```
mininet@mininet-vm: ~/pox
File Edit View Search Terminal Help
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py pox.forwarding.l2_learning openflow.spanning_
tree
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
WARNING:core:Still waiting on 1 component(s)
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
```

➤ Display Elements and test connectivity

```
mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth4
h3 h3-eth0:s1-eth5
h4 h4-eth0:s2-eth2
h5 h5-eth0:s2-eth3
h6 h6-eth0:s2-eth4
h7 h7-eth0:s3-eth2
h8 h8-eth0:s3-eth3
h9 h9-eth0:s3-eth4
s1 lo: s1-eth1:s2-eth1 s1-eth2:s3-eth1 s1-eth3:h1-eth0 s1-eth4:h2-eth0 s1-eth5:
h3-eth0
s2 lo: s2-eth1:s1-eth1 s2-eth2:h4-eth0 s2-eth3:h5-eth0 s2-eth4:h6-eth0
s3 lo: s3-eth1:s1-eth2 s3-eth2:h7-eth0 s3-eth3:h8-eth0 s3-eth4:h9-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 h9 s1 s2 s3
```


b) Find the average RTT from h4 to h9 for 40 pings.

```
mininet> h4 ping -c 40 h9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=125 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=107 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=97.8 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=93.8 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=94.6 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=86.6 ms
64 bytes from 10.0.0.9: icmp_seq=7 ttl=64 time=87.3 ms
64 bytes from 10.0.0.9: icmp_seq=8 ttl=64 time=90.8 ms
64 bytes from 10.0.0.9: icmp_seq=9 ttl=64 time=1063 ms
64 bytes from 10.0.0.9: icmp_seq=10 ttl=64 time=89.0 ms
64 bytes from 10.0.0.9: icmp_seq=11 ttl=64 time=95.4 ms
64 bytes from 10.0.0.9: icmp_seq=12 ttl=64 time=94.8 ms
64 bytes from 10.0.0.9: icmp_seq=13 ttl=64 time=90.3 ms
64 bytes from 10.0.0.9: icmp_seq=14 ttl=64 time=87.3 ms
64 bytes from 10.0.0.9: icmp_seq=15 ttl=64 time=87.2 ms
64 bytes from 10.0.0.9: icmp_seq=16 ttl=64 time=103 ms
64 bytes from 10.0.0.9: icmp_seq=17 ttl=64 time=90.8 ms
64 bytes from 10.0.0.9: icmp_seq=18 ttl=64 time=99.3 ms
64 bytes from 10.0.0.9: icmp_seq=19 ttl=64 time=107 ms
64 bytes from 10.0.0.9: icmp_seq=20 ttl=64 time=84.6 ms
64 bytes from 10.0.0.9: icmp_seq=21 ttl=64 time=98.5 ms
64 bytes from 10.0.0.9: icmp_seq=30 ttl=64 time=92.8 ms
64 bytes from 10.0.0.9: icmp_seq=31 ttl=64 time=86.8 ms
64 bytes from 10.0.0.9: icmp_seq=32 ttl=64 time=89.7 ms
64 bytes from 10.0.0.9: icmp_seq=33 ttl=64 time=90.0 ms
64 bytes from 10.0.0.9: icmp_seq=34 ttl=64 time=100 ms
64 bytes from 10.0.0.9: icmp_seq=35 ttl=64 time=93.5 ms
64 bytes from 10.0.0.9: icmp_seq=36 ttl=64 time=99.1 ms
64 bytes from 10.0.0.9: icmp_seq=37 ttl=64 time=96.1 ms
64 bytes from 10.0.0.9: icmp_seq=38 ttl=64 time=98.7 ms
64 bytes from 10.0.0.9: icmp_seq=39 ttl=64 time=98.4 ms
64 bytes from 10.0.0.9: icmp_seq=40 ttl=64 time=98.2 ms

--- 10.0.0.9 ping statistics ---
40 packets transmitted, 40 received, 0% packet loss, time 39305ms
rtt min/avg/max/mdev = 84.620/96.583/134.380/10.186 ms
mininet> █
```


c). Repeat 40 pings from h4 to h9 after increasing the congestion by sending data from h1 to h8, and h7 to h2 for 4 minutes simultaneously (Use IPERF). Take two screenshots (During Iperf data transfer and after Iperf data transfer) such that all windows (h1, h8, h7 and h2) are visible.

➤ Iperf during data transfer

```

"Node: h7"
root@mininet-vm:~# iperf -t 240 -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 31] local 10.0.0.7 port 54664 connected with 10.0.0.2 port 5001

"Node: h1"
root@mininet-vm:~# iperf -t 240 -c 10.0.0.8
-----
Client connecting to 10.0.0.8, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 31] local 10.0.0.1 port 34542 connected with 10.0.0.8 port 5001

"Node: h2"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 32] local 10.0.0.2 port 5001 connected with 10.0.0.7 port 54664

"Node: h8"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 32] local 10.0.0.8 port 5001 connected with 10.0.0.1 port 34542

mininet@mininet-vm: ~
File Edit View Search Terminal Help
64 bytes from 10.0.0.9: icmp_seq=36 ttl=64 time=83.2 ms
64 bytes from 10.0.0.9: icmp_seq=37 ttl=64 time=82.8 ms
64 bytes from 10.0.0.9: icmp_seq=38 ttl=64 time=83.2 ms
64 bytes from 10.0.0.9: icmp_seq=39 ttl=64 time=83.3 ms
64 bytes from 10.0.0.9: icmp_seq=40 ttl=64 time=82.4 ms

--- 10.0.0.9 ping statistics ---
40 packets transmitted, 40 received, 0% packet loss, time 39679ms
rtt min/avg/max/mdev = 82.418/132.577/1448.965/220.152 ms, pipe 2
mininet>

```

➤ Iperf after data transfer

```

"Node: h7"
root@mininet-vm:~# iperf -t 240 -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 31] local 10.0.0.7 port 54664 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 31] 0.0-241.2 sec  15.3 GBytes  545 Mbits/sec
root@mininet-vm:~#

"Node: h1"
root@mininet-vm:~# iperf -t 240 -c 10.0.0.8
-----
Client connecting to 10.0.0.8, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 31] local 10.0.0.1 port 34542 connected with 10.0.0.8 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 31] 0.0-240.0 sec  19.4 GBytes  695 Mbits/sec
root@mininet-vm:~#

"Node: h2"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 32] local 10.0.0.2 port 5001 connected with 10.0.0.7 port 54664
[ ID] Interval      Transfer    Bandwidth
[ 32] 0.0-241.4 sec  15.3 GBytes  544 Mbits/sec
root@mininet-vm:~#

"Node: h8"
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 32] local 10.0.0.8 port 5001 connected with 10.0.0.1 port 34542
[ ID] Interval      Transfer    Bandwidth
[ 32] 0.0-240.0 sec  19.4 GBytes  695 Mbits/sec
root@mininet-vm:~#

mininet@mininet-vm: ~
File Edit View Search Terminal Help
64 bytes from 10.0.0.9: icmp_seq=36 ttl=64 time=93.7 ms
64 bytes from 10.0.0.9: icmp_seq=37 ttl=64 time=89.6 ms
64 bytes from 10.0.0.9: icmp_seq=38 ttl=64 time=96.0 ms
64 bytes from 10.0.0.9: icmp_seq=39 ttl=64 time=91.0 ms
64 bytes from 10.0.0.9: icmp_seq=40 ttl=64 time=88.1 ms

--- 10.0.0.9 ping statistics ---
40 packets transmitted, 40 received, 0% packet loss, time 39742ms
rtt min/avg/max/mdev = 83.844/157.249/2149.072/326.361 ms, pipe 2
mininet>

```

d) Compare the average RTT values from h4 to h9 with (part b.) and without congestion (part c.)

- Average RTT before congestion=**132.577 ms**
- Average RTT after congestion=**15.249 ms**

e) Ping 4 times each from h1 to h4, h1 to h7, h4 to h7. What does the RTT tell you about the number of hops? Try to find the path they take.

```
mininet@mininet-vm: ~  
File Edit View Search Terminal Help  
mininet> h1 ping -c 4 h4  
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.  
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=67.4 ms  
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=70.8 ms  
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=67.7 ms  
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=65.4 ms  
  
--- 10.0.0.4 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3008ms  
rtt min/avg/max/mdev = 65.470/67.880/70.830/1.943 ms  
mininet> h1 ping -c 4 h7  
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.  
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=147 ms  
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=69.3 ms  
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=72.3 ms  
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=66.6 ms  
  
--- 10.0.0.7 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3008ms  
rtt min/avg/max/mdev = 66.647/88.975/147.479/33.839 ms  
mininet> h4 ping -c 4 h7  
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.  
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=151 ms  
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=97.5 ms  
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=93.9 ms  
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=92.9 ms  
  
--- 10.0.0.7 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3005ms  
rtt min/avg/max/mdev = 92.985/109.031/151.712/24.702 ms
```

In case of,

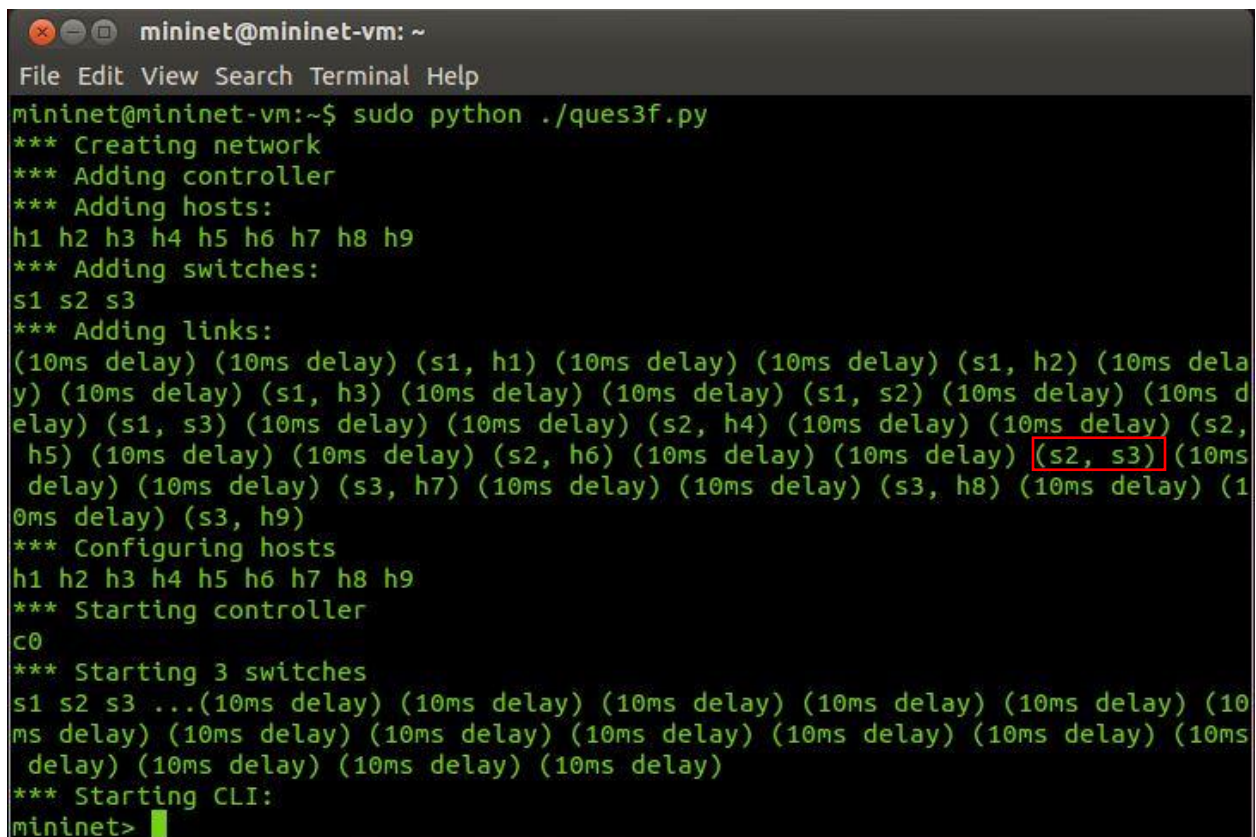
H1 to H4---- Average RTT = 68 ms, As, there are 3 links between H1 and H4, ideal RTT should be 60 ms (Delay in each link is 10 ms)

H1 to H7---- Average RTT = 89 ms, As, there are 3 links between H1 and H7, ideal RTT should be 60 ms (Delay in each link is 10 ms)

H4 to H7---- Average RTT = 109 ms, As, there are 4 links between H1 and H4, ideal RTT should be 80 ms (Delay in each link is 10 ms)

f) Now add a link between s2 and s3 in your code. Try to perform an all host connectivity test using pingall. What do you observe? What is the reason for it?

➤ Adding a link between s2 and s3

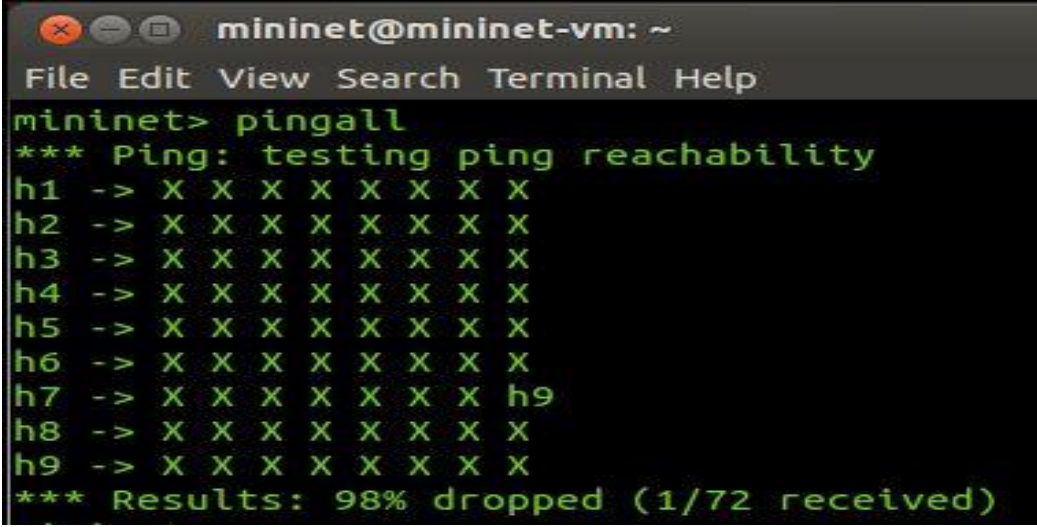


```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo python ./ques3f.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3
*** Adding links:
(10ms delay) (10ms delay) (s1, h1) (10ms delay) (10ms delay) (s1, h2) (10ms delay) (10ms delay) (s1, h3) (10ms delay) (10ms delay) (s1, s2) (10ms delay) (10ms delay) (s1, s3) (10ms delay) (10ms delay) (s2, h4) (10ms delay) (10ms delay) (s2, h5) (10ms delay) (10ms delay) (s2, h6) (10ms delay) (10ms delay) (s2, s3) (10ms delay) (10ms delay) (s3, h7) (10ms delay) (10ms delay) (s3, h8) (10ms delay) (10ms delay) (s3, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ... (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay) (10ms delay)
*** Starting CLI:
mininet>

```

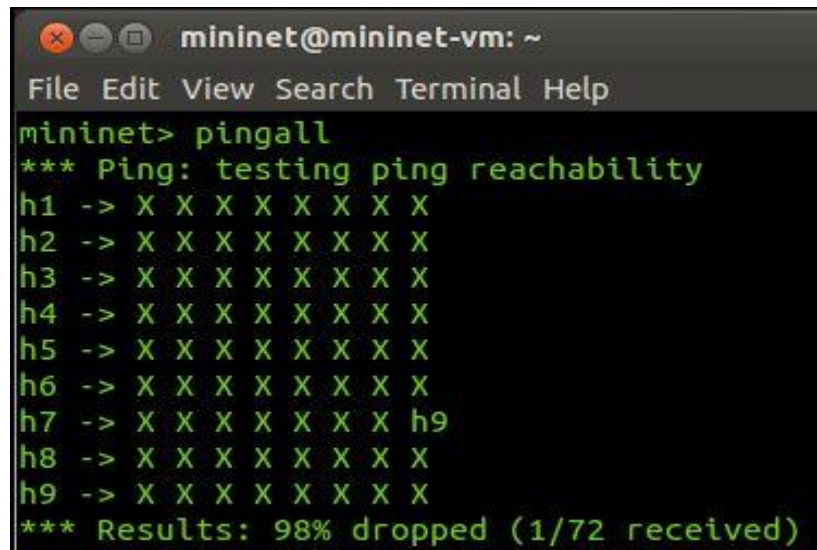
- As a new link is introduced in the topology, loops are formed since there is no convergence taking place as spanning tree protocol is not enabled so the connectivity test amongst the hosts fail



```
mininet@mininet-vm: ~  
File Edit View Search Terminal Help  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> X X X X X X X X  
h2 -> X X X X X X X X  
h3 -> X X X X X X X X  
h4 -> X X X X X X X X  
h5 -> X X X X X X X X  
h6 -> X X X X X X X X  
h7 -> X X X X X X X h9  
h8 -> X X X X X X X X  
h9 -> X X X X X X X X  
*** Results: 98% dropped (1/72 received)
```

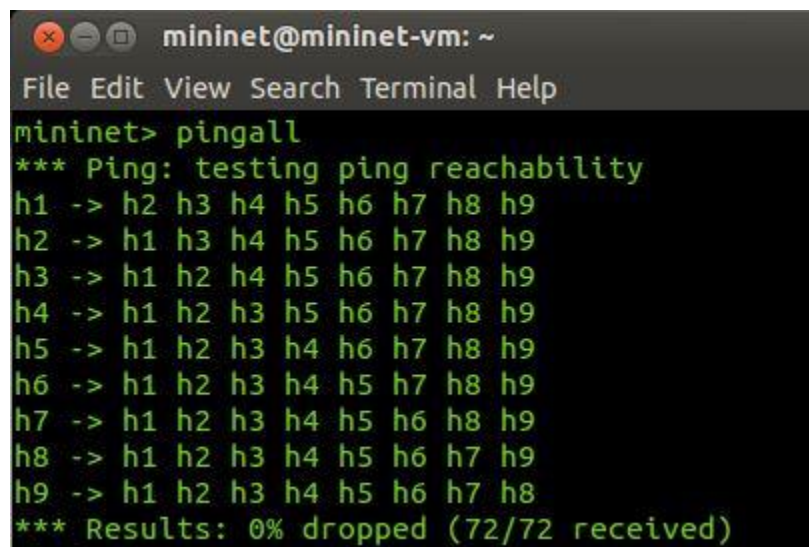

g) Now stop pox controller as l2 learning by going back to the terminal on which POX is running and pressing 'Ctrl' + 'C'. Start pox controller with spanning tree enabled by using the command '~/pox/pox.py samples.spanning_tree' on Linux terminal. Wait 50 sec for the network to converge do a pingall and then repeat part e. Which link is blocked by Spanning tree?

- Pox controller stopped



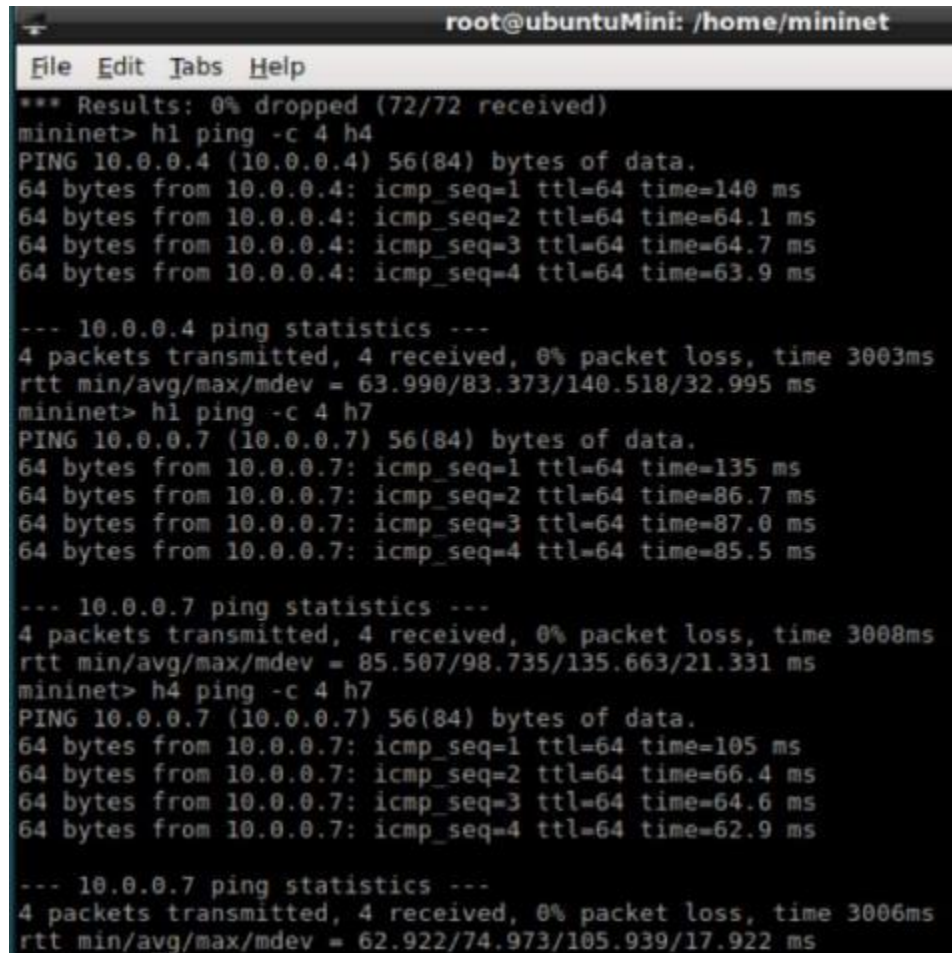
```
mininet@mininet-vm: ~  
File Edit View Search Terminal Help  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> X X X X X X X X  
h2 -> X X X X X X X X  
h3 -> X X X X X X X X  
h4 -> X X X X X X X X  
h5 -> X X X X X X X X  
h6 -> X X X X X X X X  
h7 -> X X X X X X X h9  
h8 -> X X X X X X X X  
h9 -> X X X X X X X X  
*** Results: 98% dropped (1/72 received)
```

- Pox controller started with spanning tree



```
mininet@mininet-vm: ~  
File Edit View Search Terminal Help  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> h2 h3 h4 h5 h6 h7 h8 h9  
h2 -> h1 h3 h4 h5 h6 h7 h8 h9  
h3 -> h1 h2 h4 h5 h6 h7 h8 h9  
h4 -> h1 h2 h3 h5 h6 h7 h8 h9  
h5 -> h1 h2 h3 h4 h6 h7 h8 h9  
h6 -> h1 h2 h3 h4 h5 h7 h8 h9  
h7 -> h1 h2 h3 h4 h5 h6 h8 h9  
h8 -> h1 h2 h3 h4 h5 h6 h7 h9  
h9 -> h1 h2 h3 h4 h5 h6 h7 h8  
*** Results: 0% dropped (72/72 received)
```


- Repeat part e)



```
root@ubuntuMini: /home/mininet
File Edit Tabs Help
*** Results: 0% dropped (72/72 received)
mininet> h1 ping -c 4 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=140 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=64.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=64.7 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=63.9 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 63.990/83.373/140.518/32.995 ms
mininet> h1 ping -c 4 h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=135 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=86.7 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=87.0 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=85.5 ms

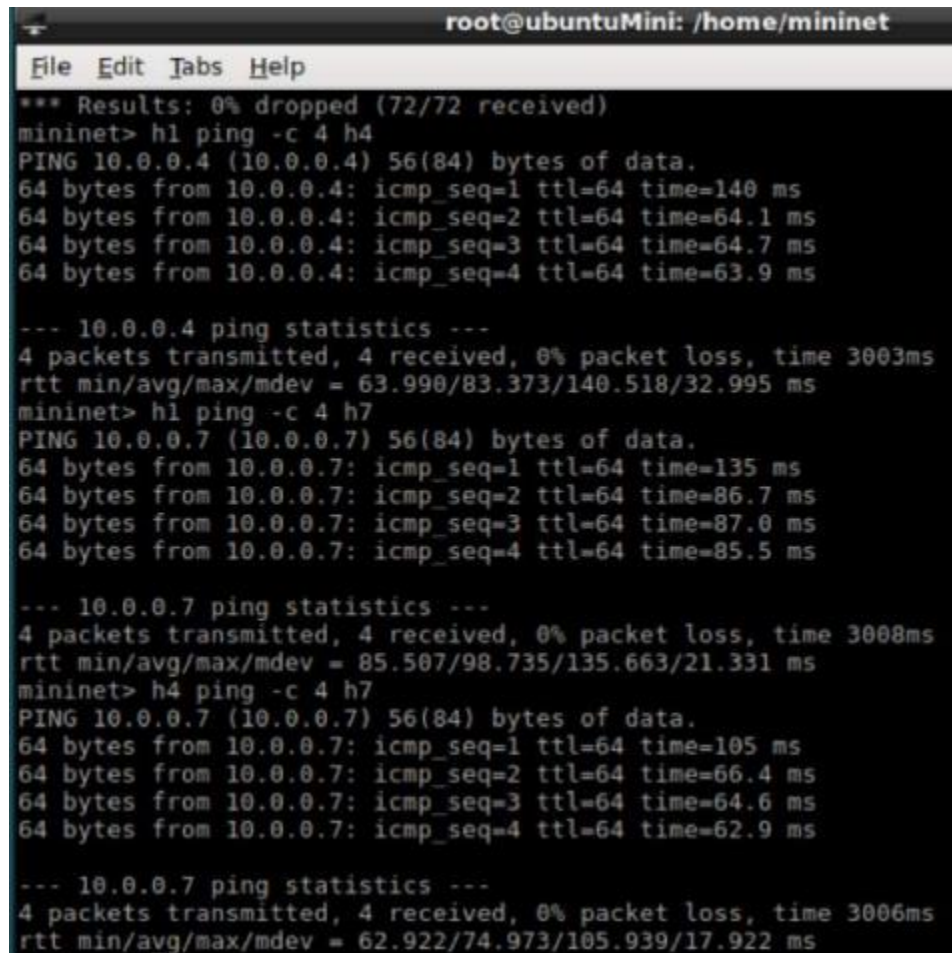
--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 85.507/98.735/135.663/21.331 ms
mininet> h4 ping -c 4 h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=105 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=66.4 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=64.6 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=62.9 ms

--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 62.922/74.973/105.939/17.922 ms
```

- The link between s2 and s3 is blocked by spanning tree since there was a loop generated when a link was added between s2 and s3 and to get rid of this loop spanning-tree blocked one of the ports on the switch

h) Bring down any other link (that is not blocked by spanning tree) and repeat part e. What do you observe?

- Removing Link between s1 and h1
- The link between s2 and s3 which was blocked earlier is now unblocked and running as the other link is blocked after convergence and the function of spanning tree is achieved to provide a loop-free network



```
root@ubuntuMini: /home/mininet
File Edit Tabs Help
*** Results: 0% dropped (72/72 received)
mininet> h1 ping -c 4 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=140 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=64.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=64.7 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=63.9 ms

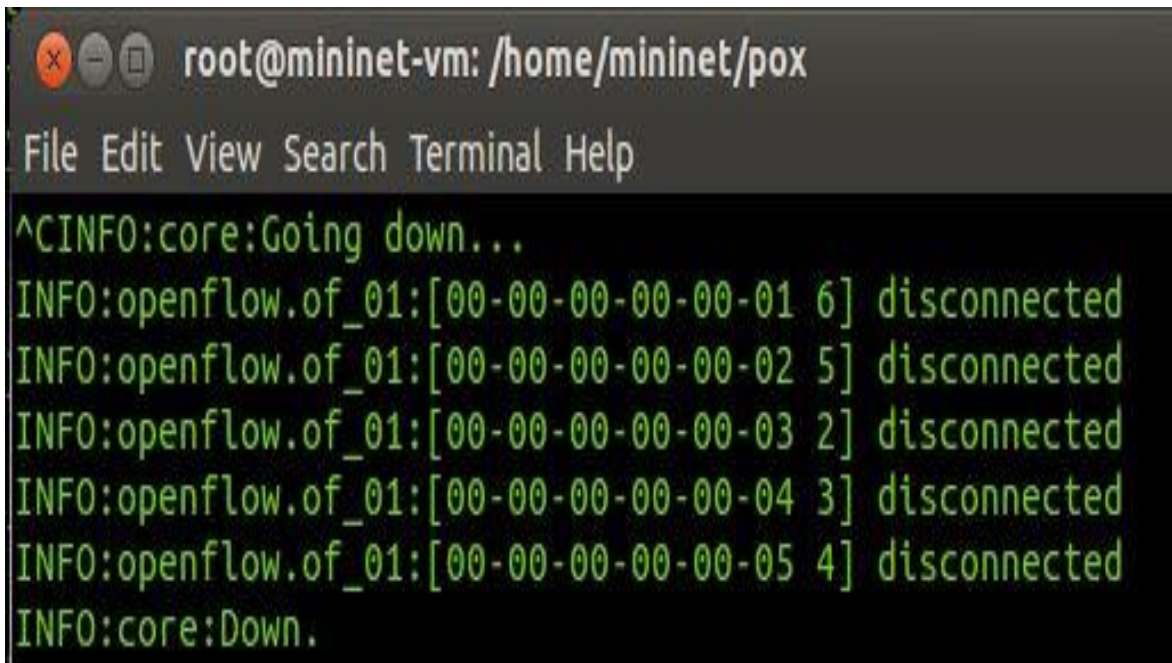
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 63.990/83.373/140.518/32.995 ms
mininet> h1 ping -c 4 h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=135 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=86.7 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=87.0 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=85.5 ms

--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 85.507/98.735/135.663/21.331 ms
mininet> h4 ping -c 4 h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=105 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=66.4 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=64.6 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=62.9 ms

--- 10.0.0.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 62.922/74.973/105.939/17.922 ms
```

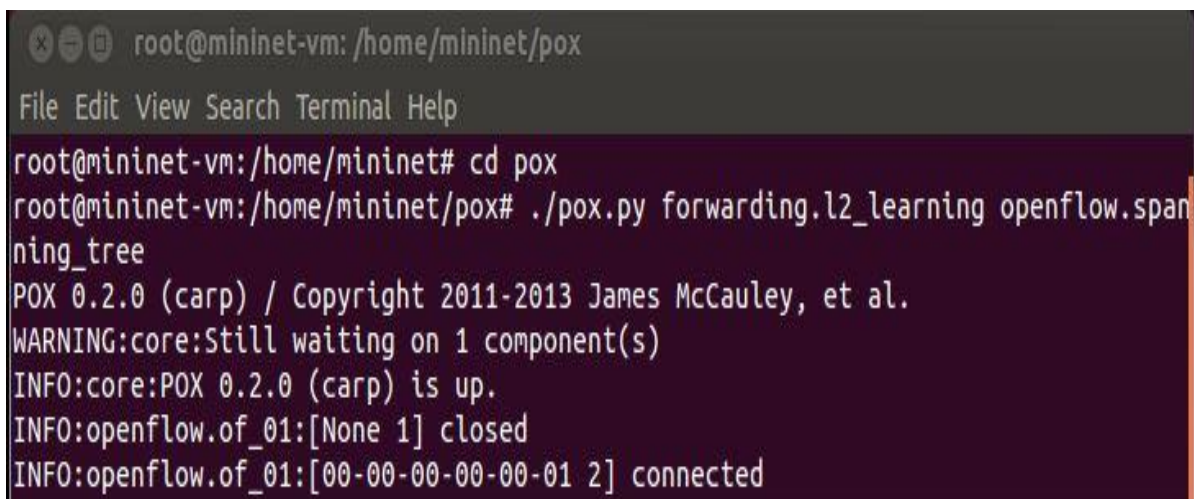
i) Once done kill the controller by going back to the terminal and pressing ‘Ctrl’ + ‘C’ and bring the POX controller back on as a L2 forwarding switch by using the command ‘~/pox/pox.py forwarding.l2_learning’

➤ Kill Controller



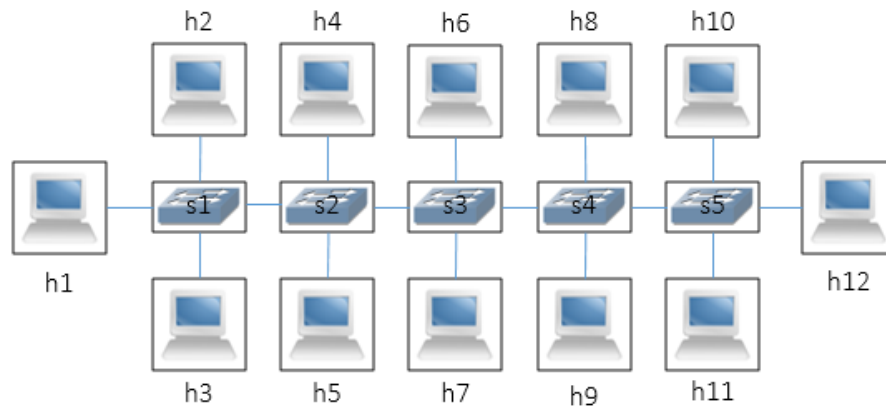
```
root@mininet-vm: /home/mininet/pox
File Edit View Search Terminal Help
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 6] disconnected
INFO:openflow.of_01:[00-00-00-00-00-02 5] disconnected
INFO:openflow.of_01:[00-00-00-00-00-03 2] disconnected
INFO:openflow.of_01:[00-00-00-00-00-04 3] disconnected
INFO:openflow.of_01:[00-00-00-00-00-05 4] disconnected
INFO:core:Down.
```

➤ Bring back Pox controller



```
root@mininet-vm: /home/mininet/pox
File Edit View Search Terminal Help
root@mininet-vm:/home/mininet# cd pox
root@mininet-vm:/home/mininet/pox# ./pox.py forwarding.l2_learning openflow.spanning_tree
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
WARNING:core:Still waiting on 1 component(s)
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```


Q.4) Create a python script for the topology shown below and enter Performance parameters of Bandwidth = 15, Delay = 5ms, loss = 1 and max queue length 1000. (Performance parameters must be entered along with each link that you create in the script. Use the same code as the previous question, with the only difference being adding a greater number of hosts, a greater number of switches and different links in this question). Also, include a screenshot of the initialization of the network which occurs after you run the command "sudo python ./filename.py"



➤ Topology

```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo python ./q4.py
python: can't open file './q4.py': [Errno 2] No such file or directory
mininet@mininet-vm:~$ gedit it.py
mininet@mininet-vm:~$ sudo python ./it.py
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s1, h1) (15.00Mbit
5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s1, h2) (15.00Mbit 5ms delay 1
% loss) (15.00Mbit 5ms delay 1% loss) (s1, h3) (15.00Mbit 5ms delay 1% loss) (15
.00Mbit 5ms delay 1% loss) (s1, s2) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms
delay 1% loss) (s2, h4) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% l
oss) (s2, h5) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s2, s
3) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, h6) (15.00Mb
it 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, h7) (15.00Mbit 5ms dela
y 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, s4) (15.00Mbit 5ms delay 1% loss)
(15.00Mbit 5ms delay 1% loss) (s4, h8) (15.00Mbit 5ms delay 1% loss) (15.00Mbit
5ms delay 1% loss) (s4, h9) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1
% loss) (s4, s5) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s5

```

```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
oss) (s2, h5) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s2, s
3) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, h6) (15.00Mb
it 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, h7) (15.00Mbit 5ms dela
y 1% loss) (15.00Mbit 5ms delay 1% loss) (s3, s4) (15.00Mbit 5ms delay 1% loss)
(15.00Mbit 5ms delay 1% loss) (s4, h8) (15.00Mbit 5ms delay 1% loss) (15.00Mbit
5ms delay 1% loss) (s4, h9) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1
% loss) (s4, s5) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s5
, h10) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s5, h11) (15
.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (s5, h12)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ... (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (1
5.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1
% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5
ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (1
5.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1
% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5
ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss) (1
5.00Mbit 5ms delay 1% loss) (15.00Mbit 5ms delay 1% loss)

```

a) Check network configuration, elements and connectivity between all its hosts. What do you observe? Provide an explanation for your answer.

```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 X h8 h9 h10 X h12
h2 -> h1 h3 X h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 X h11 h12
h5 -> h1 h2 h3 h4 h6 h7 X h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 X h5 X h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> X h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 X h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 6% dropped (123/132 received)

```



```

mininet@mininet-vm: ~
File Edit View Search Terminal Help

mininet> nodes
available nodes are:
c0 h1 h10 h11 h12 h2 h3 h4 h5 h6 h7 h8 h9 s1 s2 s3 s4 s5
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth4
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth4
h5 h5-eth0:s2-eth3
h6 h6-eth0:s3-eth4
h7 h7-eth0:s3-eth3
h8 h8-eth0:s4-eth2
h9 h9-eth0:s4-eth3
h10 h10-eth0:s5-eth2
h11 h11-eth0:s5-eth3
h12 h12-eth0:s5-eth4
s1 lo: s1-eth1:s2-eth1 s1-eth2:h3-eth0 s1-eth3:h1-eth0 s1-eth4:h2-eth0
s2 lo: s2-eth1:s1-eth1 s2-eth2:s3-eth2 s2-eth3:h5-eth0 s2-eth4:h4-eth0
s3 lo: s3-eth1:s4-eth1 s3-eth2:s2-eth2 s3-eth3:h7-eth0 s3-eth4:h6-eth0
s4 lo: s4-eth1:s3-eth1 s4-eth2:h8-eth0 s4-eth3:h9-eth0 s4-eth4:s5-eth1
s5 lo: s5-eth1:s4-eth4 s5-eth2:h10-eth0 s5-eth3:h11-eth0 s5-eth4:h12-eth0
c0

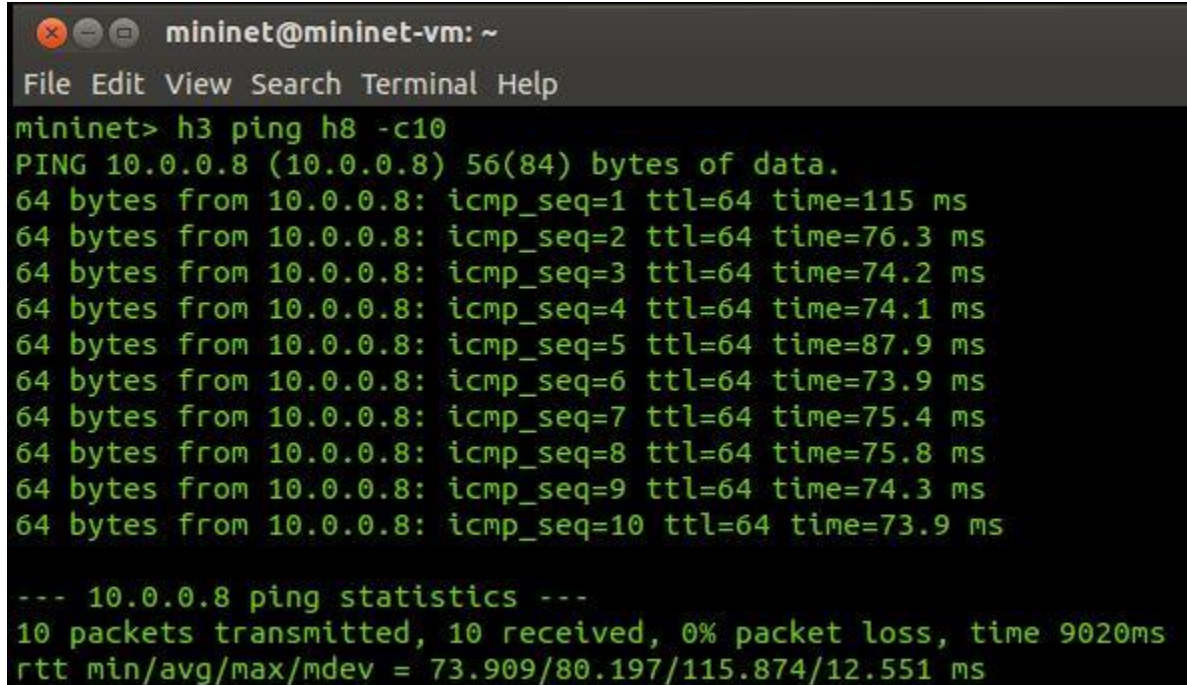
```

Observation: By pinging amongst the hosts it is observed that, about 6% packets are lost since:

h1 to h7, h11 link is down
 h2 to h4 link is down
 h4 to h10 link is down
 h5 to h8 link is down
 h8 to h4, h6 link is down
 h10 to h1 link is down
 h11 to h3 link is down

These links are down due to introduction of loss factor = 1 as mentioned in the question.

b) Find the Min/Avg/Max RTT from h3 to h8 for 10 packets. (Use Ping)

A terminal window titled 'mininet@mininet-vm: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The command 'mininet> h3 ping h8 -c10' is entered. The output shows 10 ping results for 10.0.0.8, each with 64 bytes of data and an ICMP sequence number from 1 to 10. The times are: 115 ms, 76.3 ms, 74.2 ms, 74.1 ms, 87.9 ms, 73.9 ms, 75.4 ms, 75.8 ms, 74.3 ms, and 73.9 ms. Below the results, it shows '--- 10.0.0.8 ping statistics ---' and '10 packets transmitted, 10 received, 0% packet loss, time 9020ms'. The final line shows 'rtt min/avg/max/mdev = 73.909/80.197/115.874/12.551 ms'.

```
mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet> h3 ping h8 -c10
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=115 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=76.3 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=74.2 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=74.1 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=87.9 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=73.9 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=75.4 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=75.8 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=74.3 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=73.9 ms

--- 10.0.0.8 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9020ms
rtt min/avg/max/mdev = 73.909/80.197/115.874/12.551 ms
```

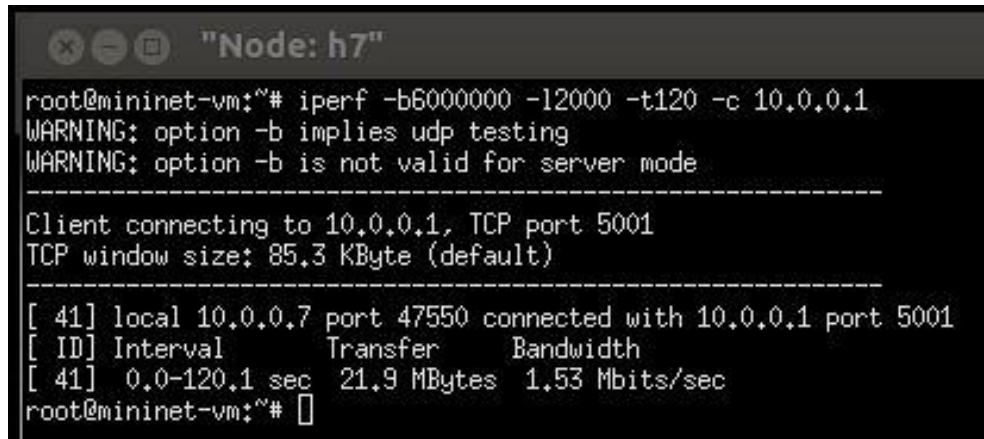
RTT min= 73.909 ms

RTT average= 80.197 ms

RTT max= 115.874 ms

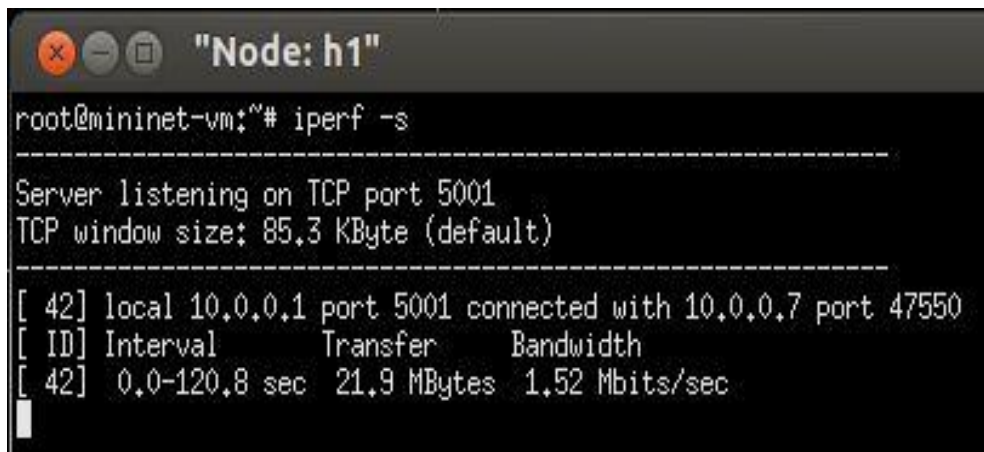
c) Generate data flows for 2 minutes from h7 to h1 with packet length 2000 and bandwidth 6000 KB. Find the maximum/minimum utilization of the link and state the maximum/minimum and total data transferred along this link. (Use IPERF, use -l for packet length and -b for bandwidth). Provide the screenshot of the client.

➤ Node H7 client



```
root@mininet-vm:~# iperf -b6000000 -l2000 -t120 -c 10.0.0.1
WARNING: option -b implies udp testing
WARNING: option -b is not valid for server mode
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 41] local 10.0.0.7 port 47550 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 41] 0.0-120.1 sec  21.9 MBytes  1.53 Mbits/sec
root@mininet-vm:~#
```

➤ Node H1 server



```
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 42] local 10.0.0.1 port 5001 connected with 10.0.0.7 port 47550
[ ID] Interval      Transfer    Bandwidth
[ 42] 0.0-120.8 sec  21.9 MBytes  1.52 Mbits/sec

```

```
"Node: h7"
root@mininet-vm:~# iperf -c 10.0.0.1 -i 10 -t 120 -l 2000 -b 6000000
WARNING: option -b implies udp testing
-----
Client connecting to 10.0.0.1, UDP port 5001
Sending 2000 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 41] local 10.0.0.7 port 54094 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 41] 0.0-10.0 sec   7.15 MBytes 6.00 Mbits/sec
[ 41] 10.0-20.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 20.0-30.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 30.0-40.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 40.0-50.0 sec  7.13 MBytes 5.98 Mbits/sec
[ 41] 50.0-60.0 sec  7.16 MBytes 6.00 Mbits/sec
[ 41] 60.0-70.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 70.0-80.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 80.0-90.0 sec  7.15 MBytes 6.00 Mbits/sec
[ 41] 90.0-100.0 sec 7.15 MBytes 6.00 Mbits/sec
[ 41] 100.0-110.0 sec 7.14 MBytes 5.99 Mbits/sec
[ 41] 110.0-120.0 sec 7.16 MBytes 6.00 Mbits/sec
[ 41] 0.0-120.0 sec 85.8 MBytes 6.00 Mbits/sec
```

Total data transferred= 21.9 MB

Maximum Link Utilization= 6.2 Mbps

Minimum Link Utilization= 301 Kbps

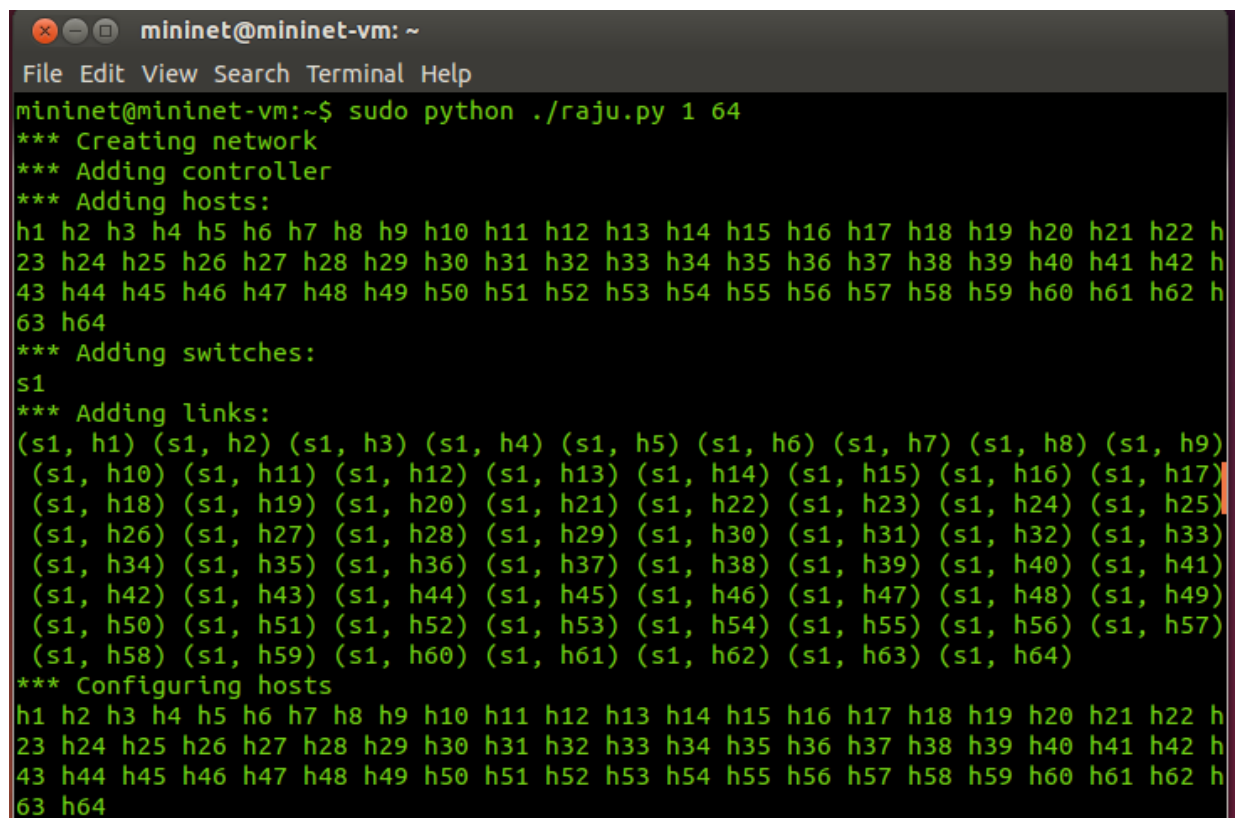
Maximum Data= 351 Kb

Minimum Data= 56 Kb

Q.5) As an admin in a recently expanded company, your boss asks you to implement a network to incorporate 64 hosts on it. Since there is more than one way of doing this, implement 4 separate tree networks, each has a different topology; provide details on each of these topologies and the number of switches initialized in each network architecture. Check average RTT from the first host to every other host and explain which network you think is optimal, taking the cost and number of hosts per switch into consideration. (Include Screenshot)

You may use the code below to create a tree topology and make h1 ping all other hosts and collect the average RTT and store it in a file called “avgResults-depth-fanout.txt”. To run the code just save it in a file and type 'sudo python script_name.py \$depth \$fanout'. where script_name.py is the name you save the script and \$depth and \$fanout being the number corresponding to the depth and fanout required.

➤ Topology Creation



```

mininet@mininet-vm: ~
File Edit View Search Terminal Help
mininet@mininet-vm:~$ sudo python ./raju.py 1 64
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4) (s1, h5) (s1, h6) (s1, h7) (s1, h8) (s1, h9)
(s1, h10) (s1, h11) (s1, h12) (s1, h13) (s1, h14) (s1, h15) (s1, h16) (s1, h17)
(s1, h18) (s1, h19) (s1, h20) (s1, h21) (s1, h22) (s1, h23) (s1, h24) (s1, h25)
(s1, h26) (s1, h27) (s1, h28) (s1, h29) (s1, h30) (s1, h31) (s1, h32) (s1, h33)
(s1, h34) (s1, h35) (s1, h36) (s1, h37) (s1, h38) (s1, h39) (s1, h40) (s1, h41)
(s1, h42) (s1, h43) (s1, h44) (s1, h45) (s1, h46) (s1, h47) (s1, h48) (s1, h49)
(s1, h50) (s1, h51) (s1, h52) (s1, h53) (s1, h54) (s1, h55) (s1, h56) (s1, h57)
(s1, h58) (s1, h59) (s1, h60) (s1, h61) (s1, h62) (s1, h63) (s1, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64

```


➤ **Depth = 1 fanout = 64**

mininet@mininet-vm: ~	mininet@mini	mininet@mininet-vm: ~
File Edit View Search Terminal Help	File Edit View Search	File Edit View Search Terminal Help
Average RTTs are	1.287 ms	1.125 ms
0.702 ms	1.654 ms	1.273 ms
0.602 ms	1.561 ms	0.810 ms
0.904 ms	1.147 ms	1.161 ms
0.557 ms	0.618 ms	1.803 ms
1.577 ms	0.751 ms	1.208 ms
0.899 ms	0.777 ms	1.395 ms
1.921 ms	3.380 ms	1.006 ms
0.522 ms	1.640 ms	1.891 ms
2.246 ms	0.676 ms	1.344 ms
0.760 ms	0.652 ms	0.715 ms
1.160 ms	2.030 ms	1.546 ms
1.460 ms	1.027 ms	1.254 ms
0.341 ms	1.160 ms	1.194 ms
1.659 ms	0.586 ms	1.036 ms
0.757 ms	1.414 ms	0.942 ms
0.701 ms	1.630 ms	0.338 ms
2.408 ms	0.969 ms	0.556 ms
1.030 ms	1.125 ms	0.651 ms
1.167 ms	1.273 ms	0.853 ms
1.419 ms	0.810 ms	0.693 ms
1.276 ms	1.161 ms	2.983 ms
1.181 ms	1.803 ms	
0.920 ms	1.208 ms	

The tentative average RTT for depth=1 and fanout=64 is 1-2 ms.

➤ **Depth = 2 fanout = 8**

mininet@mininet	mininet@mininet-vm: ~
Average RTTs are	7.178 ms
1.101 ms	5.433 ms
4.653 ms	3.183 ms
2.165 ms	3.793 ms
1.979 ms	13.750 ms
1.417 ms	7.147 ms
0.939 ms	9.840 ms
5.463 ms	4.230 ms
10.283 ms	3.150 ms
11.543 ms	8.823 ms
1.851 ms	8.620 ms
1.684 ms	7.656 ms
5.640 ms	13.813 ms
1.696 ms	9.006 ms
15.714 ms	16.186 ms
4.018 ms	1.202 ms
11.153 ms	10.723 ms
10.173 ms	6.285 ms
10.347 ms	11.350 ms
7.173 ms	12.207 ms
10.505 ms	10.617 ms
16.454 ms	8.559 ms
18.381 ms	7.128 ms
9.642 ms	5.210 ms
3.823 ms	5.423 ms
9.384 ms	3.662 ms
4.279 ms	3.642 ms
6.474 ms	9.605 ms
6.607 ms	8.863 ms
7.783 ms	9.309 ms
	2.248 ms
	15.010 ms
	7.216 ms
	1.453 ms
	mininet@mininet-vm:~\$

The tentative average RTT for depth=2 and fanout=8 is 4-10 ms.

➤ Depth = 3 fanout = 4

mininet@min	mininet@mininet-vm: ~	mininet@mininet-vm: ~
File Edit View Search	File Edit View Search Terminal Help	File Edit View Search Terminal Help
Average RTTs are	11.011 ms	8.999 ms
4.138 ms	8.367 ms	17.283 ms
3.321 ms	13.294 ms	16.905 ms
21.689 ms	12.279 ms	10.151 ms
29.144 ms	15.027 ms	10.420 ms
8.398 ms	11.020 ms	12.303 ms
11.782 ms	8.999 ms	15.494 ms
13.601 ms	17.283 ms	11.559 ms
40.952 ms	16.905 ms	9.770 ms
9.492 ms	10.151 ms	9.806 ms
6.054 ms	10.420 ms	9.465 ms
40.393 ms	12.303 ms	11.124 ms
6.837 ms	15.494 ms	14.782 ms
11.099 ms	11.559 ms	11.801 ms
9.135 ms	9.770 ms	10.609 ms
38.961 ms	9.806 ms	17.142 ms
14.003 ms	9.465 ms	15.831 ms
20.610 ms	11.124 ms	10.372 ms
12.584 ms	14.782 ms	15.070 ms
15.794 ms	11.801 ms	8.811 ms
16.019 ms	10.609 ms	17.363 ms
20.434 ms	17.142 ms	16.614 ms
12.825 ms	15.831 ms	13.455 ms
11.273 ms	10.372 ms	20.996 ms
14.015 ms	15.070 ms	20.746 ms
13.021 ms	8.811 ms	13.690 ms
12.362 ms	17.363 ms	10.527 ms
15.783 ms	16.614 ms	10.915 ms
7.961 ms	13.455 ms	
13.296 ms	20.996 ms	

The tentative average RTT for depth=3 and fanout=4 is 10-15 ms.

After observing the average RTT's for all 3 topologies, we can conclude that the average RTT rises with the increase in depth. Each host in the network must communicate to every other host through a single switch so the traversal time reduces. So, a network with depth=1, fanout=64 seems to be an optimal design in terms of cost and the number of hosts.