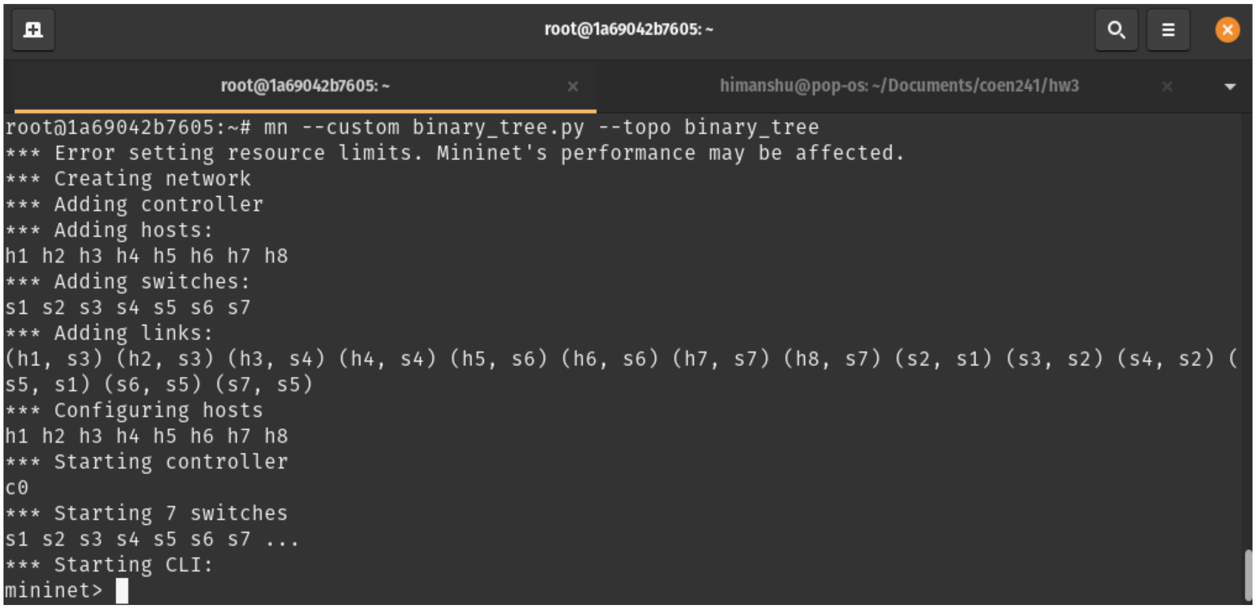
COEN 241 HW 3: Mininet and OpenFlow

Task 1

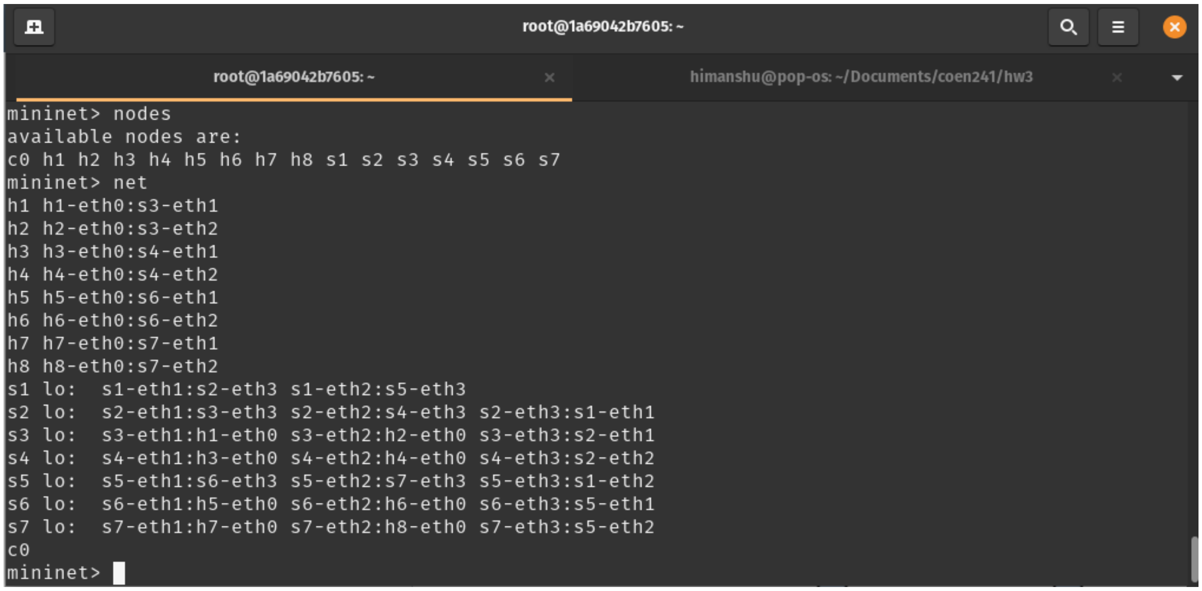
Defining Custom Topology



```
root@1a69042b7605:~# mn --custom binary_tree.py --topo binary_tree
*** Error setting resource limits. Mininet's performance may be affected.
*** 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:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s2, s1) (s3, s2) (s4, s2) (
s5, s1) (s6, s5) (s7, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

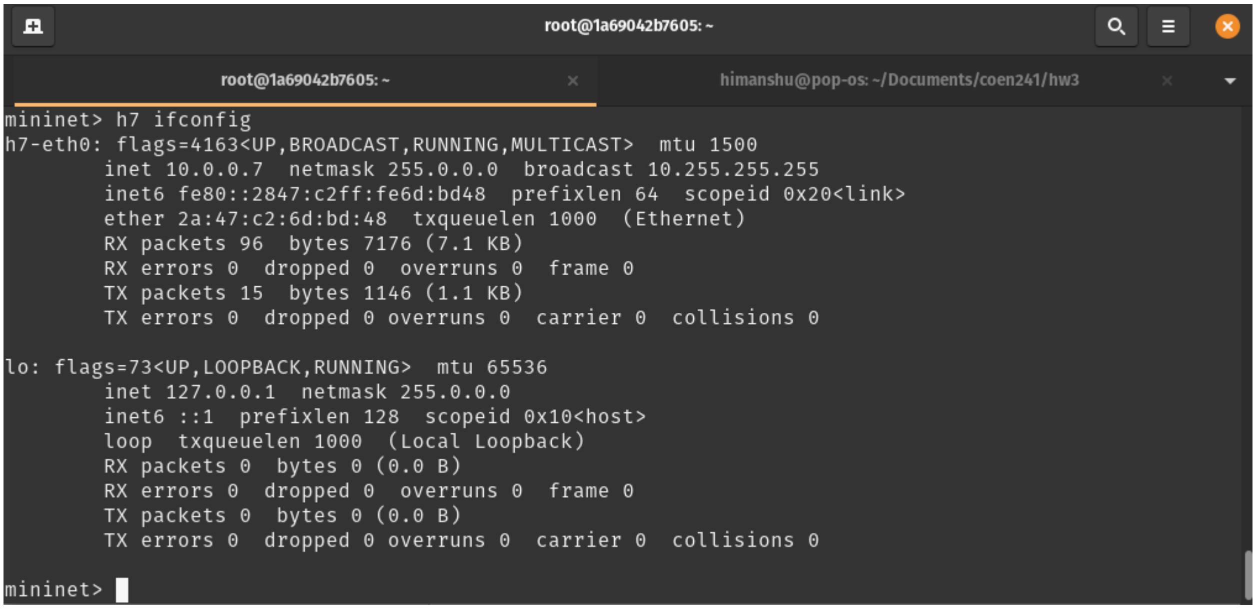
Questions

1. Output of nodes and net



```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
mininet>
```

2. Output of h7 ifconfig



```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::2847:c2ff:fe6d:bd48  prefixlen 64  scopeid 0x20<link>
        ether 2a:47:c2:6d:bd:48  txqueuelen 1000  (Ethernet)
        RX packets 96  bytes 7176 (7.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15  bytes 1146 (1.1 KB)
        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
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> 
```

Task 2

Analyze the “of_tutorial” controller

Questions

1. Draw the function call graph of this controller. For example, once a packet comes to the controller, which function is the first to be called, which one is the second, and so forth?

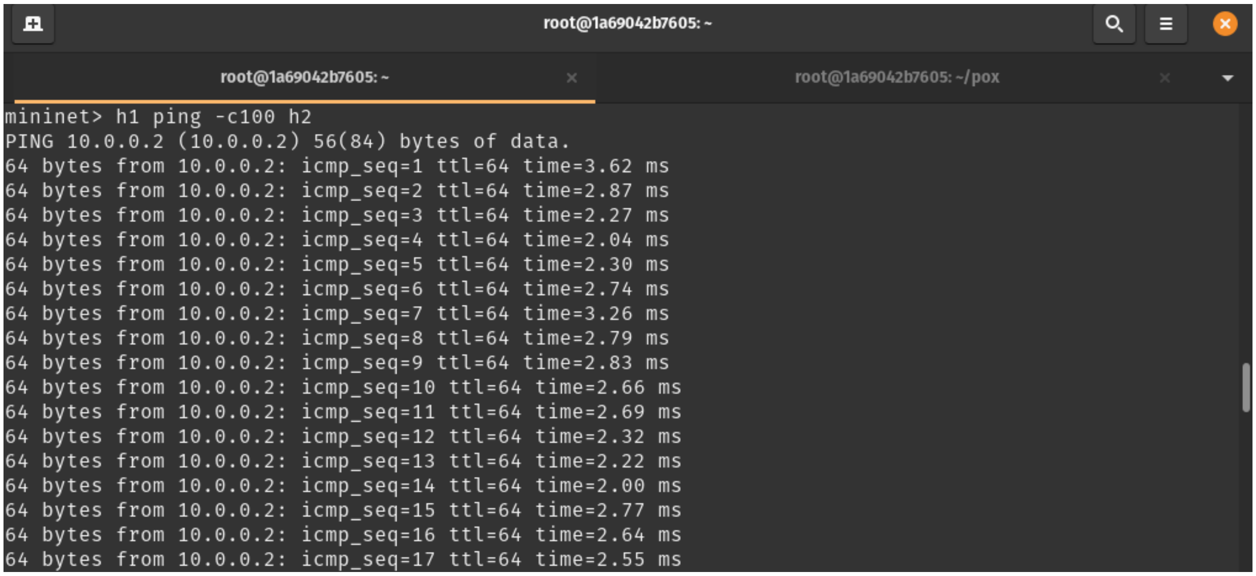
Ans: start_switch is called first.

start_switch —> _handle_PacketIn —> act_like_hub —> resend_packet —> send(msg)

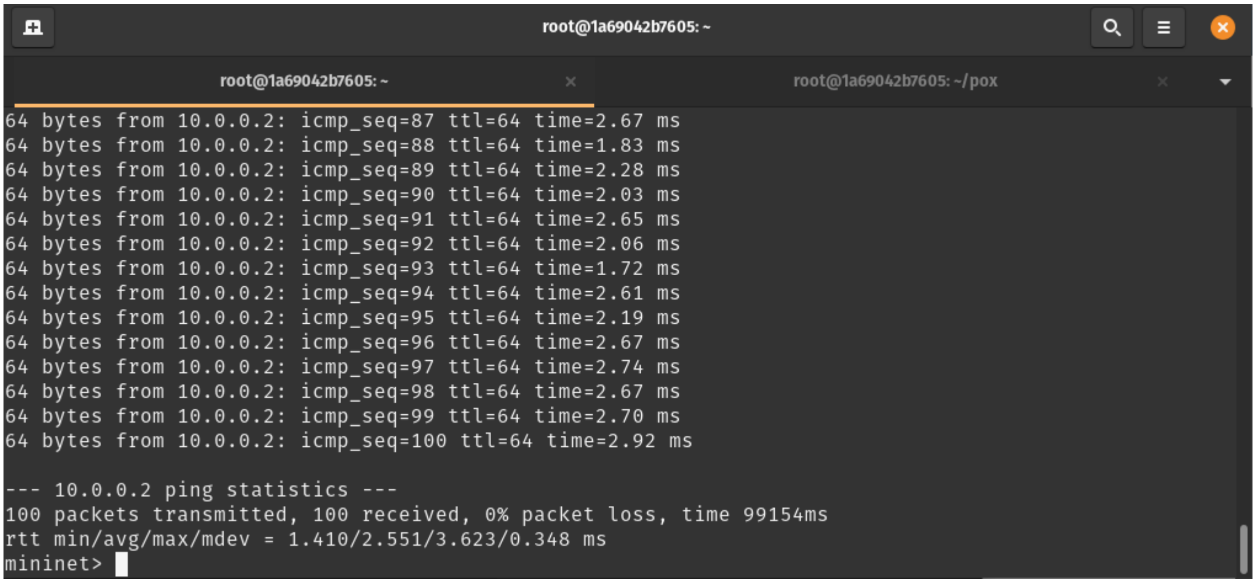
2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

   a. How long does it take (on average) to ping for each case?

   Ans: For h1 ping -c100 h2: it takes 2.551 ms.



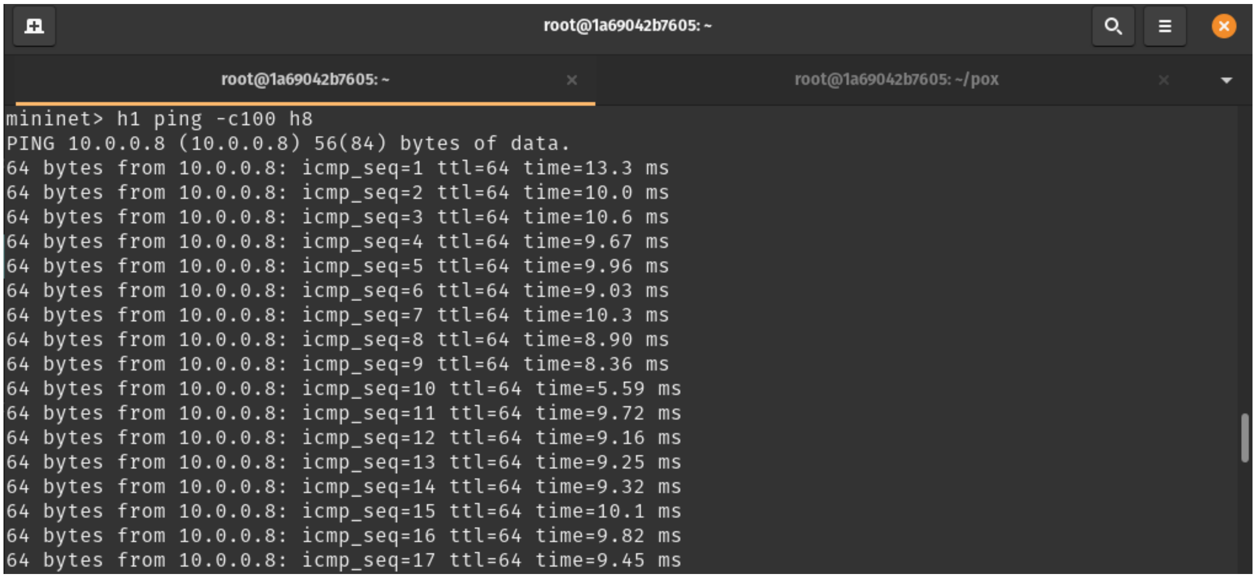
```
mininet> h1 ping -c100 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.62 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.87 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.27 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2.04 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2.30 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=2.74 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=3.26 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2.79 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=2.83 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=2.66 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=2.69 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=2.32 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=2.22 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=2.00 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=2.77 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=2.64 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=2.55 ms
```



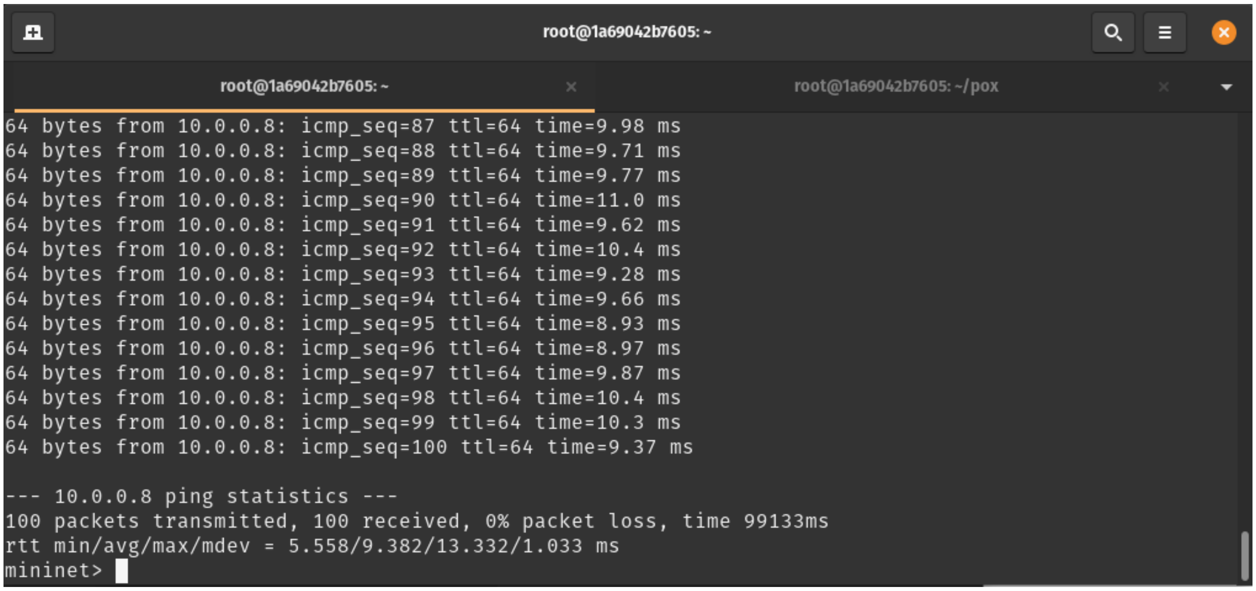
```
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=2.67 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=1.83 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=2.28 ms
64 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=2.03 ms
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=2.65 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=2.06 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=1.72 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=2.61 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=2.19 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=2.67 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=2.74 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=2.67 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=2.70 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=2.92 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99154ms
rtt min/avg/max/mdev = 1.410/2.551/3.623/0.348 ms
mininet> 
```

For h1 ping -c100 h8: it takes 9.382 ms.



```
mininet> h1 ping -c100 h8
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=13.3 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=10.0 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=10.6 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=9.67 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=9.96 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=9.03 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=10.3 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=8.90 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=8.36 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=5.59 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=9.72 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=9.16 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=9.25 ms
64 bytes from 10.0.0.8: icmp_seq=14 ttl=64 time=9.32 ms
64 bytes from 10.0.0.8: icmp_seq=15 ttl=64 time=10.1 ms
64 bytes from 10.0.0.8: icmp_seq=16 ttl=64 time=9.82 ms
64 bytes from 10.0.0.8: icmp_seq=17 ttl=64 time=9.45 ms
```



```
64 bytes from 10.0.0.8: icmp_seq=87 ttl=64 time=9.98 ms
64 bytes from 10.0.0.8: icmp_seq=88 ttl=64 time=9.71 ms
64 bytes from 10.0.0.8: icmp_seq=89 ttl=64 time=9.77 ms
64 bytes from 10.0.0.8: icmp_seq=90 ttl=64 time=11.0 ms
64 bytes from 10.0.0.8: icmp_seq=91 ttl=64 time=9.62 ms
64 bytes from 10.0.0.8: icmp_seq=92 ttl=64 time=10.4 ms
64 bytes from 10.0.0.8: icmp_seq=93 ttl=64 time=9.28 ms
64 bytes from 10.0.0.8: icmp_seq=94 ttl=64 time=9.66 ms
64 bytes from 10.0.0.8: icmp_seq=95 ttl=64 time=8.93 ms
64 bytes from 10.0.0.8: icmp_seq=96 ttl=64 time=8.97 ms
64 bytes from 10.0.0.8: icmp_seq=97 ttl=64 time=9.87 ms
64 bytes from 10.0.0.8: icmp_seq=98 ttl=64 time=10.4 ms
64 bytes from 10.0.0.8: icmp_seq=99 ttl=64 time=10.3 ms
64 bytes from 10.0.0.8: icmp_seq=100 ttl=64 time=9.37 ms

--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99133ms
rtt min/avg/max/mdev = 5.558/9.382/13.332/1.033 ms
mininet> 
```

b. What is the minimum and maximum ping you have observed?

Ans:

For h1 ping -c100 h2: Minimum ping is 1.418 ms and maximum ping is 3.623 ms.

For h1 ping -c100 h8: Minimum ping is 5.558 ms and maximum ping is 13.332 ms.

c. What is the difference, and why?

Ans: Ping for h1 ping h8 (9.382ms avg) is more than the ping for h1 ping h2 (2.551ms avg) because s3 is the only switch between h1 and h2 so the packets travel faster, while packets need to travel from s3 to s2 to s1 to s5 to s7 to h8 for h1 ping h8.

3. Run “iperf h1 h2” and “iperf h1 h8”

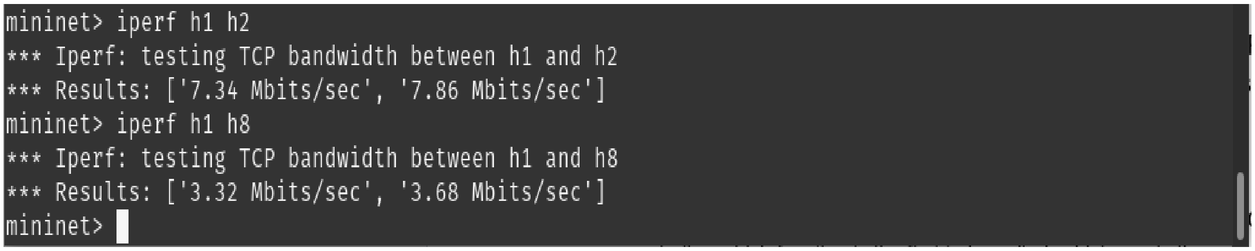
a. What is “iperf” used for?

Ans: iperf is an open source tool command that is used to test the bandwidth of TCP/UDP data transmission.

b. What is the throughput for each case?

Ans: For h1 to h2: [‘7.34Mbits/sec’, ‘7.86Mbits/sec’]

For h1 to h8: [‘3.32Mbits/sec’, ‘3.68Mbits/sec’]



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['7.34 Mbits/sec', '7.86 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.32 Mbits/sec', '3.68 Mbits/sec']
mininet> 
```

c. What is the difference, and explain the reasons for the difference.

Ans: Throughput for h1 to h2 is more than the throughput from h1 to h8 because s3 is the only switch between h1 and h2 so the packets are transferred faster from h1 to s3 to h2, while packets are transferred from s3 to s2 to s1 to s5 to s7 to h8 for h1 ping h8.

4. Which of the switches observe traffic? Please describe your way for observing such traffic on switches (e.g., adding some functions in the “of_tutorial” controller).

Ans: We can observe the traffic on switches by adding the log.info function as it has been added inside the _handle_PacketIn function because it is the function that is called when a packet is received. From this, it is observed that all switches observed traffic.

Task 3

MAC Learning Controller

Questions

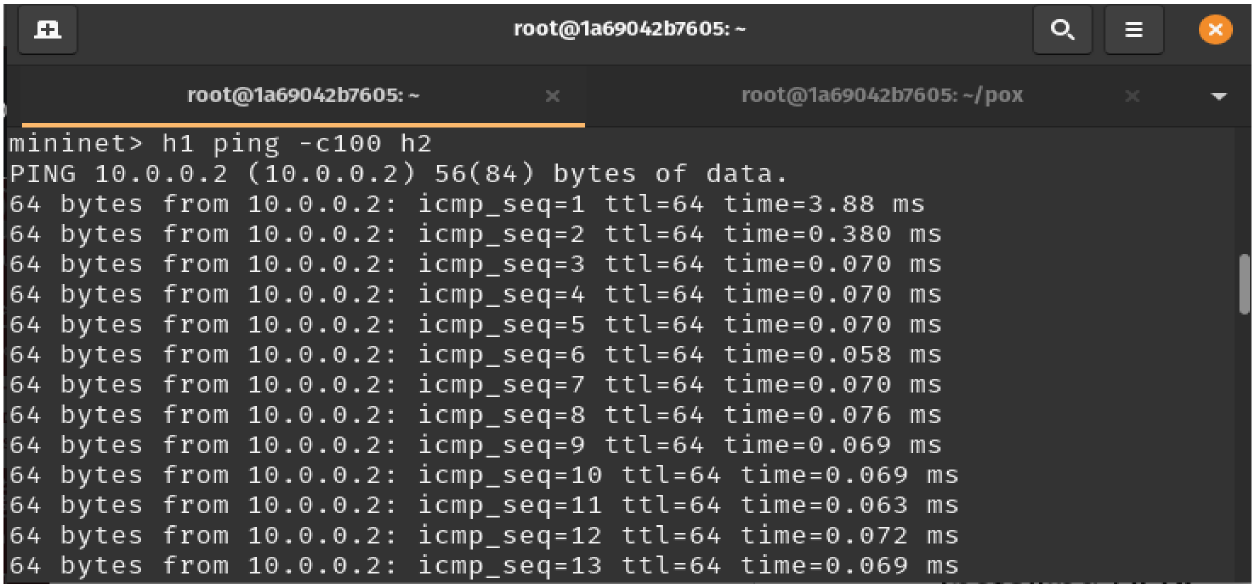
1. Describe how the above code works, such as how the "MAC to Port" map is established. You could use a ‘ping’ example to describe the establishment process (e.g., h1 ping h2).

Ans: The “MAC to port” map is established using act_like_switch function by giving it a switch function. The port for the source MAC is learned, if the port associated with the destination MAC of the packet is known then the packet is sent to the associated port, else the packet is flooded to all destination ports except the input port.

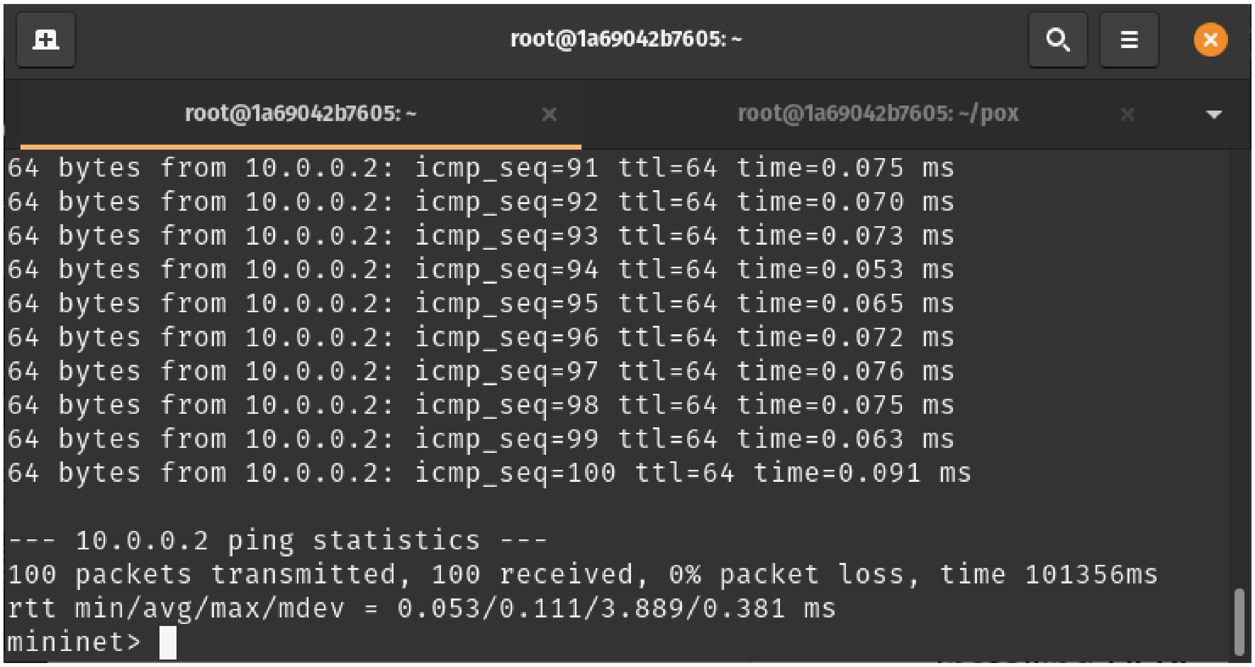
2. (Comment out all prints before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

   a. How long did it take (on average) to ping for each case?

   Ans: For h1 ping -c100 h2: It takes 0.111 ms.



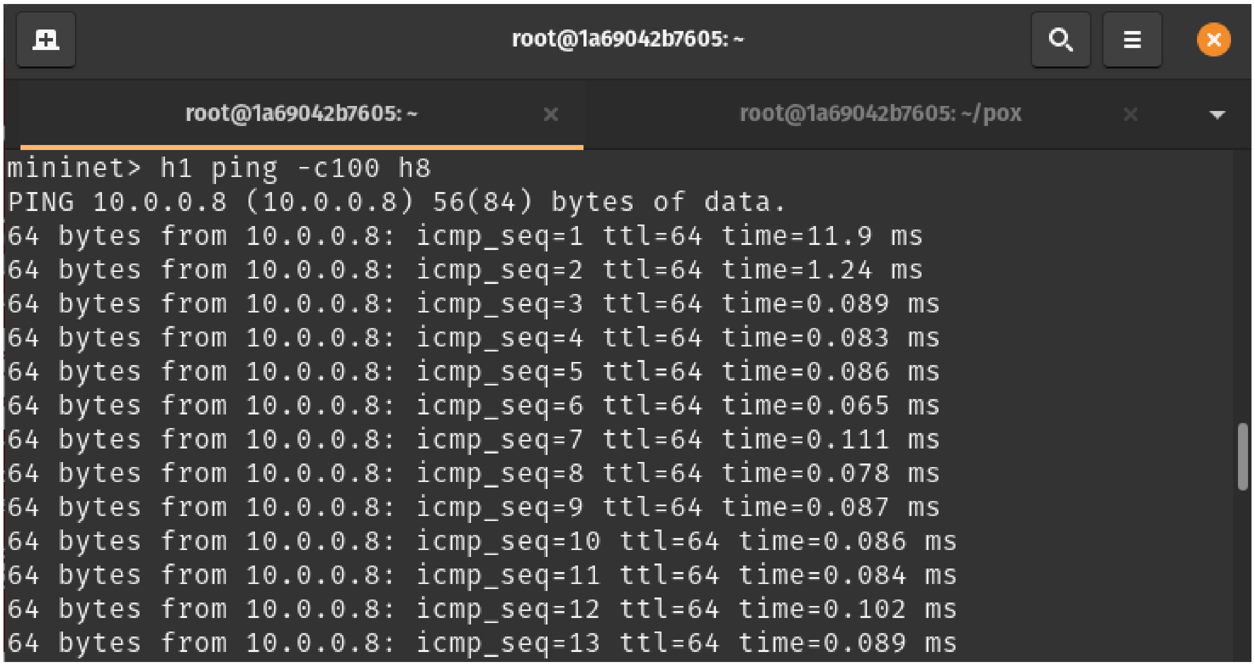
```
mininet> h1 ping -c100 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.88 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.380 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.069 ms
```



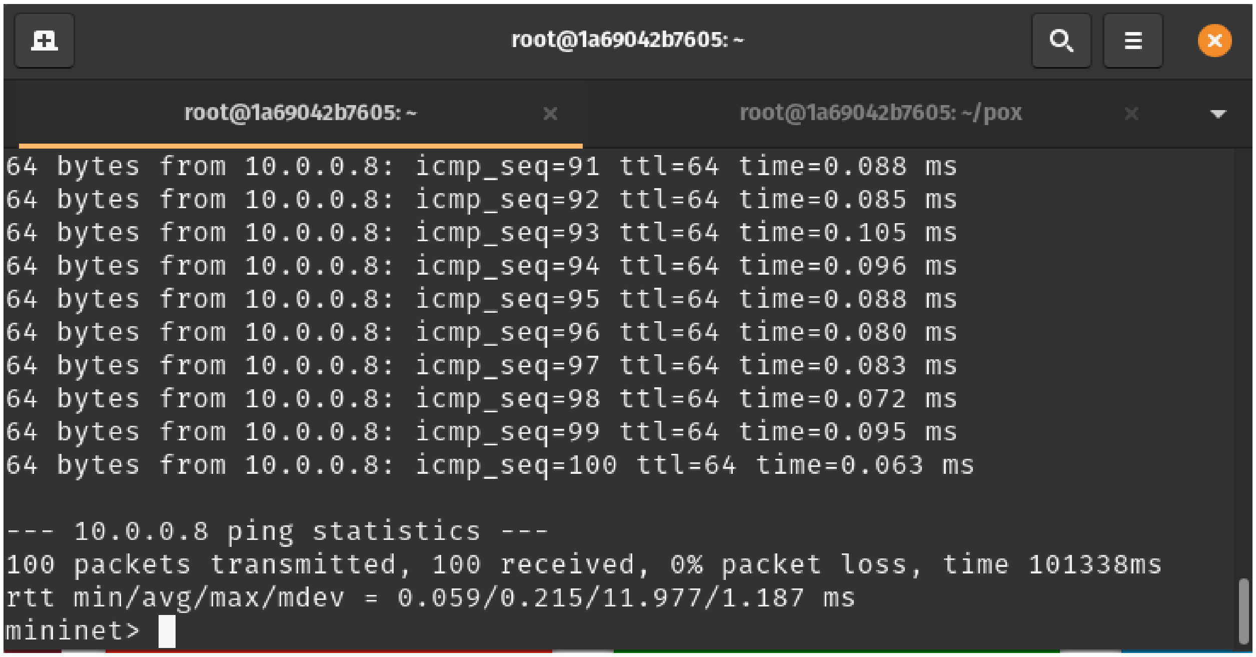
```
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=0.091 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101356ms
rtt min/avg/max/mdev = 0.053/0.111/3.889/0.381 ms
mininet> 
```

For h1 ping -c100 h8: It takes 0.215 ms.



```
mininet> h1 ping -c100 h8
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=11.9 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=1.24 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.086 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.065 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.111 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.078 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.087 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.086 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=0.084 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=0.102 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=0.089 ms
```



```
64 bytes from 10.0.0.8: icmp_seq=91 ttl=64 time=0.088 ms
64 bytes from 10.0.0.8: icmp_seq=92 ttl=64 time=0.085 ms
64 bytes from 10.0.0.8: icmp_seq=93 ttl=64 time=0.105 ms
64 bytes from 10.0.0.8: icmp_seq=94 ttl=64 time=0.096 ms
64 bytes from 10.0.0.8: icmp_seq=95 ttl=64 time=0.088 ms
64 bytes from 10.0.0.8: icmp_seq=96 ttl=64 time=0.080 ms
64 bytes from 10.0.0.8: icmp_seq=97 ttl=64 time=0.083 ms
64 bytes from 10.0.0.8: icmp_seq=98 ttl=64 time=0.072 ms
64 bytes from 10.0.0.8: icmp_seq=99 ttl=64 time=0.095 ms
64 bytes from 10.0.0.8: icmp_seq=100 ttl=64 time=0.063 ms

--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101338ms
rtt min/avg/max/mdev = 0.059/0.215/11.977/1.187 ms
mininet> 
```

b. What is the minimum and maximum ping you have observed?

Ans: For h1 ping -c100 h2: Minimum ping is 0.053 ms and maximum ping is 3.889 ms.

For h1 ping -c100 h8: Minimum ping is 0.059 ms and maximum ping is 11.977 ms.

c. Any difference from Task 2 and why do you think there is a change if there is?

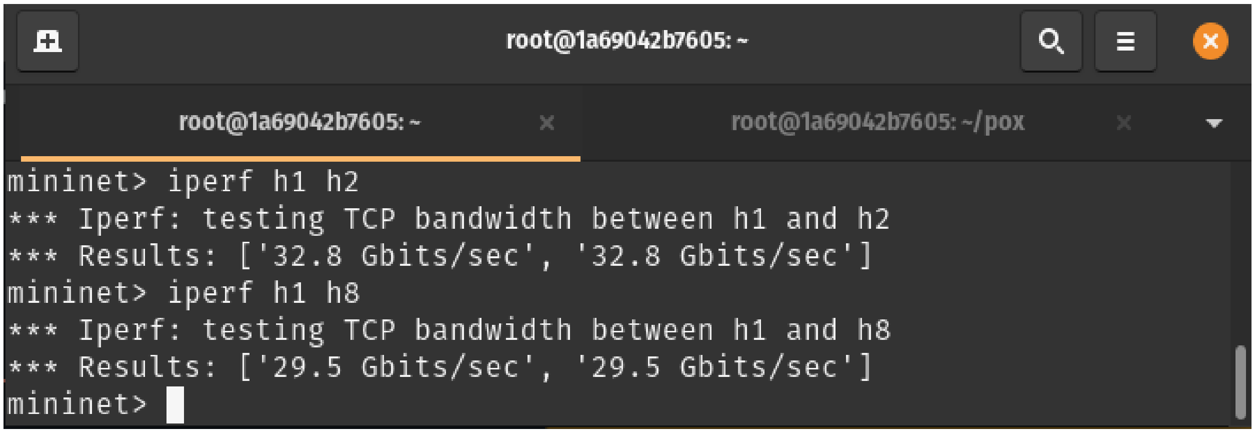
Ans: Task 3 takes way less time than Task 2 for both h1 ping h2 and h1 ping h8, and the time difference is substantial. This is because the packets are flooded at first but after the destination MAC address is mapped to "MAC to port" the packets are just sent to the destination MAC address by the switches. It can be seen that the first ping has the maximum time for both h1 ping h2 and h1 ping h8 because there is no "MAC to port" mapping.

3. Run "iperf h1 h2" and "iperf h1 h8".

a. What is the throughput for each case?

Ans: For h1 to h2: ['32.8Gbits/sec', '32.8Gbits/sec']

For h1 to h8: ['29.5Gbits/sec', '29.5Gbits/sec']



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['32.8 Gbits/sec', '32.8 Gbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['29.5 Gbits/sec', '29.5 Gbits/sec']
mininet>
```

b. What is the difference from Task 2 and why do you think there is a change if there is?

Ans: The throughput for task 3 is significantly higher than the throughput for task2. This is because of the "MAC to port" mapping of the destination

MAC address to the port, which prevents flooding and increases the throughput.