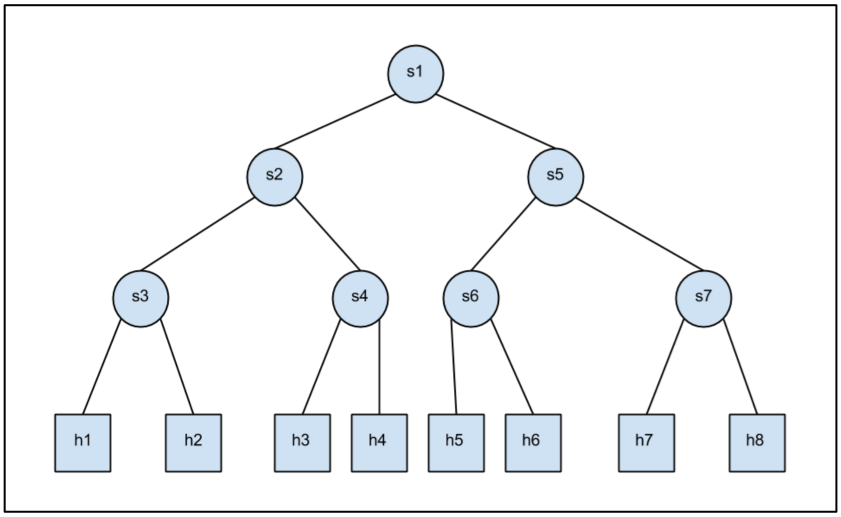
Santa Clara University

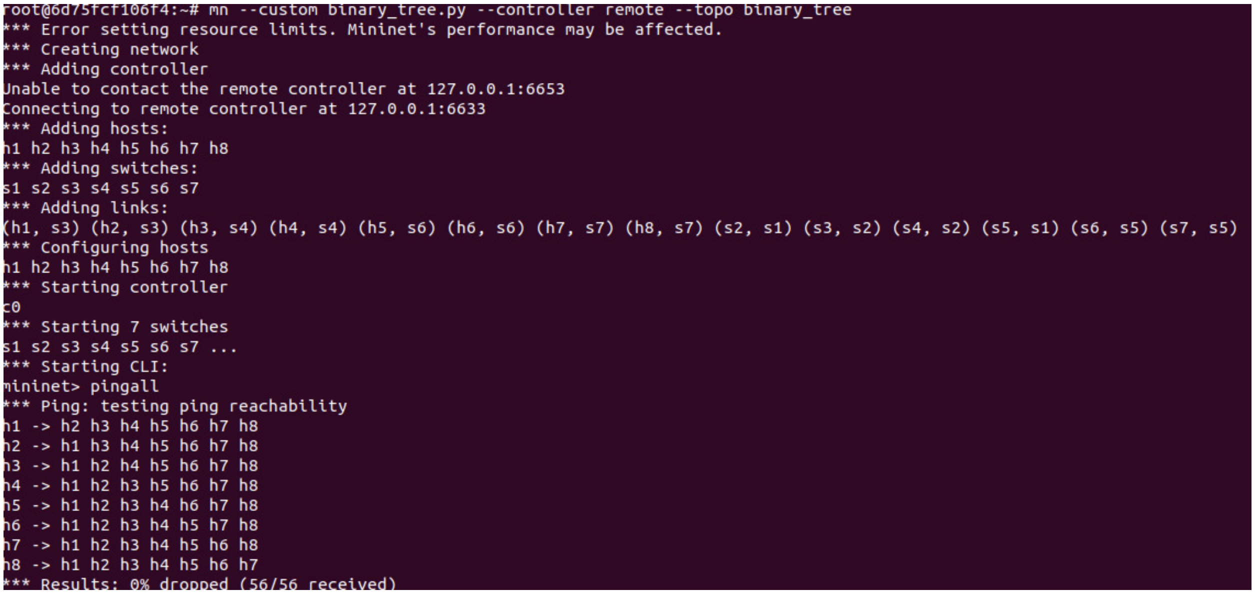
COEN 241: Cloud Computing

HW3: Mininet

Ankita Kumar (W1620637)

The binary tree we are creating has **7 switches and 8 hosts**:

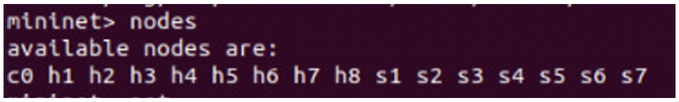




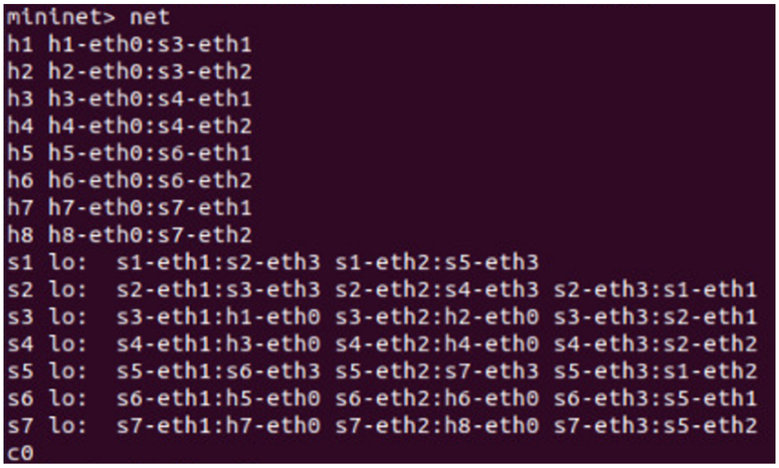
```
root@6d75fcf106f4:~# mn --custom binary_tree.py --controller remote --topo binary_tree
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 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> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

Task 1: Defining custom topologies

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

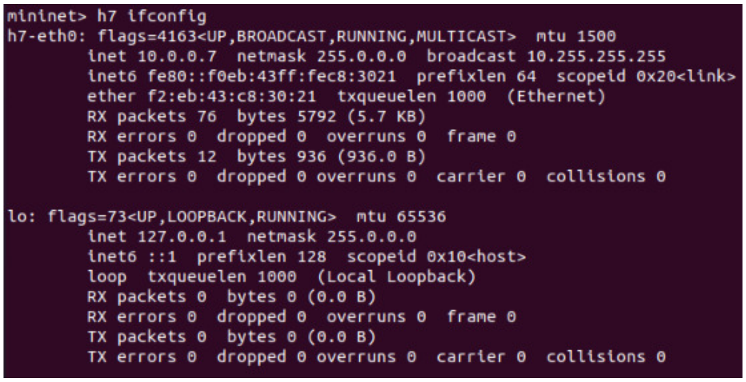


```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7
```



```
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
```

2. What is the output of “h7 ifconfig”?



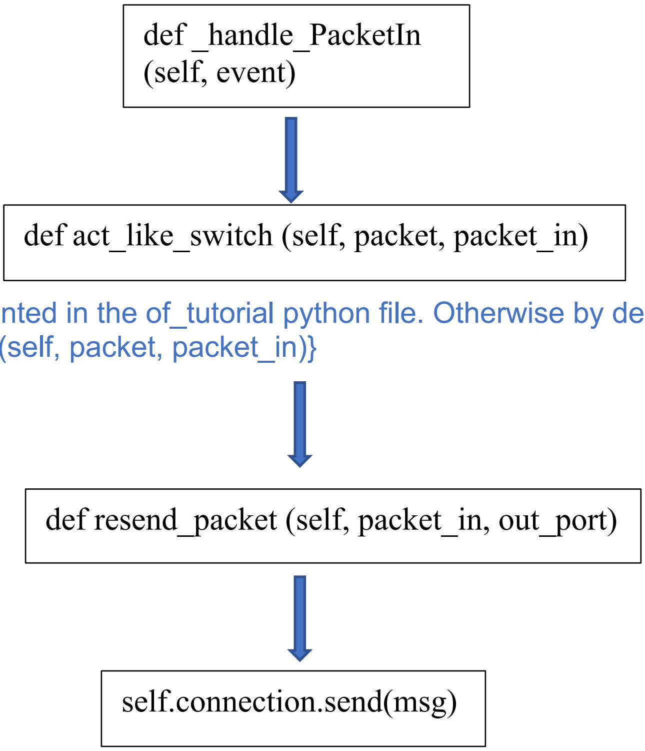
```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f0eb:43ff:fec8:3021  prefixlen 64  scopeid 0x20<link>
        ether f2:eb:43:c8:30:21  txqueuelen 1000  (Ethernet)
        RX packets 76  bytes 5792 (5.7 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
```

Task 2: Analyze the “of_tutorial’ controller

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?

When packet arrives to the controller:



def _handle_PacketIn (self, event)

def act_like_switch (self, packet, packet_in)

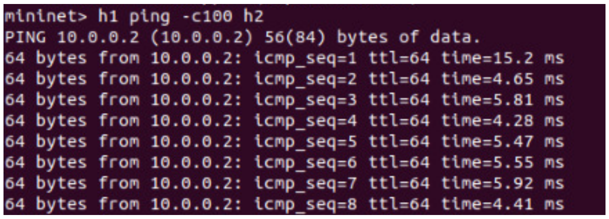
# if uncommented in the of_tutorial python file. Otherwise by default it will go to {def act_like_hub (self, packet, packet_in)}

def resend_packet (self, packet_in, out_port)

self.connection.send(msg)

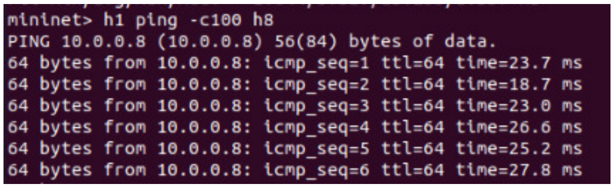
2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

H1 ping h2 100 times.



```
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=15.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=4.65 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=5.81 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.28 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=5.47 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=5.55 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=5.92 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=4.41 ms
```

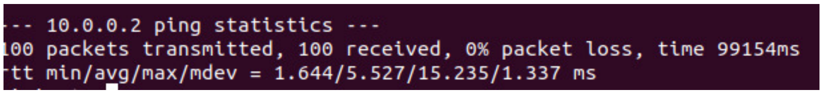
H1 ping h8 100 times.



```
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=23.7 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=18.7 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=23.0 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=26.6 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=25.2 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=27.8 ms
```

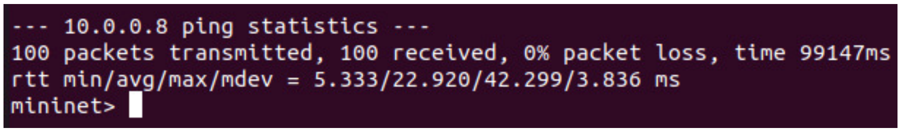
a. How long does it take (on average) to ping for each case?

- H1 ping h2 100 times:
  Average time to ping: 5.527ms



```
-- 10.0.0.2 ping statistics ---
00 packets transmitted, 100 received, 0% packet loss, time 99154ms
tt min/avg/max/mdev = 1.644/5.527/15.235/1.337 ms
```

- H1 ping h8 100 times.
  Average time to ping: 22.920 ms



```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99147ms
rtt min/avg/max/mdev = 5.333/22.920/42.299/3.836 ms
mininet> 
```

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

H1 ping h2 100 times:
minimum and maximum ping observed: 1.644 ms & 15.235 ms
H1 ping h8 100 times.
minimum and maximum ping observed: 5.333 ms & 42.299 ms

c. What is the difference, and why?

H1 ping h8 took longer compared to h1 ping h2. This is due to greater number of switches between h1& h8.

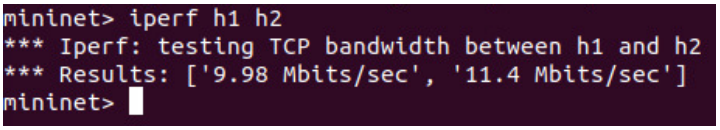
3. Run “iperf h1 h2” and “iperf h1 h8”

a. What is “iperf” used for?

It is used to test TCP bandwidth to evaluate the network performance and quality of a network line.

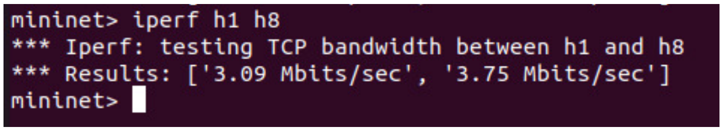
b. What is the throughput for each case?

“iperf h1 h2”



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.98 Mbits/sec', '11.4 Mbits/sec']
mininet> 
```

“iperf h1 h8”



```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.09 Mbits/sec', '3.75 Mbits/sec']
mininet> 
```

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

The throughput for h1 -> h8 is low compared to h1->h2. This is because there are multiple switches between h1 & h8, as compared to h1& h2.
Hence there is more packet drop in case of h1 -> h8 as the packet flows through a greater number of switches.

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).

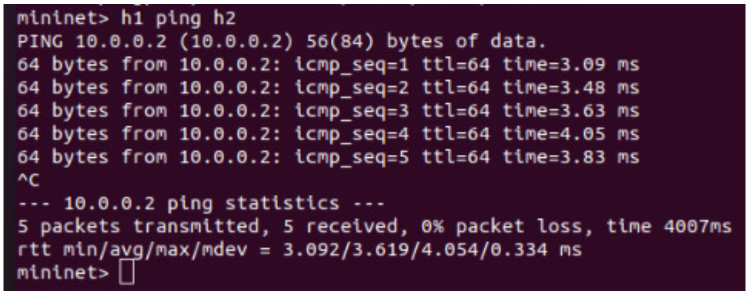
I have added below code in the of_tutorial.py file to display traffic in the connection.

```
        print("Swtich:", event.connection,"source address :", packet.src,"destination address:", packet.dst)
```

As an output we can see there is traffic in each switch. This is in accordance to the binary tree topology created. For "iperf h1 h2", s3 observes traffic & for "iperf h1 h8", s3, s2, s1, s5, s7 observe traffic.

Task 3: MAC Learning Controller

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).



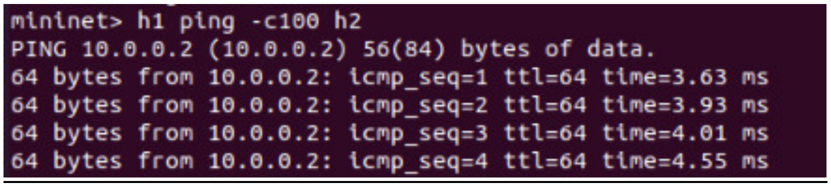
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.09 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3.48 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=3.63 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.05 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=3.83 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 3.092/3.619/4.054/0.334 ms
mininet> 
```

As "MAC to Port" map is established.

- When we enable the “act_like_switch” function, there is learning by mapping of each switch connection to a host.
- When the source MAC isn't in the map mac_to_port, script adds add the {MAC, Port} into the map mac_to_port.
- When the destination MAC is in the map, script sends the packet out the port associated with the MAC.
- When the destination MAC isn't in the map, script sends output to all ports except the input port.

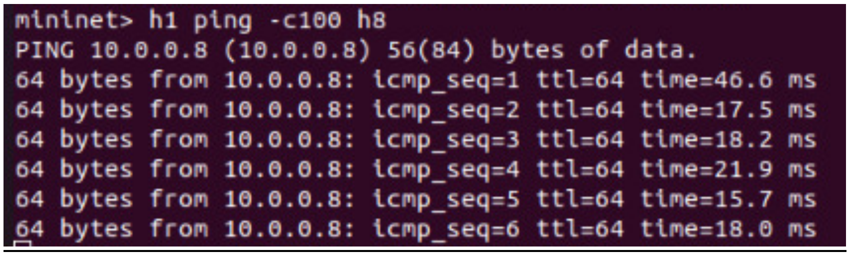
2. **(Comment out all prints before doing this experiment)** Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

h1 ping -c100 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=3.63 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3.93 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=4.01 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.55 ms
```

h1 ping -c100 h8

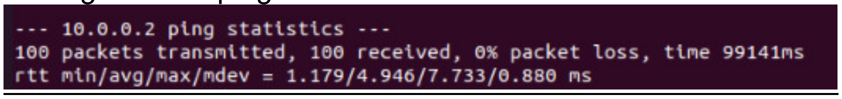


```
mininet> h1 ping -c100 h8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=46.6 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=17.5 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=18.2 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=21.9 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=15.7 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=18.0 ms
```

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

H1 ping h2 100 times:

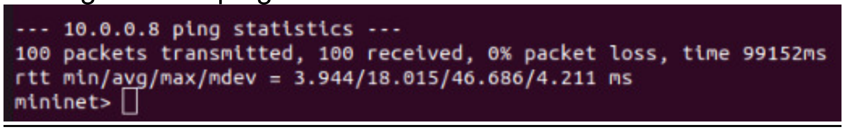
Average time to ping: 4.946ms



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99141ms
rtt min/avg/max/mdev = 1.179/4.946/7.733/0.880 ms
```

H1 ping h8 100 times:

Average time to ping: 18.015 ms



```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99152ms
rtt min/avg/max/mdev = 3.944/18.015/46.686/4.211 ms
mininet> 
```

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

H1 ping h2 100 times:

minimum and maximum ping observed: 1.179 ms & 7.733 ms

H1 ping h8 100 times.

minimum and maximum ping observed: 3.944 ms & 46.686 ms

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

Yes, there is a difference in all the three parameters:

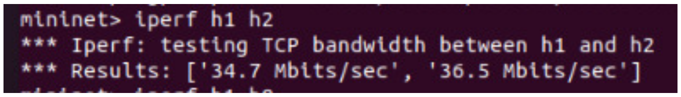
- average ping time: Task3 < Task 2.
- There is decrease in minimum ping in task 3 compared to Task 2.
- Overall, there is a decrease in ping time compared to task2. This because of “mac_to_port “is established. There is no flooding of packets everywhere. Now the specific port to transfer the packet is learnt.

	Task 2	**Task 3**
H1 ping h2 100 times: Average time to ping	5.527ms	4.946ms
H1 ping h8 100 times: Average time to ping	22.920 ms	18.015 ms
H1 ping h2 100 times: minimum and maximum	1.644 ms & 15.235 ms	1.179 ms & 7.733 ms
H1 ping h8 100 times: minimum and maximum	5.333 ms & 42.299 ms	3.944ms & 46.686 ms

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

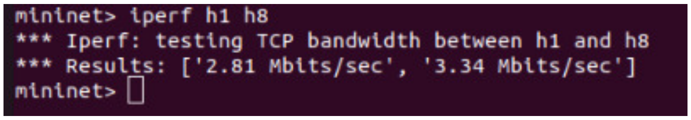
a. What is the throughput for each case?

“iperf h1 h2”



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['34.7 Mbits/sec', '36.5 Mbits/sec']
```

“iperf h1 h8”

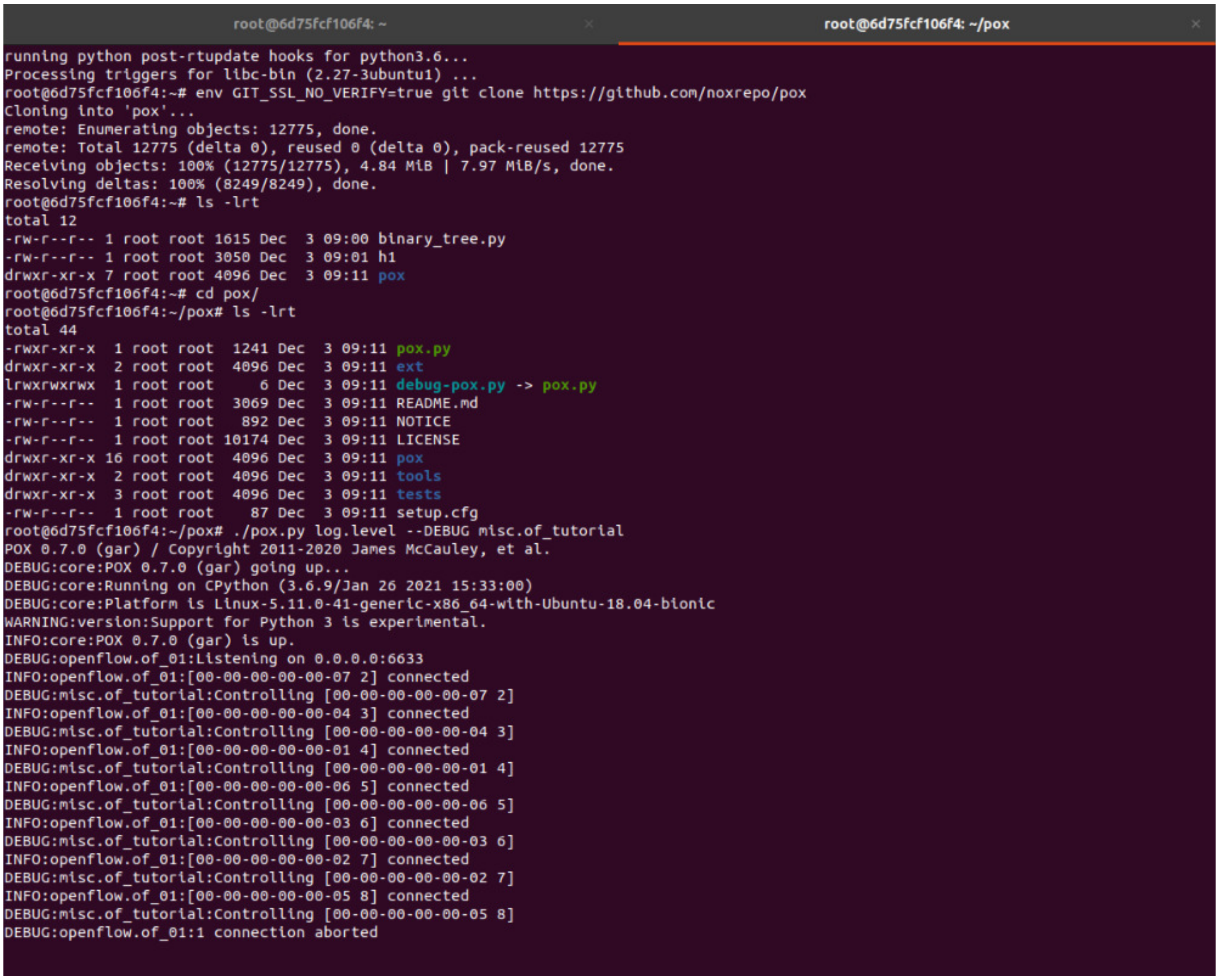


```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.81 Mbits/sec', '3.34 Mbits/sec']
mininet> 
```

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

Yes, there is visible difference in the throughput for “iperf h1 h2”. It has increased in case of Task3. This is because now after the “mac_to_port “is established there is no flooding of packets everywhere. Now the specific port to transfer the packet is learnt.

Install pox:



```
running python post-rtupdate hooks for python3.6...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
root@6d75fcf106f4:~# env GIT_SSL_NO_VERIFY=true git clone https://github.com/noxrepo/pox
Cloning into 'pox'...
remote: Enumerating objects: 12775, done.
remote: Total 12775 (delta 0), reused 0 (delta 0), pack-reused 12775
Receiving objects: 100% (12775/12775), 4.84 MiB | 7.97 MiB/s, done.
Resolving deltas: 100% (8249/8249), done.
root@6d75fcf106f4:~# ls -lrt
total 12
-rw-r--r-- 1 root root 1615 Dec  3 09:00 binary_tree.py
-rw-r--r-- 1 root root 3050 Dec  3 09:01 h1
drwxr-xr-x 7 root root 4096 Dec  3 09:11 pox
root@6d75fcf106f4:~# cd pox/
root@6d75fcf106f4:~/pox# ls -lrt
total 44
-rwxr-xr-x  1 root root  1241 Dec  3 09:11 pox.py
drwxr-xr-x  2 root root  4096 Dec  3 09:11 ext
lrwxrwxrwx  1 root root     6 Dec  3 09:11 debug-pox.py -> pox.py
-rw-r--r--  1 root root  3069 Dec  3 09:11 README.md
-rw-r--r--  1 root root   892 Dec  3 09:11 NOTICE
-rw-r--r--  1 root root 10174 Dec  3 09:11 LICENSE
drwxr-xr-x 16 root root  4096 Dec  3 09:11 pox
drwxr-xr-x  2 root root  4096 Dec  3 09:11 tools
drwxr-xr-x  3 root root  4096 Dec  3 09:11 tests
-rw-r--r--  1 root root    87 Dec  3 09:11 setup.cfg
root@6d75fcf106f4:~/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/Jan 26 2021 15:33:00)
DEBUG:core:Platform is Linux-5.11.0-41-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:[00-00-00-00-00-04 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-04 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:[00-00-00-00-00-06 5] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 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
```