Alfonso De La Rosa
00001051360
COEN 241: Cloud Computing
Professor Sean Choi
03/13/2023

COEN 241 Homework 3

Introduction

The goal of this assignment is to create a fat-free topology network in Mininet, a network emulator, and to also use POX as an OpenFlow controller to analyze the performance of the network topology, with and without MAC learning. The performance of both networks are measured by ping values and TCP bandwidth values.

Task 1

Creating a Mininet network topology based on a custom topology

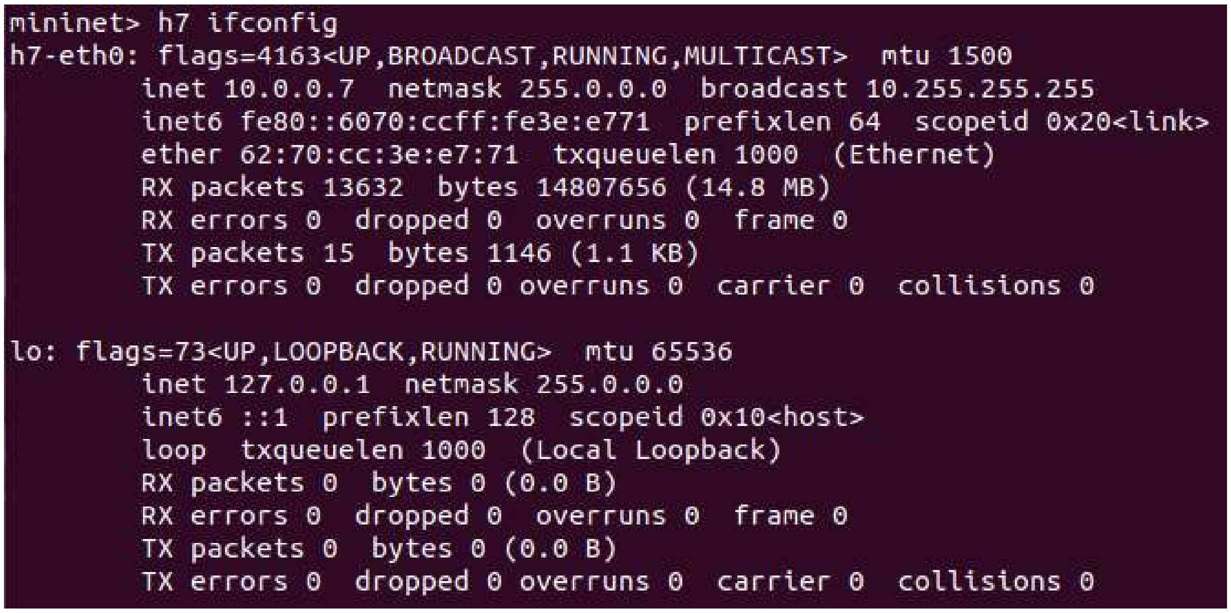
```
mininet> h1 ping h8

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=80.2 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=9.78 ms

64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=8.80 ms

64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=5.99 ms

64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=10.6 ms

64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=12.6 ms

64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=10.2 ms

64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=8.18 ms

64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=9.01 ms
```

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
```

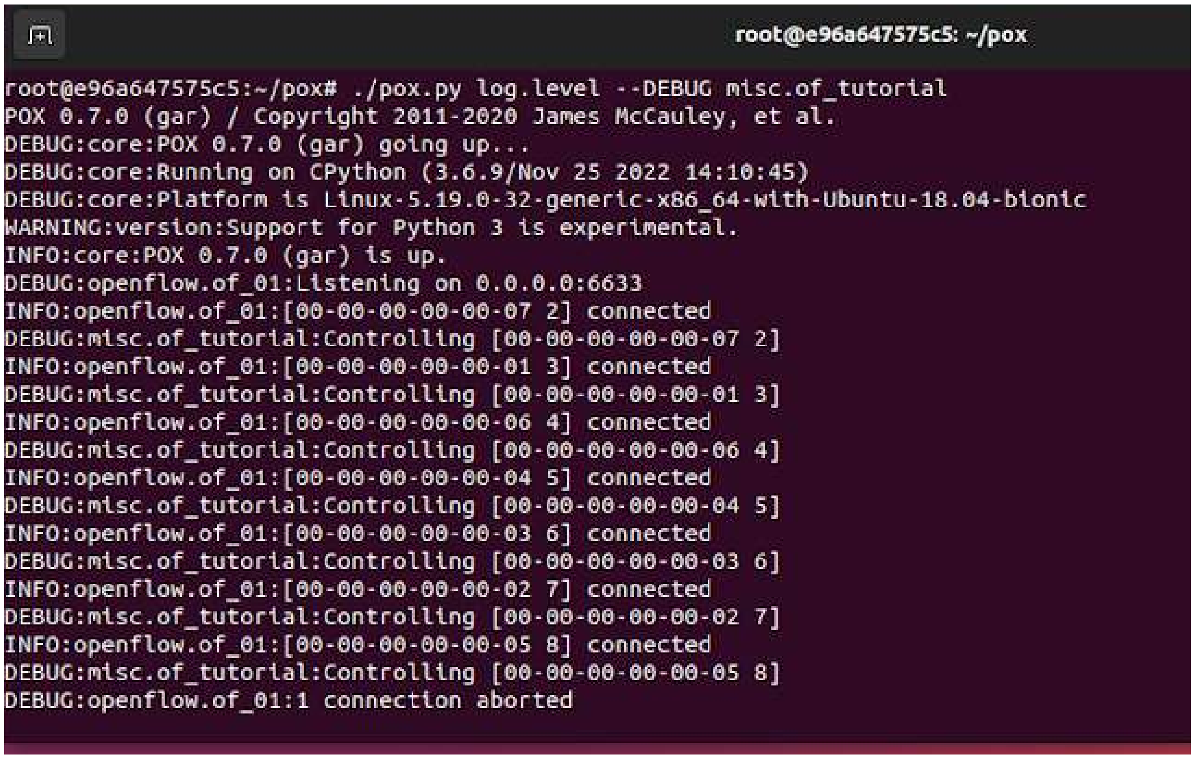
2. **What is the output of “h7 ifconfig”**



```
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::6070:ccff:fe3e:e771  prefixlen 64  scopeid 0x20<link>
        ether 62:70:cc:3e:e7:71  txqueuelen 1000  (Ethernet)
        RX packets 13632  bytes 14807656 (14.8 MB)
        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
```

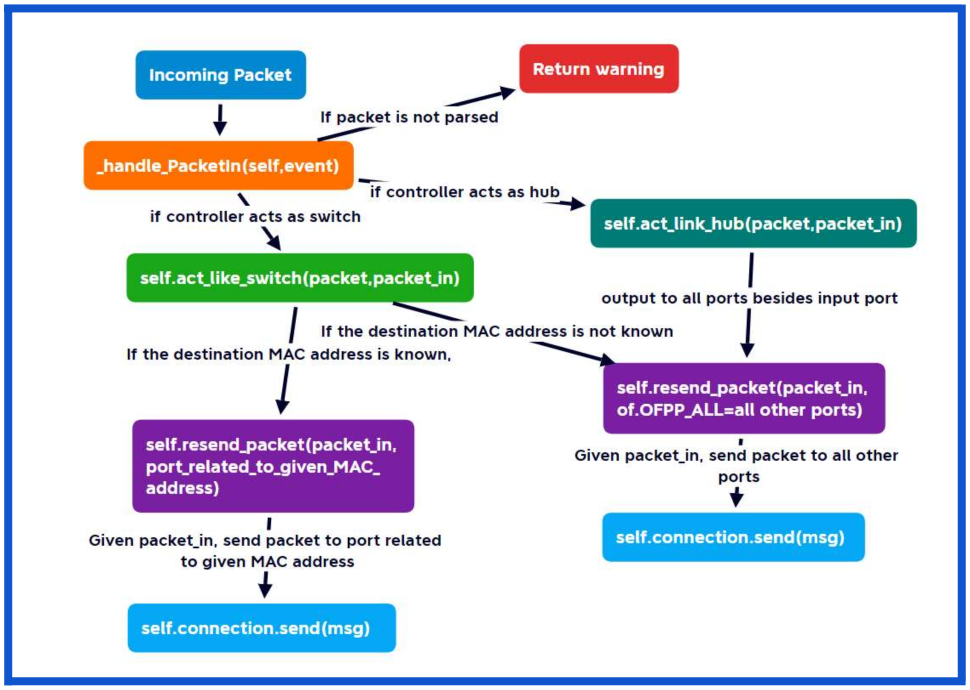
Task 2

Output of running the following command: ./pox.py log.level --DEBUG misc.of_tutorial



```
root@e96a647575c5:~/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/Nov 25 2022 14:10:45)
DEBUG:core:Platform is Linux-5.19.0-32-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:[00-00-00-00-00-01 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
```

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. Since the switches act as hubs in task 2, it will flood the network until it reaches its destination.


Incoming Packet
Return warning
If packet is not parsed
_handle_PacketIn(self,event)
if controller acts as hub
if controller acts as switch
self.act_link_hub(packet,packet_in)
self.act_like_switch(packet,packet_in)
output to all ports besides input port
If the destination MAC address is not known
If the destination MAC address is known,
self.resend_packet(packet_in, of.OFPP_ALL=all other ports)
self.resend_packet(packet_in, port_related_to_given_MAC_address)
Given packet_in, send packet to all other ports
self.connection.send(msg)
Given packet_in, send packet to port related to given MAC address
self.connection.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? What is the minimum and maximum ping you have observed?**

Case	min (ms)	max (ms)	mean (ms)
h1 ping h2	1.672	34.476	5.374
h1 ping h8	6.384	77.990	14.311

    b. **What is the difference, and why?**
    The main difference between these two cases is that the time it takes for **h1 to ping h2** is shorter than the time it takes for **h1 to ping h8**. The reason why is because there are less links for **h1 to ping h2** than there are for **h1 to ping h3**. The ping signal has to travel a longer distance, which takes a longer time.

3. **Run "iperf h1 h2" and "iperf h1 h8"**
    a. **What is "iperf" used for?**
    The command iperf measures the TCP bandwidth performance between two hosts, such as providing throughput values.

b. **What is the throughput for each case?**

	Throughput (Mbits/sec)
mininet> iperf h1 h2	*** Iperf: testing TCP bandwidth between h1 and h2 *** **Results: ['14.6 Mbits/sec', '16.1 Mbits/sec']**
mininet> iperf h1 h8	*** Iperf: testing TCP bandwidth between h1 and h8 *** **Results: ['5.28 Mbits/sec', '6.18 Mbits/sec']**

c. **What is the difference, and explain the reasons for the difference.**
The main difference between these two cases is that the throughput values for **iperf h1 h2** are larger than the throughput values for **iperf h1 h8**. The cause of this difference is due to the fact that there are more links between h1 and h8. In other words, the data would travel a longer distance from h1 to h8 than h1 to h2, leading to smaller bandwidth performance for 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).**
In order to see which switches observe traffic, or receive packets, I needed to implement a functionality that reports whenever a switch receives a packet. Since the **_handle_PacketIn** function handles the arrival of packets for a switch, I inserted the following code for the **_handle_PacketIn(self,event)**:

```
def _handle_PacketIn (self, event):
  """
  Handles packet in messages from the switch.
  """
  packet = event.parsed # This is the parsed packet data.

  if not packet.parsed:

    log.warning("Ignoring incomplete packet")

    return

  packet_in = event.ofp # The actual ofp_packet_in message.
  print("Packet received on switch {} on port {}.".format(str(event.dpid),
str(event.port)))

  # Comment out the following line and uncomment the one after

  # when starting the exercise.
```

```
self.act_like_hub(packet, packet_in)

#self.act_like_switch(packet, packet_in)
```

With this code, while the Mininet is running commands in one terminal, the **_handle_PacketIn** function will print the following statement in the separate terminal where POX is running whenever a switch receives a packet, which will report whenever a switch receives a packet:

"Packet received on switch {} on port {}."

When entering the following command on mininet on one terminal, the terminal where POX is running gives the subsequent output: **h1 ping -c1 h2**

Output:

```
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 3 on port 2.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 3 on port 2.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 2 on port 1.
Packet received on switch 7 on port 3.
Packet received on switch 1 on port 1.
Packet received on switch 6 on port 3.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 3 on port 2.
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 6 on port 3.
```

Therefore, since the current switches flood the packets to the other switches, running the **h1 ping -c1 h2** command, sends the packet to all switches. For the same reason, all the switches receive the packet when the following command is run for Mininet: **h1 ping -c1 h8**

Output:

```
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 7 on port 2.
Packet received on switch 5 on port 2.
Packet received on switch 6 on port 3.
Packet received on switch 1 on port 2.
Packet received on switch 2 on port 3.
Packet received on switch 4 on port 3.
Packet received on switch 3 on port 3.
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
Packet received on switch 7 on port 2.
Packet received on switch 5 on port 2.
Packet received on switch 1 on port 2.
Packet received on switch 6 on port 3.
Packet received on switch 2 on port 3.
Packet received on switch 4 on port 3.
Packet received on switch 3 on port 3.
Packet received on switch 7 on port 2.
Packet received on switch 5 on port 2.
Packet received on switch 1 on port 2.
Packet received on switch 6 on port 3.
Packet received on switch 2 on port 3.
Packet received on switch 4 on port 3.
Packet received on switch 3 on port 3.
Packet received on switch 3 on port 1.
Packet received on switch 2 on port 1.
Packet received on switch 1 on port 1.
Packet received on switch 4 on port 3.
Packet received on switch 5 on port 3.
Packet received on switch 7 on port 3.
Packet received on switch 6 on port 3.
```

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).**
   With the given code and setting the switches to act as switches instead of hubs, when a switch receives a packet, **_handle_PacketIn** function is called. Then, this function will call the **act_like_switch** function. With the code provided, the **act_like_switch** function has a map called “MAC to Port” which maps MAC addresses to a port number. The **act_like_switch** function executes the following commands whenever it receives a packet:
   a. If the source MAC address of a packet is not stored on the “MAC to Port” map, the switch links the source MAC address and the port the packet comes from, so that the switch knows that this port leads to this MAC address.
   b. Then, one of the following two situations occurs:
      i. If the destination address of the packet is stored on the “MAC to Port” map, the switch sends the packet to the port that will lead it to the destination MAC address because this information is stored on the “MAC to Port” map.
      ii. Otherwise, the switch floods the packet to all the other ports except the port that it comes from because the “MAC to Port” map does not have the port that leads to the packet’s destination MAC address.

   When the **h1 ping -c2 h2** command is executed, the terminal running POX prints an output. The following describes the behavior with the given code with the “MAC to Port.”
   c. In this case, 6e:79:65:49:dd:fd is the MAC address of host 1 and 0a:e4:0a:26:d8:ab is the MAC address of host 2.
   d. The first time h1 pings h2:
      i. Initially, all the switches flood the packets to all ports because they do not recognize the destination MAC address. At the same time, these switches are mapping the source MAC address (6e:79:65:49:dd:fd) to the port that it came onto the “MAC to Port” map.
      ii. Eventually, the packet reaches the destination MAC address (0a:e4:0a:26:d8:ab) via flooding.
      iii. Then, another packet is sent from the original destination MAC address (0a:e4:0a:26:d8:ab) back to the MAC address that originally sent the ping signal (6e:79:65:49:dd:fd). Because the adjacent switch receiving the second packet recognizes the MAC address (6e:79:65:49:dd:fd) since the

switch previously mapped the MAC address to the port on the "MAC to Port" map, this switch sends the second packed to the port listed on the "MAC to Port" map. Afterwards, the subsequent switches receiving this packet will send the packet to the port that leads to the new destination MAC address (6e:79:65:49:dd:fd) because it is listed on their "MAC to Port" maps.

1. Eventually, the second packet will reach the original host.

e. The second h2 pings h1:
    i. Since the switches on the path between host 1 and host 2 are familiar with which ports lead to the appropriate MAC addresses with the "MAC to Port" map, the switches will send the packets to the ports that lead to the destination.

Output

# after running h1 pings h2 the first time

```
Packet received on switch 3 on port 1.
Learning that 6e:79:65:49:dd:fd is attached at port 1
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 3 on port 2.
Learning that 0a:e4:0a:26:d8:ab is attached at port 2
6e:79:65:49:dd:fd destination known. only send message to it

Packet received on switch 2 on port 1.
Learning that 6e:79:65:49:dd:fd is attached at port 1
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 1 on port 1.
Learning that 6e:79:65:49:dd:fd is attached at port 1
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 4 on port 3.
Learning that 6e:79:65:49:dd:fd is attached at port 3
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 5 on port 3.
Learning that 6e:79:65:49:dd:fd is attached at port 3
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 7 on port 3.
Learning that 6e:79:65:49:dd:fd is attached at port 3
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 6 on port 3.
Learning that 6e:79:65:49:dd:fd is attached at port 3
0a:e4:0a:26:d8:ab not known, resend to everybody

Packet received on switch 3 on port 1.
0a:e4:0a:26:d8:ab destination known. only send message to it
```

```
Packet received on switch 3 on port 2.
6e:79:65:49:dd:fd destination known. only send message to it

# after running h1 pings h2 the second time

Packet received on switch 3 on port 1.
0a:e4:0a:26:d8:ab destination known. only send message to it

Packet received on switch 3 on port 2.
6e:79:65:49:dd:fd destination known. only send message to it
```

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

command	min (ms)	max (ms)	mean (ms)
h1 ping h2	1.518	18.523	4.289
h1 ping h8	5.022	107.070	13.735

    c. **Any difference from Task 2 and why do you think there is a change if there is?**
    Yes, there is a difference between the ping values between task 2 and task 3. The ping values are larger in task 2 because each switch acts as a hub while the ping values are smaller in task 3 because each switch acts as a smart switch with the “MAC to Port” map. In other words, it takes longer for the packets to travel to their destination in task 2 than in task 3. Due to the “MAC to Port” map, the packets are sent directly to the switches that lead to the destination MAC address in task 3. On the other hand, there are more packets flooded to switches that do not lead to the destination MAC address in task 2. As a result, the flooding of packets can slow down network performance, increasing ping values in task 2.

3. **Run “iperf h1 h2” and “iperf h1 h8”.**
    a. **What is the throughput for each case?**
        i.

	Throughput (Mbits/sec)
mininet> iperf h1 h2	*** Iperf: testing TCP bandwidth between h1 and h2 ***** Results: ['67.0 Mbits/sec', '68.4 Mbits/sec']**
mininet> iperf h1 h8	*** Iperf: testing TCP bandwidth between h1 and h8 ***** Results: ['5.62 Mbits/sec', '6.17 Mbits/sec']**

    b. **What is the difference from Task 2 and why do you think there is a change if there is?**
    Yes, there is a difference between the TCP bandwidths between task 2 and task 3. While the TCP bandwidth values are higher in task 3, the TCP bandwidth values are lower in task 2. This discrepancy is due to the fact that the network in task 3 uses the “MAC to Port” map, which task 2 does not. Here, TCP bandwidth can be thought of as the size of data being transmitted over a time period. In task 3, since the switches behave as smart switches and utilize the “MAC to Port” map, more packets are being sent in the right direction, increasing the TCP bandwidth values. On the other hand, since the switches behave as hubs and flood the packets in task 2, less packets are being sent in the right directions, lowering the TCP bandwidth values.