CSEN 241: Cloud Computing HW 3

Title: Mininet and OpenFlow

By:

Name	Student ID	Email ID
Kartiki Rajendra Dindorkar	1651519	kdindorkar@scu.edu

I. System Configurations

Operating System	MAC OS Ventura 13.1
Chip	Intel Core i5
System Architecture	i386
Memory	8 GB
Number of Cores	2

II. GitHub Repository

[GitHub - CloudComputing]

III. Task 1: Defining Custom Topology

1. Run binary_tree.py

```
root@aaf0723f9523:/home# ls
binary_tree.py
root@aaf0723f9523:/home# 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>
```

2. 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=45.2 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.479 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.068 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.123 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.075 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.076 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.149 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=0.088 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=0.072 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=0.069 ms
64 bytes from 10.0.0.8: icmp_seq=14 ttl=64 time=0.077 ms
```

3. 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
mininet>
```

4. What is the output of “h7 ifconfig”

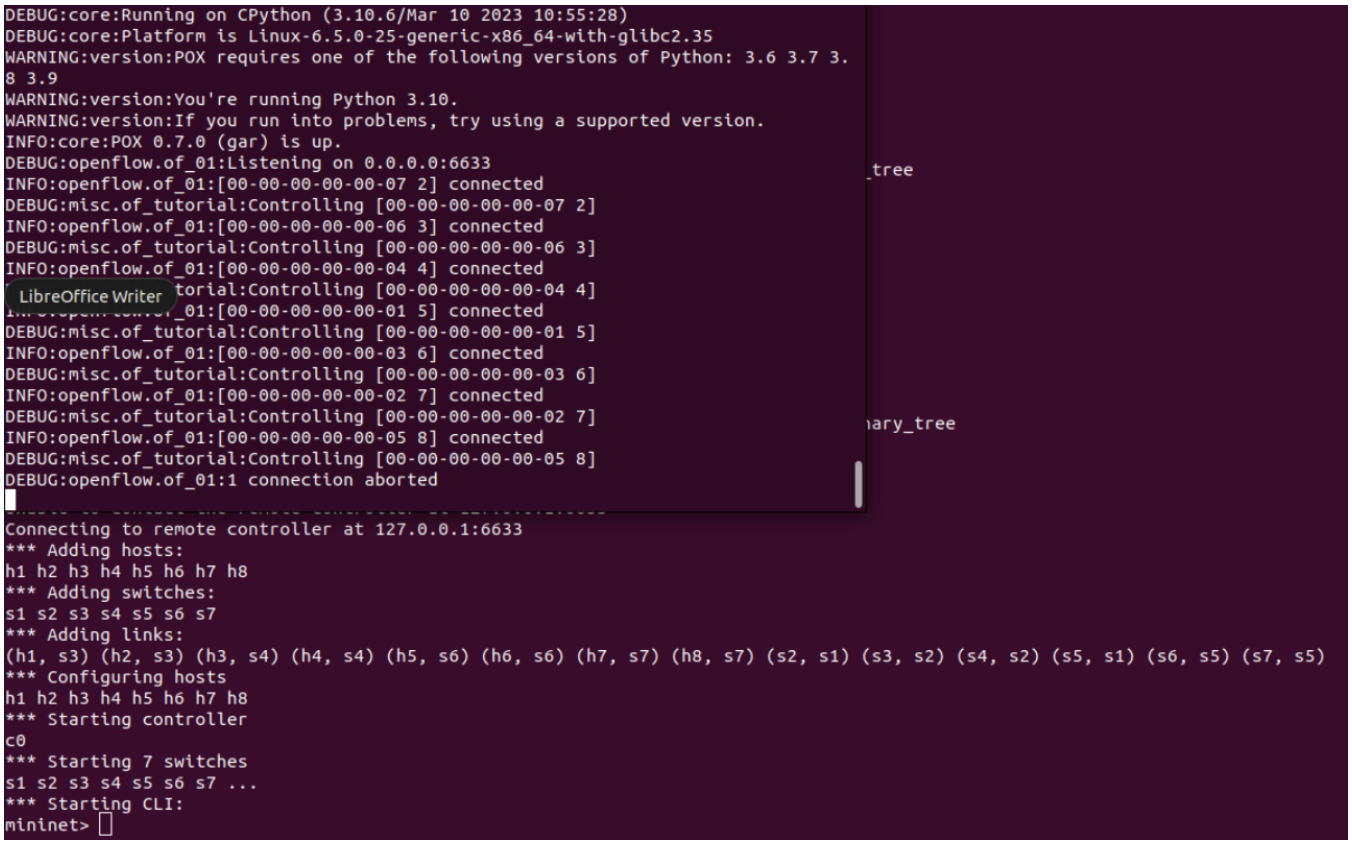
```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::44f1:caff:fe23:894a  prefixlen 64  scopeid 0x20<link>
        ether 46:f1:ca:23:89:4a  txqueuelen 1000  (Ethernet)
        RX packets 73  bytes 5394 (5.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1006 (1.0 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>
```

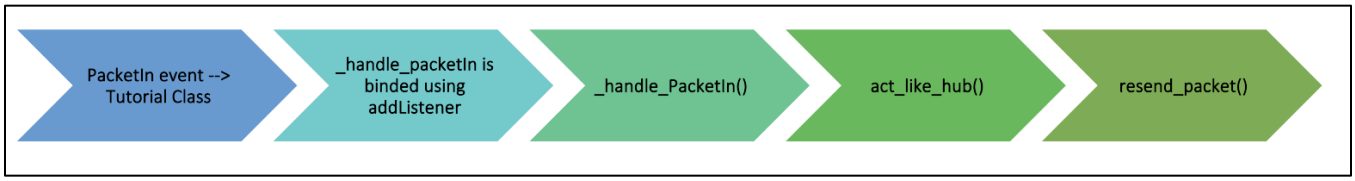
Help

5. Install pox and run `of_tutorial.py`



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?
    a. "`Tutorial`" class is designed to handle OpenFlow events.
    b. When a switch encounters a packet for which it has no matching entry in its flow table, it generates a "`PacketIn`" message and sends it to the controller.
    c. "`_handle_PacketIn`" function is registered as a listener for the "`PacketIn`" event. The registration is done in "`Tutorial`" class constructor i.e., inside the "`__init__`" method, using "`connection.addListerners(self)`" method.
    d. Controller i.e., "`of_tutorial.py`", after receiving "`PacketIn`" message from switch, "`_handle_PacketIn`" function is called automatically to handle the event.
    e. This function parses the packet and extracts the relevant information
    f. After that "`act_like_hub`" function is called from "`_handle_PacketIn`" function. This function resends the packet out of all the ports except the input port



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

```
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=9.26 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.342 ms
```

```
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=3.18 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=2.23 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99974ms
rtt min/avg/max/mdev = 1.405/2.621/6.005/0.812 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=19.2 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.873 ms
```

```
64 bytes from 10.0.0.8: icmp_seq=99 ttl=64 time=9.75 ms
64 bytes from 10.0.0.8: icmp_seq=100 ttl=64 time=11.5 ms

--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99025ms
rtt min/avg/max/mdev = 6.135/10.528/32.857/2.931 ms
mininet>
```

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

Sr No	Case	Time in ms
1.	h1 ping –c100 h2	2.621
2.	h1 ping –c100 h8	10.528

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

Sr No	Case	Min Time (ms)	Max Time (ms)
1.	h1 ping –c100 h2	1.405	6.005
2.	h1 ping –c100 h8	6.135	32.857

c. What is the difference, and why?
   i. The reason for the higher ping time from h1 to h8 compared to h1 to h2 is because h1 needs to go through more switches to reach h8.
   ii. Specifically, the path from h1 to h8 involves passing through five switches (s3, s2, s1, s5, and s7).
   iii. While the path to h2 only has one switch in between.
   iv. Each switch introduces some delay as it processes and forwards the packet, leading to higher latency when more switches are involved in the route.
   v. This difference in the number of switches along the path explains why the ping time is higher for h1 to h8.

3. Run "`iperf h1 h2`" and "`iperf h1 h8`"
   a. What is "iperf" used for?
      i. "iperf" is a tool for network performance measurement and tuning.
      ii. iperf is used to measure the maximum achievable bandwidth between two endpoints. By generating network traffic and measuring the throughput, iperf provides insights into the capacity of a network link.

   b. What is the throughput for each case?

Sr No	Case	Bandwidth (Mb/sec)
1.	iperf h1 h2	[10.4, 10.2]
2.	iperf h1 h8	[6.06, 5.91]

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.4 Mbits/sec', '10.2 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['6.06 Mbits/sec', '5.91 Mbits/sec']
mininet>
```

c. What is the difference and explain the reasons for the difference.
    i. The reason why more data can be sent faster from h1 to h2 compared to h1 to h8 is similar to why pings are faster between those pairs.
    ii. There's less traffic congestion, shorter distance, and less delay between h1 and h2. This means data can travel more smoothly and quickly between them.
    iii. Thus, the throughput, or the amount of data transferred per unit of time, is higher from h1 to h2 than h1 to 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)
    a. By adding loggers like `log.info("Switch observing traffic: %s" (self.connection))` inside the `_handle_PacketIn()` function of the "`of_tutorial.py`" file in the controller, we can track network traffic.
    b. This helps us understand that switches can monitor traffic, especially when they're handling a lot of packets. The function `_handle_PacketIn()` is triggered whenever a packet arrives, allowing us to examine the incoming traffic.

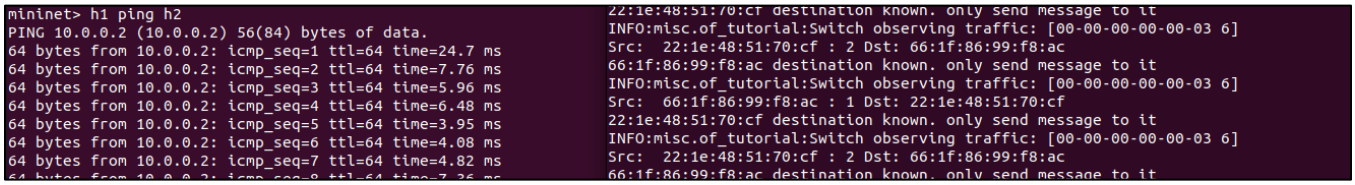
```
root@3874a37e3522:~/pox# ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.10.6/Mar 10 2023 10:55:28)
DEBUG:core:Platform is Linux-6.5.0-25-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-06 4] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-06 4]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-06 4]
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:[00-00-00-00-00-05 8] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-05 8]
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:misc.of_tutorial:Switch Observing traffic: [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:[00-00-00-00-00-04 5] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-04 5]
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:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-05 8]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-01 3]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-06 4]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-02 7]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-04 5]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-03 6]
INFO:misc.of_tutorial:Switch Observing traffic: [00-00-00-00-00-07 2]
```

IV. Task 3 – MAC Learning Controller

1. Updated "`act_like_switch()`" and made it available for running

```
root@3874a37e3522:~/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 (3.10.6/Mar 10 2023 10:55:28)
DEBUG:core:Platform is Linux-6.5.0-25-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
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
Terminal
c.of_tutorial:Controlling [00-00-00-00-00-03 6]
flow.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]
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-06 5]
Src:  ba:8b:d2:42:f8:ee : 1 Dst: 33:33:00:00:00:16
Learning that ba:8b:d2:42:f8:ee is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-04 3]
Src:  96:27:cd:99:02:cf : 1 Dst: 33:33:00:00:00:16
Learning that 96:27:cd:99:02:cf is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-05 8]
Src:  ba:8b:d2:42:f8:ee : 1 Dst: 33:33:00:00:00:16
Learning that ba:8b:d2:42:f8:ee is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-07 2]
Src:  ba:8b:d2:42:f8:ee : 3 Dst: 33:33:00:00:00:16
Learning that ba:8b:d2:42:f8:ee is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-01 4]
Src:  ba:8b:d2:42:f8:ee : 2 Dst: 33:33:00:00:00:16
Learning that ba:8b:d2:42:f8:ee is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-02 7]
Src:  96:27:cd:99:02:cf : 2 Dst: 33:33:00:00:00:16
Learning that 96:27:cd:99:02:cf is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
INFO:misc.of_tutorial:Switch observing traffic: [00-00-00-00-00-02 7]
Src:  ba:8b:d2:42:f8:ee : 3 Dst: 33:33:00:00:00:16
Learning that ba:8b:d2:42:f8:ee is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
```

2. 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).
    a. Upon receiving a packet, when "`act_like_switch()`" the receives the packet from "`handle_PacketIn()`" the function learns the source MAC address and associates it with the input port.
    b. If the destination MAC address is known, the switch forwards the packet directly to the associated port. Otherwise, it floods the packet to all ports except the input port. \
    c. This process allows the switch to dynamically establish a "MAC to Port" mapping, facilitating efficient packet forwarding in the network.
    d. This method minimizes unnecessary packet flooding.



1. (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?
        i. h1 ping h2

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99237ms
rtt min/avg/max/mdev = 1.557/3.215/27.107/3.075 ms
```

        ii. h1 ping h8

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99109ms
rtt min/avg/max/mdev = 6.015/11.739/50.429/5.646 ms
```

Sr No	Case	Time in ms
1.	h1 ping –c100 h2	3.215
2.	h1 ping –c100 h8	11.739

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

Sr No	Case	Min Time (ms)	Max Time (ms)
1.	h1 ping –c100 h2	1.557	27.107
2.	h1 ping –c100 h8	6.015	50.429

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

Sr No	Case	Task 2 AVG Time (ms)	Task 3 Avg Time (ms)
1.	h1 ping –c100 h2	2.621	3.215
2.	h1 ping –c100 h8	10.528	11.739

Sr No	Case	Task 2 Min Time (ms)	Task 2 Max Time (ms)	Task 3 Min Time (ms)	Task 3 Max Time (ms)
1.	h1 ping –c100 h2	1.405	6.005	1.557	27.107
2.	h1 ping –c100 h8	6.135	32.857	6.015	50.429

2. 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: ['54.3 Mbits/sec', '54.2 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.68 Mbits/sec', '3.59 Mbits/sec']
mininet>
```

Sr No	Case	Bandwidth (Mb/sec)
1.	iperf h1 h2	[54.3, 54.2]
2.	iperf h1 h8	[3.68, 3.59]

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

Sr No	Case	Task 2 Bandwidth (Mb/sec)	Task 3 Bandwidth (Mb/sec)
1.	iperf h1 h2	[10.4, 10.2]	[54.3, 54.2]
2.	iperf h1 h8	[6.06, 5.91]	[3.68, 3.59]

i. In case of h1 and h2 Task 3 performs better than task 2 because the MAC to port mapping reduces network traffic.
ii. Since the addresses and ports are already known, there's no need for extra requests, which prevents the switches from getting overwhelmed.
iii. For communication between h1 and h2, task 3 shows significant improvement due to less congestion.
iv. However, when the distance is longer, like from h1 to h8, there is no improvement because of packet dropping and the increased distance.