Assignment 3– COEN 241 (Cloud Computing)

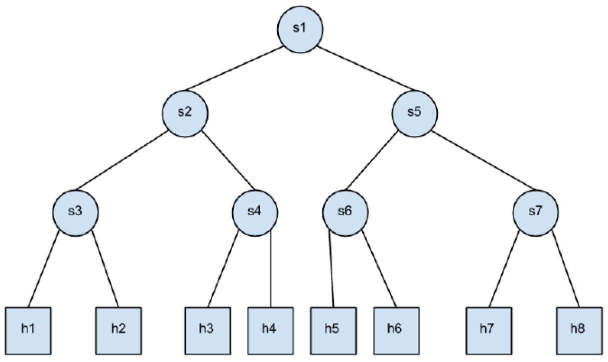
Submitted by: Arpita Verma
SCU ID: W1632653

<u>Mininet & OpenFlow</u>

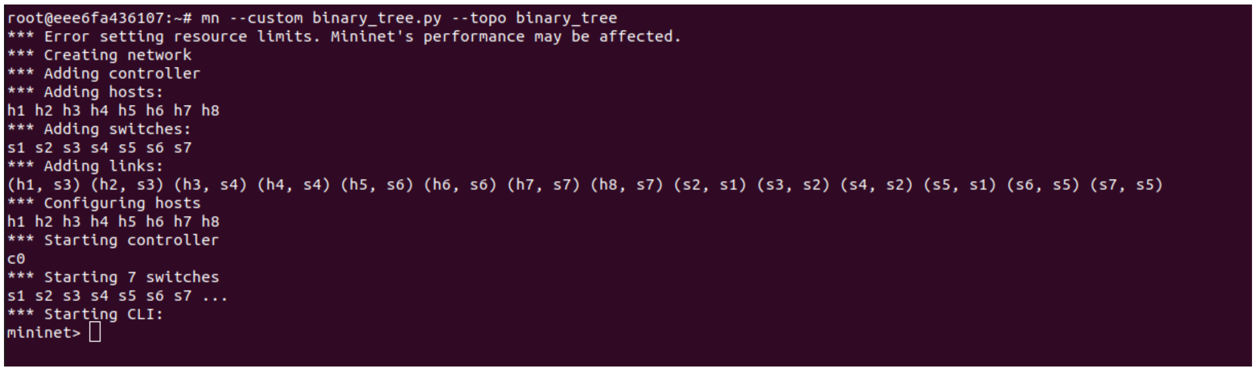
Table of Contents

Task1: Defining custom topologies

We have 7 switches and 8 hosts here n the binary tree as defined in our binary.py code.

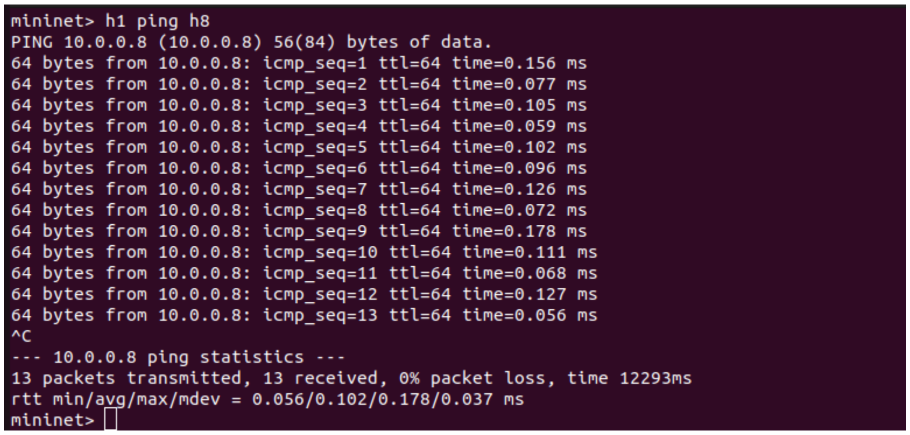


$ sudo mn --custom binary_tree.py --topo binary_tree



```
root@eee6fa436107:~# 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> 
```

mininet> h1 ping h8

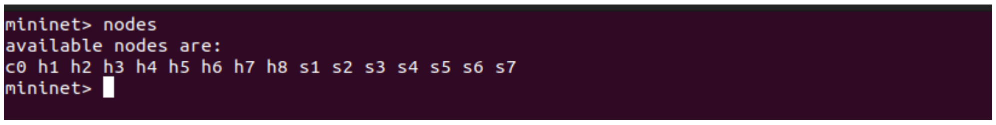


```
mininet> h1 ping h8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=0.156 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.105 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.059 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.102 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.096 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.126 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.072 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.178 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.111 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=0.068 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=0.127 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=0.056 ms
^C
--- 10.0.0.8 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12293ms
rtt min/avg/max/mdev = 0.056/0.102/0.178/0.037 ms
mininet> 
```

Questions – Task1

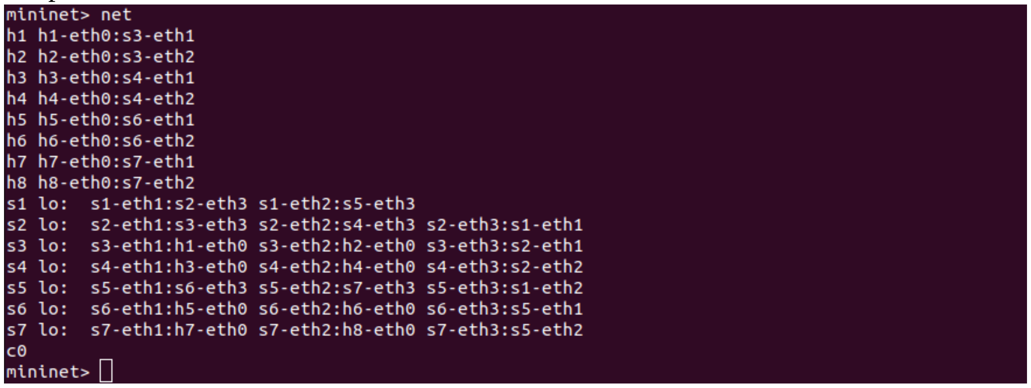
1. What is the output of “nodes” and “net”

Output of nodes:



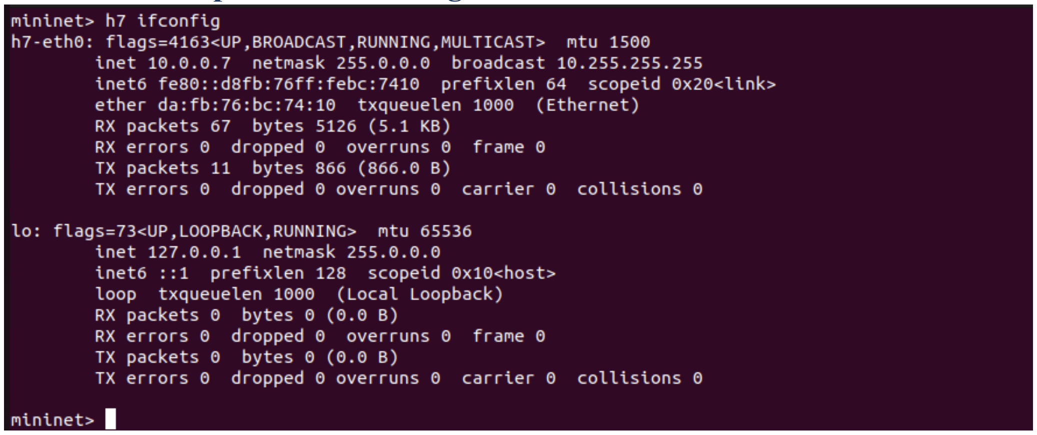
```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7
mininet> 
```

Output of net:



```
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. What is the output of “h7 ifconfig”

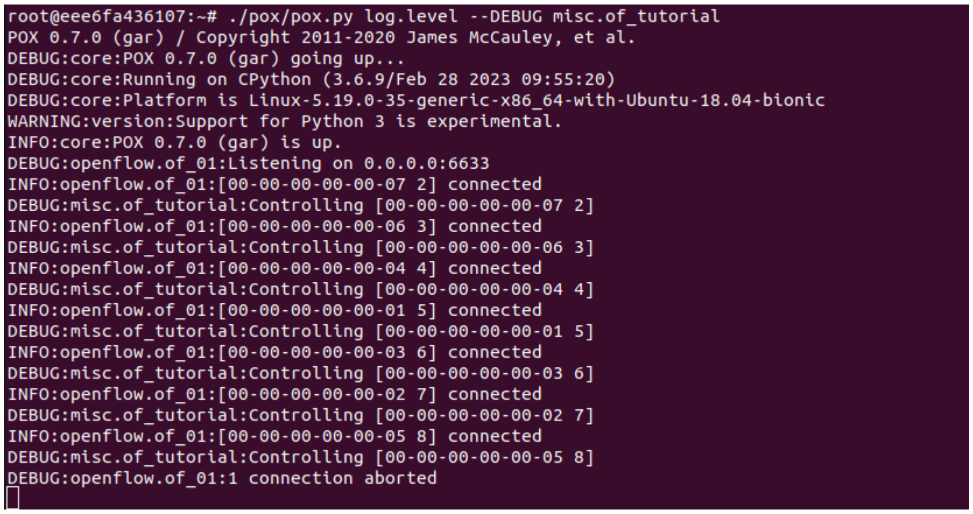


```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::d8fb:76ff:febc:7410  prefixlen 64  scopeid 0x20<link>
        ether da:fb:76:bc:74:10  txqueuelen 1000  (Ethernet)
        RX packets 67  bytes 5126 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        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

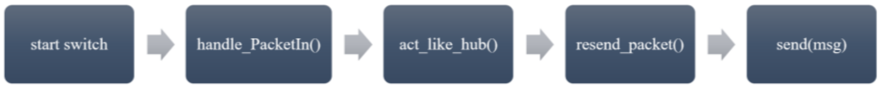


```
root@eee6fa436107:~# ./pox/pox.py log.level --DEBUG misc.of_tutorial
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.6.9/Feb 28 2023 09:55:20)
DEBUG:core:Platform is Linux-5.19.0-35-generic-x86_64-with-Ubuntu-18.04-bionic
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-07 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-07 2]
INFO:openflow.of_01:[00-00-00-00-00-06 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 3]
INFO:openflow.of_01:[00-00-00-00-00-04 4] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-04 4]
INFO:openflow.of_01:[00-00-00-00-00-01 5] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 5]
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-03 6]
INFO:openflow.of_01:[00-00-00-00-00-02 7] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-02 7]
INFO:openflow.of_01:[00-00-00-00-00-05 8] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-05 8]
DEBUG:openflow.of_01:1 connection aborted
```

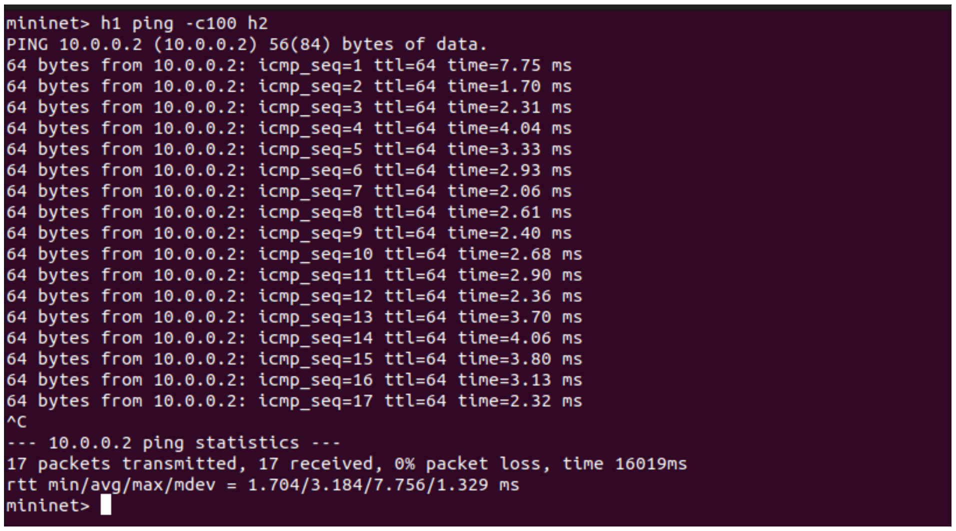
Questions – Task 2

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?
    - Step 1: Start the POX listener using `./pox.py log.level –DEBUG misc.of_tutorial` command, which then turns on the `start switch`.
    - Step 2: Now, The _handle PacketIn() method is called by the `start switch` to handle the packet in messages from the switch.
    - Step 3: After this, the act like hub() function is then called by the _handle PacketIn() function. The act like hub() function generates a behavior that sends packets to all ports except the input port, simulating a hub environment.
    - Step 4: Now, the resend packet() method is then called. The resend packet() function adds a packet to the message data and performs the action on it.
    - Step 5: The switch is then instructed to resend the packet to a specified port by this message.

    The graph is shown as below:

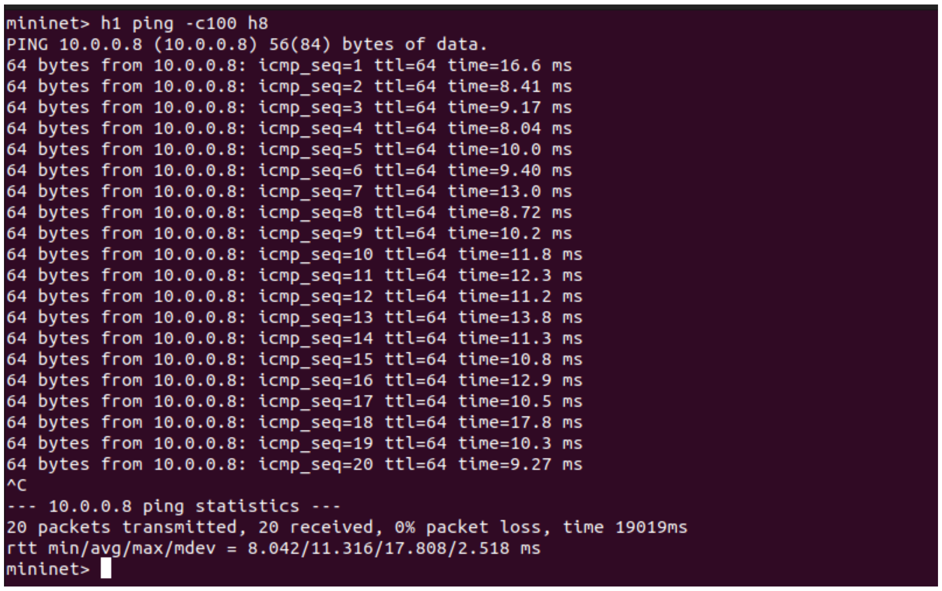


2. **Have h1 ping h2, and h1 ping h8 for 100 times** (e.g., h1 ping -c100 p2).
    a. How long does it take (on average) to ping for each case?
    For h1 ping -c100 h2
    Average time to ping in case of h1 ping h2 is 3.184 ms as shown in below screenshot –



```
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=7.75 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.70 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.31 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.04 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=3.33 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=2.93 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=2.06 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2.61 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=2.40 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=2.68 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=2.90 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=2.36 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=3.70 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=4.06 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=3.80 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=3.13 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=2.32 ms
^C
--- 10.0.0.2 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16019ms
rtt min/avg/max/mdev = 1.704/3.184/7.756/1.329 ms
mininet> 
```

For h1 ping -c100 h8 -
Average time to ping in case of h1 ping h2 is 11.316 ms as shown in below screenshot –



```
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=16.6 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=8.41 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=9.17 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=8.04 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=10.0 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=9.40 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=13.0 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=8.72 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=10.2 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=11.8 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=12.3 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=11.2 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=13.8 ms
64 bytes from 10.0.0.8: icmp_seq=14 ttl=64 time=11.3 ms
64 bytes from 10.0.0.8: icmp_seq=15 ttl=64 time=10.8 ms
64 bytes from 10.0.0.8: icmp_seq=16 ttl=64 time=12.9 ms
64 bytes from 10.0.0.8: icmp_seq=17 ttl=64 time=10.5 ms
64 bytes from 10.0.0.8: icmp_seq=18 ttl=64 time=17.8 ms
64 bytes from 10.0.0.8: icmp_seq=19 ttl=64 time=10.3 ms
64 bytes from 10.0.0.8: icmp_seq=20 ttl=64 time=9.27 ms
^C
--- 10.0.0.8 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19019ms
rtt min/avg/max/mdev = 8.042/11.316/17.808/2.518 ms
mininet> 
```

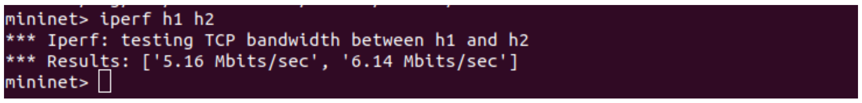
b. What is the minimum and maximum ping you have observed?
Minimum ping -
   For h1 ping -c100 h2 – 1.704 ms
   For h1 ping -c100 h8 – 8.042 ms
Maximum ping -
   For h1 ping -c100 h2 – 7.756 ms
   For h1 ping -c100 h8 – 17.808 ms

c. What is the difference, and why?
Ping time is higher for h1 to h8 (11.316 ms as shown in above screenshot) compared to h1 to h2 (3.184 ms as shown in above screenshot). h1 has to traverse through s3,s2,s1,s5 and s7 to reach h8 while there is only one switch between h1 and h2.
Therefore, the ping time is higher in the second case i.e. for h1 to h8.

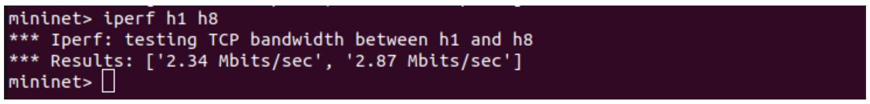
3. Run “iperf h1 h2” and “iperf h1 h8”

a. **What is "iperf" used for?**
With the open source tool Iperf, administrators can estimate bandwidth for optimal network and line performance. The network link is being restricted by two hosts running iperf. The amount of data exchanged between any two nodes on a network line can be calculated using it.

b. What is the throughput for each case?
Throughput for h1 to h2 : Results: ['5.16 Mbits/sec', '6.14 Mbits/sec']
Throughput for h1 to h8 : Results: ['2.34 Mbits/sec', '2.87 Mbits/sec']



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['5.16 Mbits/sec', '6.14 Mbits/sec']
mininet> 
```



```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.34 Mbits/sec', '2.87 Mbits/sec']
mininet> 
```

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

Throughput is higher between h1 and h2 than it is between h1 and h8 due to less network congestion and delay (same as ping time being slower). More data can be transmitted in less time since there are fewer hops between h1 and h2. Less data throughput may be delivered in a given period of time since there are more hops between h1 and 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).

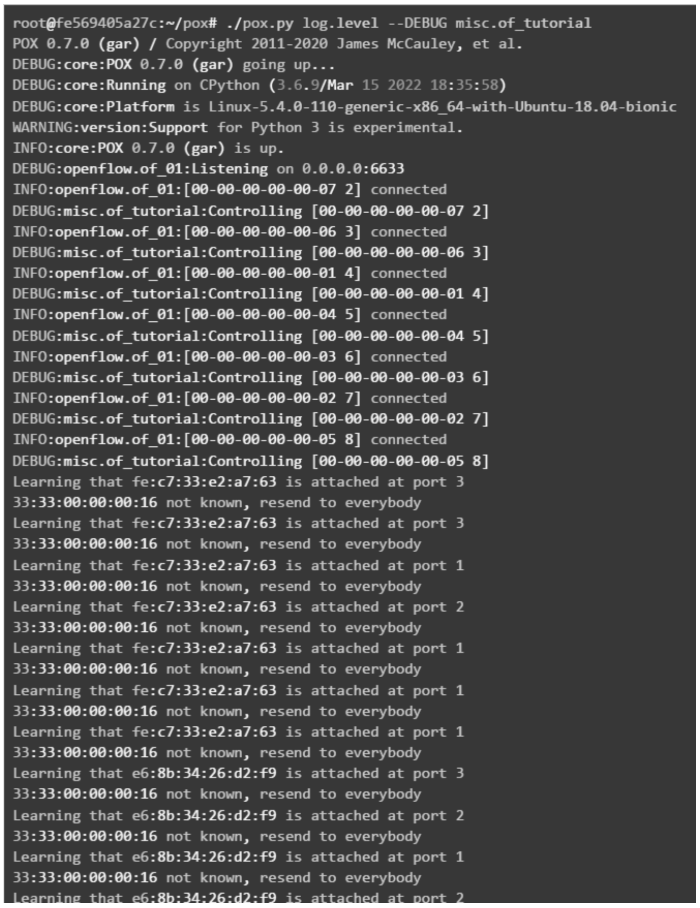
By adding log.info("Switch observing traffic: % s" percent (self.connection) to line 106 of the tutorial controller, we can examine the data that enables us to view the traffic. We can infer from this that all switches keep an eye on traffic, especially when they are slammed with packets. Every time a packet is received, the event listener function _handle PacketIn is called.

Task 3: MAC Learning Controller

Questions – Task 3

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).

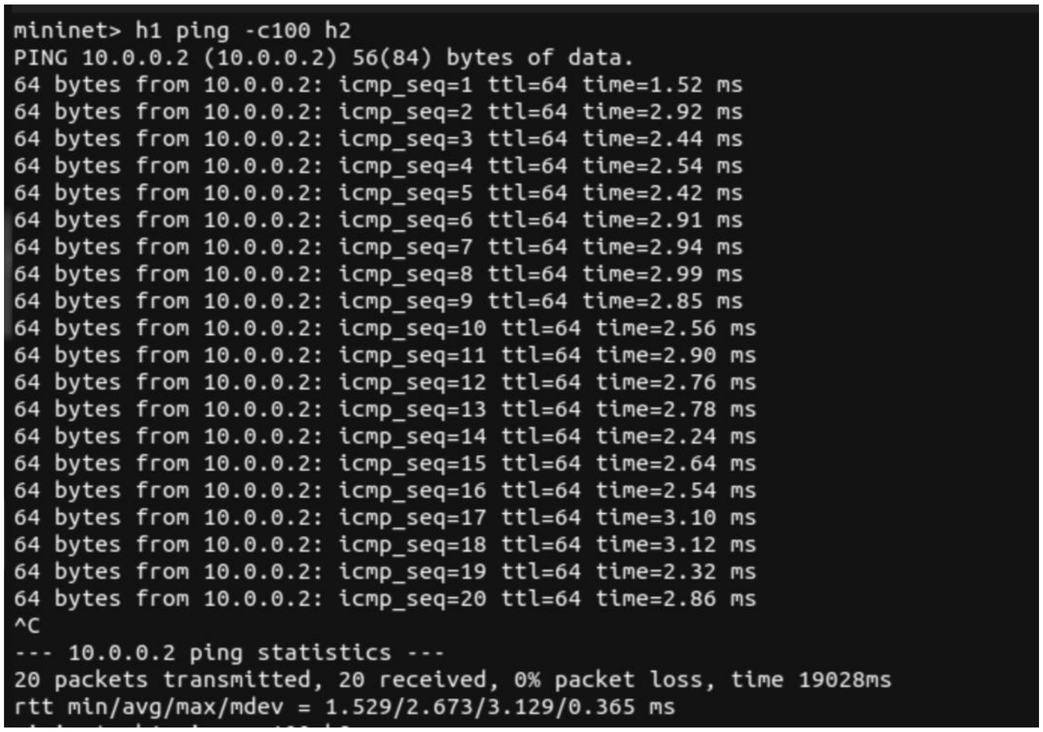
We can find out where MAC addresses are situated according to the act like switch function in our code. As a result, the controller may conveniently map a MAC address to a port when it is determined that it is the address to which a sender intends to send a message. As the packet is simply delivered to that known port, this also helps the controller's speed when delivering packets to addresses that are already known. If the destination is unknown, the function just floods the packet to all destinations. The MAC Learning Controller assists in enhancing ping times and throughput by reducing flooding.



```
root@fe569405a27c:~/pox# ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.6.9/Mar 15 2022 18:35:58)
DEBUG:core:Platform is Linux-5.4.0-110-generic-x86_64-with-Ubuntu-18.04-bionic
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-07 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-07 2]
INFO:openflow.of_01:[00-00-00-00-00-06 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 3]
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 4]
INFO:openflow.of_01:[00-00-00-00-00-04 5] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-04 5]
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-03 6]
INFO:openflow.of_01:[00-00-00-00-00-02 7] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-02 7]
INFO:openflow.of_01:[00-00-00-00-00-05 8] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-05 8]
Learning that fe:c7:33:e2:a7:63 is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that fe:c7:33:e2:a7:63 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that e6:8b:34:26:d2:f9 is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
Learning that e6:8b:34:26:d2:f9 is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that e6:8b:34:26:d2:f9 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that e6:8b:34:26:d2:f9 is attached at port 2
```

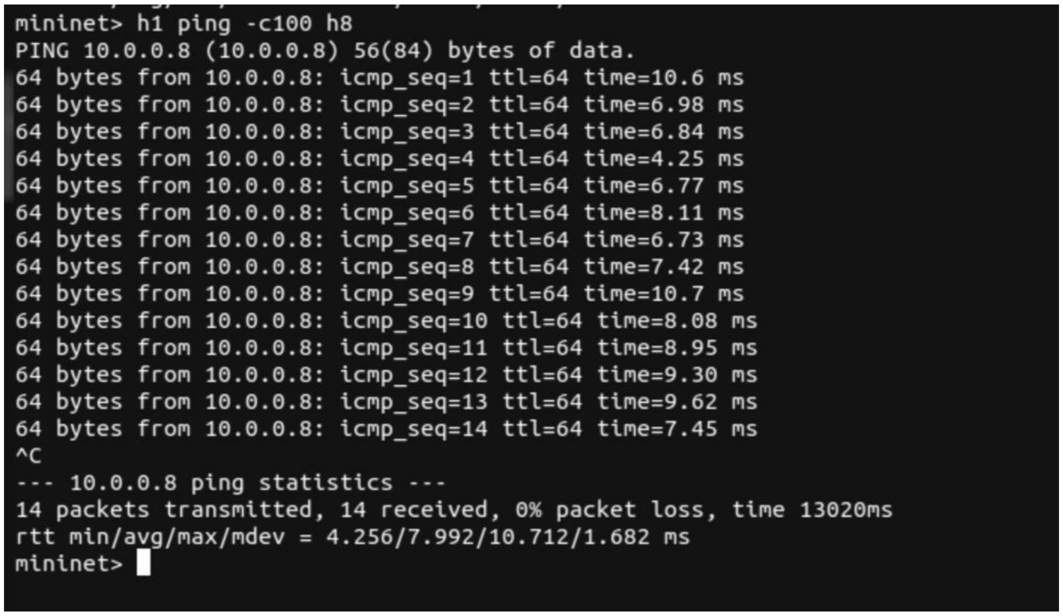
2. **Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).**
    a. How long did it take (on average) to ping for each case?

        For h1 ping -c100 h2
        Average time to ping in case of h1 ping h2 is ms as shown in below screenshot –



```
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=1.52 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.92 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.44 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2.54 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2.42 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=2.91 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=2.94 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2.99 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=2.85 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=2.56 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=2.90 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=2.76 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=2.78 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=2.24 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=2.64 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=2.54 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=3.10 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=3.12 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=2.32 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=2.86 ms
^C
--- 10.0.0.2 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19028ms
rtt min/avg/max/mdev = 1.529/2.673/3.129/0.365 ms
```

For h1 ping -c100 h8 -
Average time to ping in case of h1 ping h2 is ms as shown in below screenshot –



```
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=10.6 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=6.98 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=6.84 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=4.25 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=6.77 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=8.11 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=6.73 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=7.42 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=10.7 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=8.08 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=8.95 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=9.30 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=9.62 ms
64 bytes from 10.0.0.8: icmp_seq=14 ttl=64 time=7.45 ms
^C
--- 10.0.0.8 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13020ms
rtt min/avg/max/mdev = 4.256/7.992/10.712/1.682 ms
mininet> █
```

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

Minimum ping -
For h1 ping -c100 h2 - 1.529 ms
For h1 ping -c100 h8 - 3.129 ms

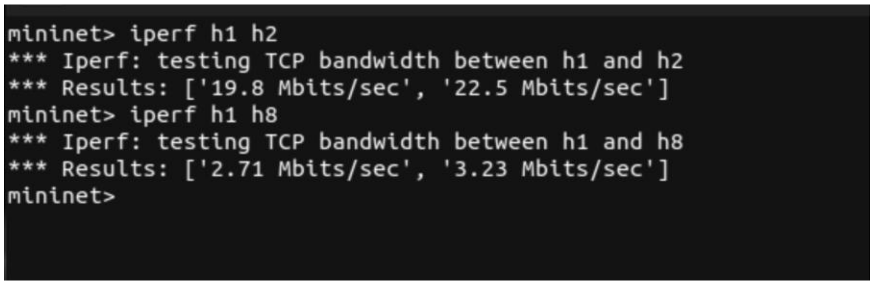
Maximum ping -
For h1 ping -c100 h2 - 4.256 ms
For h1 ping -c100 h8 - 10.712 ms

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

The time difference between tasks 3 and 2 for calculating the value of h1 ping h2 is marginal. The difference in ping time between h1 and h8 is significant because there are many more changes that must be made. It is obvious that task 3 is significantly faster or has less ping time because just the first few packets in job 3 are flooded. Once the destination MAC address has been located in the map, the switches will only resend the packet to the relevant port that is mapped to in the "mac to port" mapping. As a result of the reduced network congestion, subsequent pings are much faster.

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

a. What is the throughput for each case?



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['19.8 Mbits/sec', '22.5 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.71 Mbits/sec', '3.23 Mbits/sec']
mininet>
```

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

In both situations, task 3's throughput is greater than task 2's. This is due to the fact that task 3 will see reduced network congestion due to the porting of Macs. Once the map has learned every port, there won't be any packet flooding, which will relieve the switches of some of their load. We can observe in h1 and h2 that task 2 and 3 had a noticeable improvement in throughputs because the routes are more pre-computed and trained with changes in the controller. Due to the quantity of hops and packet losing in the case of h1 and h8, there is no appreciable improvement.

References

1. http://mininet.org/
2. http://mininet.org/walkthrough/#custom-topologies