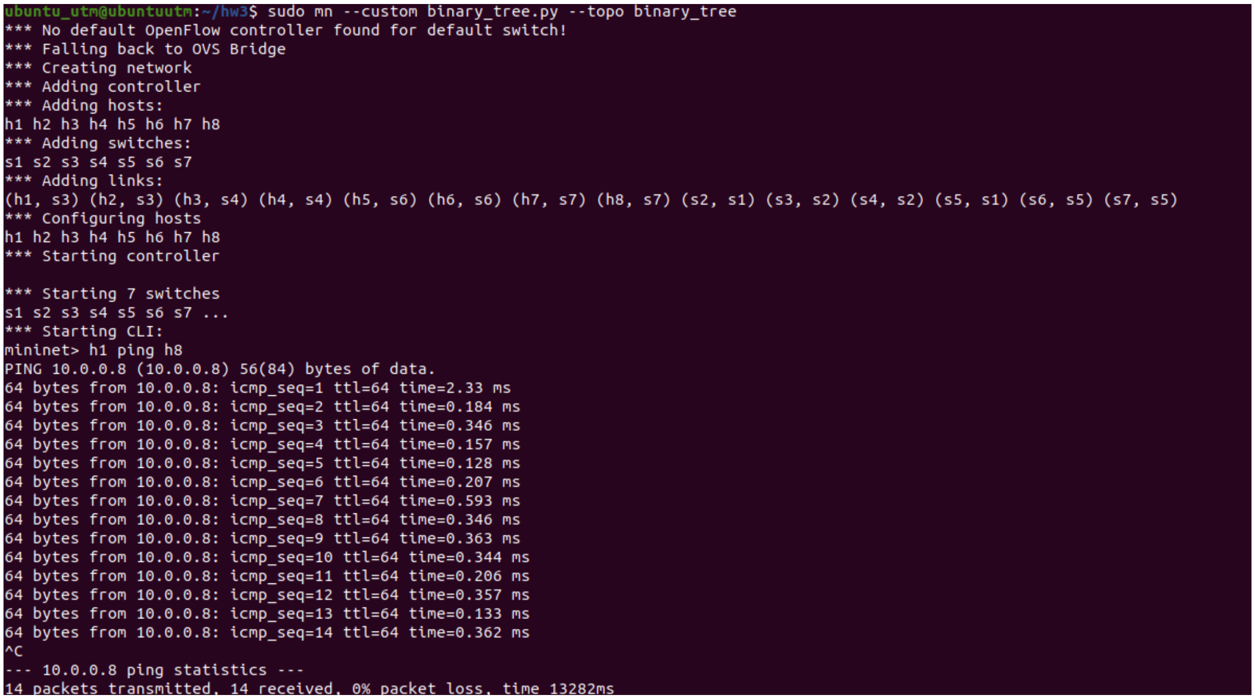
CSEN 241, Winter 2024
HW 3

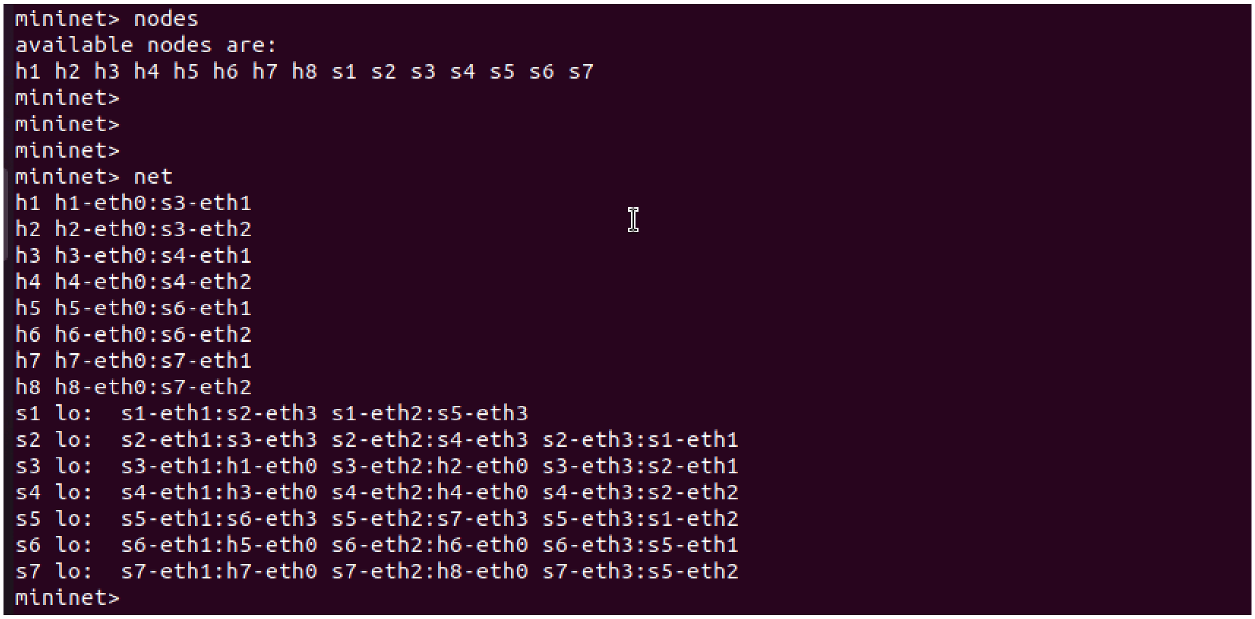
Submitted by: Eshaan Rathi
Link to Code: https://github.com/eshaanrathi2/csen241/tree/main/hw3

Task 1
Run Mininet



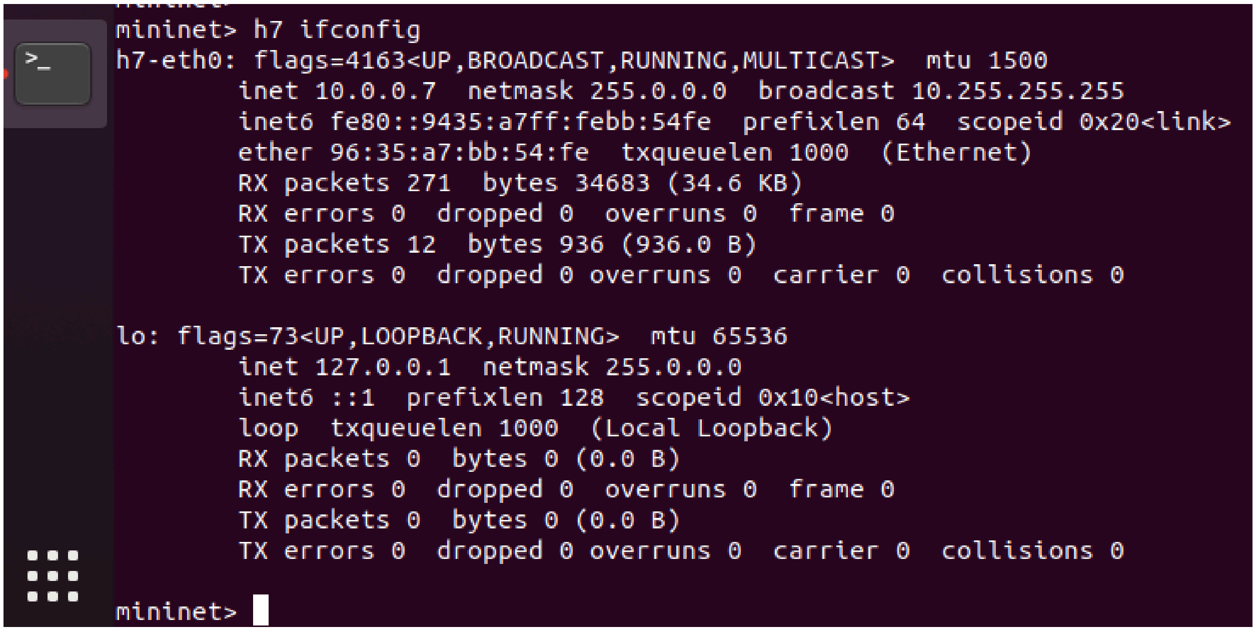
```
ubuntu_utm@ubuntuutm:~/hw3$ sudo mn --custom binary_tree.py --topo binary_tree
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(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

*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
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=2.33 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.184 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.346 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.157 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.128 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.207 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.593 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.346 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.363 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.344 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=0.206 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=0.357 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=0.133 ms
64 bytes from 10.0.0.8: icmp_seq=14 ttl=64 time=0.362 ms
^C
--- 10.0.0.8 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13282ms
```

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



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

Q 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::9435:a7ff:febb:54fe  prefixlen 64  scopeid 0x20<link>
        ether 96:35:a7:bb:54:fe  txqueuelen 1000  (Ethernet)
        RX packets 271  bytes 34683 (34.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 936 (936.0 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

Q1.
Switch is started by the start_switch() function. Then the following functions should be called in this order:

_handle_PacketIn (): Handles packet in messages from the switch.

|
V

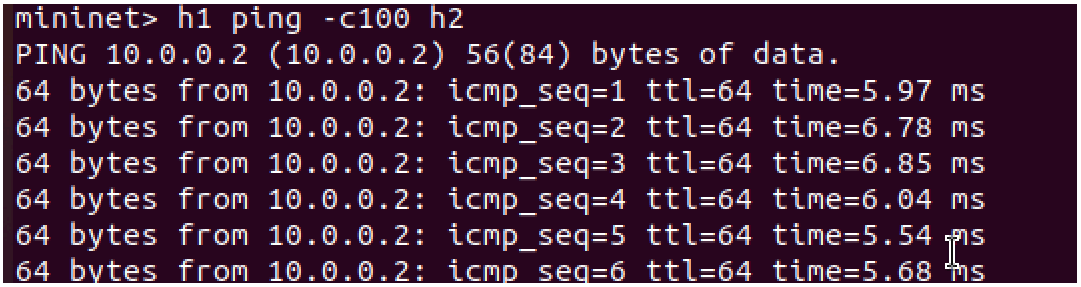
act_like_switch (): Implement switch-like behavior.

|
V

resend_packet (): Instructs the switch to resend a packet that it had sent.

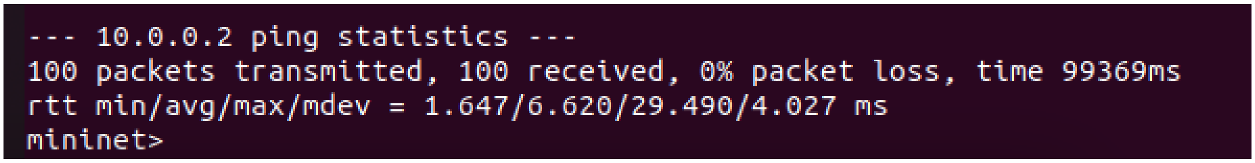
Alternatively, it can call act_like_switch() instead of act_like_switch() in between.

Q2.
h1 to h2:



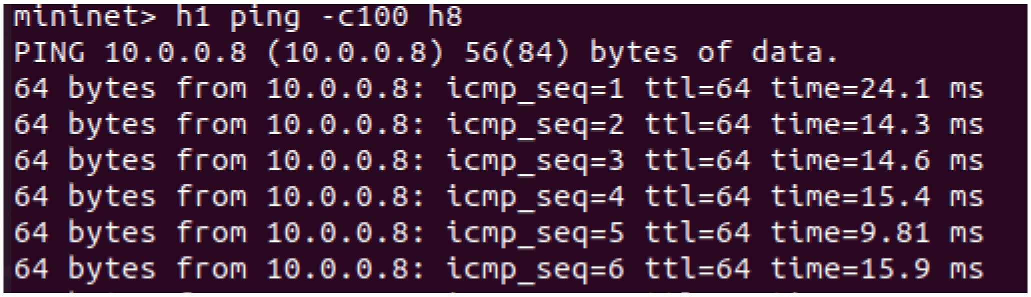
```
mininet> h1 ping -c100 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.97 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=6.78 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.85 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=6.04 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=5.54 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=5.68 ms
```

.... x100



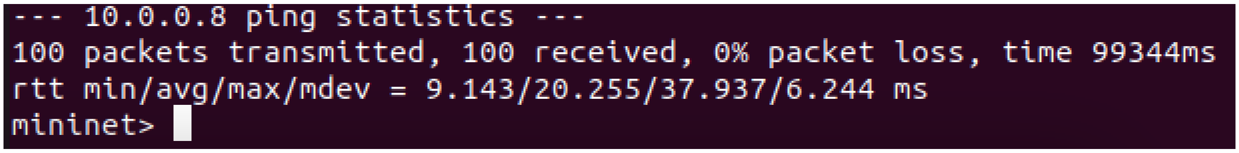
```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99369ms
rtt min/avg/max/mdev = 1.647/6.620/29.490/4.027 ms
mininet>
```

h1 to h8:



```
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=24.1 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=14.3 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=14.6 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=15.4 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=9.81 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=15.9 ms
```

...... x 100



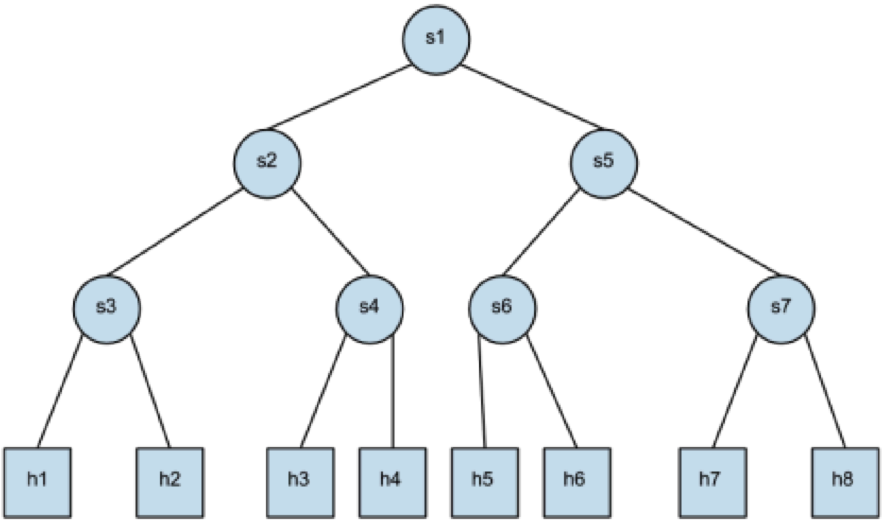
```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99344ms
rtt min/avg/max/mdev = 9.143/20.255/37.937/6.244 ms
mininet>
```

a.
Average Round trip time h1 to h2: 6.620 ms
Average Round trip time h1 to h8: 20.255 ms

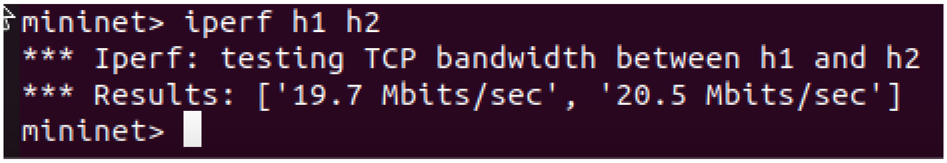
b.
Min Round trip time h1 to h2: 1.647 ms
Min Round trip time h1 to h8: 9.143 ms

Max Round trip time h1 to h2: 29.490 ms
Max Round trip time h1 to h8: 37.937ms

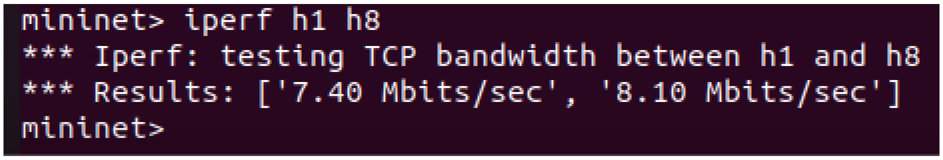
c.
h1 and h2 are 2 links away with 1 switch in between.
But h1 and h8 are 6 links away with 5 switches in between.
Higher the separation, higher the ping time.



Q3



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['19.7 Mbits/sec', '20.5 Mbits/sec']
mininet> 
```



```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['7.40 Mbits/sec', '8.10 Mbits/sec']
mininet>
```

a.
Iperf measures throughput of a network.

b.
Already In the above figure but pasting it again.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['19.7 Mbits/sec', '20.5 Mbits/sec']

mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['7.40 Mbits/sec', '8.10 Mbits/sec']
```

c.
Similar reasons as Q 2. More links brings more congestion and higher latency. Hence reduced throughput.

Q 4.
Since the network was already flooded with packets with the command:

./pox.py log.level --DEBUG misc.of_tutorial

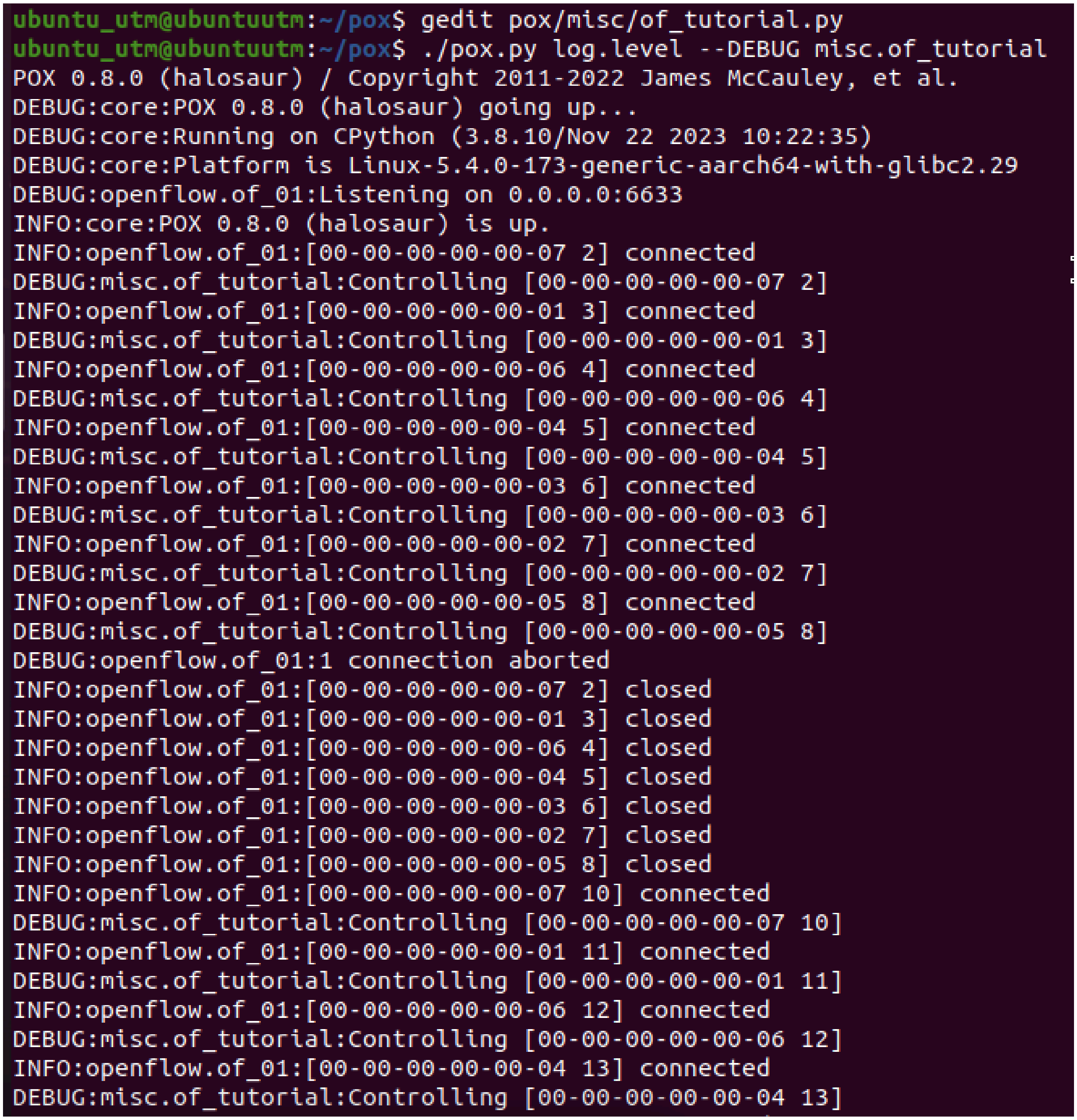
all of the switches will have traffic. We can measure it by adding log or print statements in of_totorial.py controller file.

Task 3

Q1.
The refactored code implements the basic functionality of a learning switch, which dynamically learns the mapping between MAC addresses and switch ports and makes forwarding decisions based on that information.

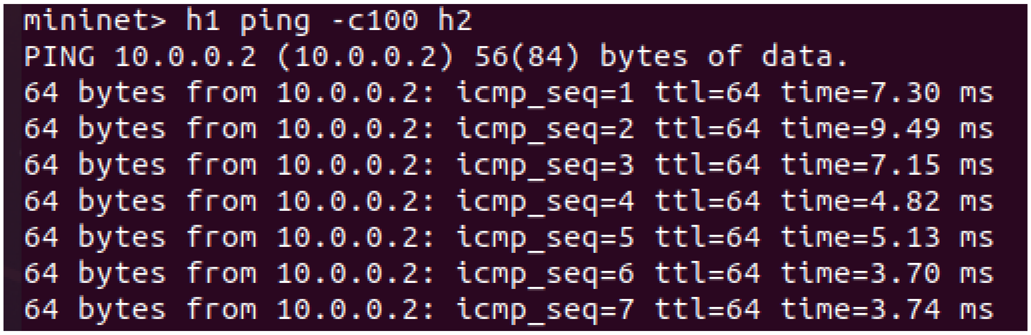
```
  # Learn the port for the source MAC
  # print("Src: ",str(packet.src),":", packet_in.in_port,"Dst:", str(packet.dst))
```

```
    if packet.src not in self.mac_to_port:
        print("Learning that " + str(packet.src) + " is attached at port " +
str(packet_in.in_port))
        self.mac_to_port[packet.src] = packet_in.in_port
    # if the port associated with the destination MAC of the packet is known:
    if packet.dst in self.mac_to_port:
        # Send packet out the associated port
        print(str(packet.dst) + " destination known. only send message to it")
        self.resend_packet(packet_in, self.mac_to_port[packet.dst])
    else:
        # Flood the packet out everything but the input port
        # This part looks familiar, right?
        print(str(packet.dst) + " not known, resend to everybody")
        self.resend_packet(packet_in, of.OFPP_ALL)
```



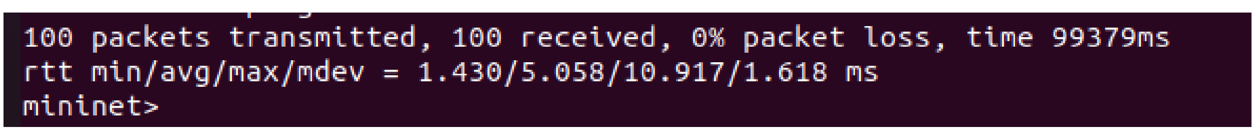
```
ubuntu_utm@ubuntuutm:~/pox$ gedit pox/misc/of_tutorial.py
ubuntu_utm@ubuntuutm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
DEBUG:core:POX 0.8.0 (halosaur) going up...
DEBUG:core:Running on CPython (3.8.10/Nov 22 2023 10:22:35)
DEBUG:core:Platform is Linux-5.4.0-173-generic-aarch64-with-glibc2.29
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:core:POX 0.8.0 (halosaur) is up.
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:[00-00-00-00-00-01 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 3]
INFO:openflow.of_01:[00-00-00-00-00-06 4] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 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]
DEBUG:openflow.of_01:1 connection aborted
INFO:openflow.of_01:[00-00-00-00-00-07 2] closed
INFO:openflow.of_01:[00-00-00-00-00-01 3] closed
INFO:openflow.of_01:[00-00-00-00-00-06 4] closed
INFO:openflow.of_01:[00-00-00-00-00-04 5] closed
INFO:openflow.of_01:[00-00-00-00-00-03 6] closed
INFO:openflow.of_01:[00-00-00-00-00-02 7] closed
INFO:openflow.of_01:[00-00-00-00-00-05 8] closed
INFO:openflow.of_01:[00-00-00-00-00-07 10] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-07 10]
INFO:openflow.of_01:[00-00-00-00-00-01 11] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 11]
INFO:openflow.of_01:[00-00-00-00-00-06 12] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 12]
INFO:openflow.of_01:[00-00-00-00-00-04 13] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-04 13]
```

Q2.

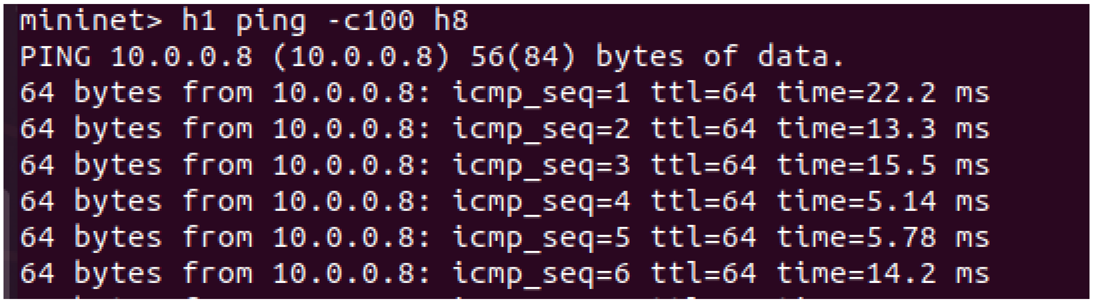


```
mininet> h1 ping -c100 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=7.30 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=9.49 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=7.15 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.82 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=5.13 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=3.70 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=3.74 ms
```

X 100 ……

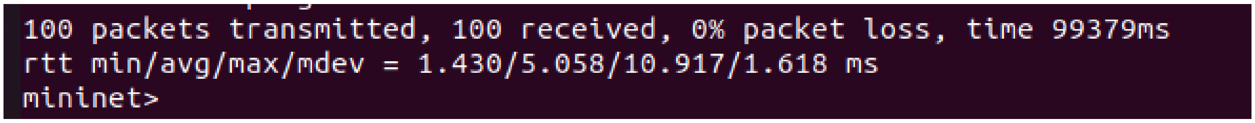


```
100 packets transmitted, 100 received, 0% packet loss, time 99379ms
rtt min/avg/max/mdev = 1.430/5.058/10.917/1.618 ms
mininet>
```



```
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=22.2 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=13.3 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=15.5 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=5.14 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=5.78 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=14.2 ms
```

X 100…..



```
100 packets transmitted, 100 received, 0% packet loss, time 99379ms
rtt min/avg/max/mdev = 1.430/5.058/10.917/1.618 ms
mininet>
```

a.

Average Round trip time h1 to h2: 5.058 ms
Average Round trip time h1 to h8: 17.257 ms

b.

Min Round trip time h1 to h2: 1.430 ms
Min Round trip time h1 to h8: 5.079 ms

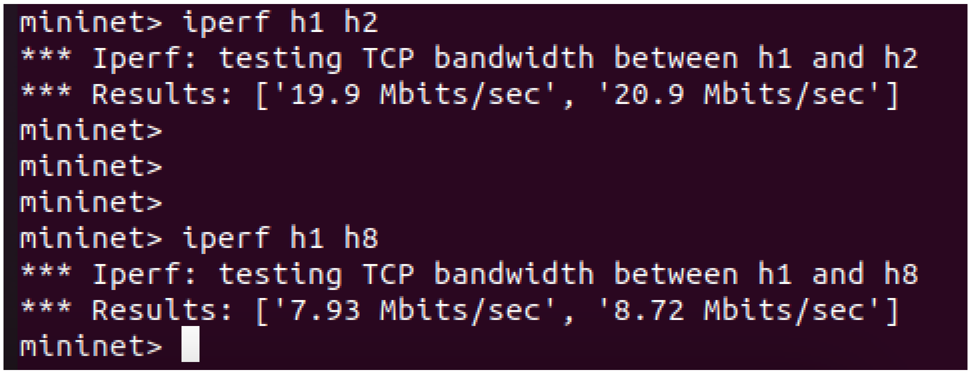
Max Round trip time h1 to h2: 10.917 ms
Max Round trip time h1 to h8: 59.692ms

c.

Yes, Round trip time has reduced by 15 - 20% (average RTT) for both cases h1->h2 and h1->h8.

Previously the switches couldn’t really do much but flood the packets they receive. But now, switches will learn the ports packets arrive from, and upon receiving a packet, if they have already seen its destination address, they will know the exact port to forward it on and avoid flooding the network (i.e., MAC learning).

Q 3.



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['19.9 Mbits/sec', '20.9 Mbits/sec']
mininet>
mininet>
mininet>
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['7.93 Mbits/sec', '8.72 Mbits/sec']
mininet>
```

a.
Already shown in above figure.

b.
Throughput has increased by ~ 0.3 Mbits/sec for h1->h2 and increased by ~ 0.5 Mbits/sec for h1->h8.
The new switching logic has improved the throughput significantly. This is due to maintaining the mac_to_port map, which reduced packet flooding and reduced latency / network congestion. Hence increasing overall throughput.