CSEN 241 HW3

Rishabh Agrawal
Student ID - W1651177
Github link -
https://github.com/agrawal-rishabh-manoj/CSEN-241-Cloud-Computing/tree/main/HW3

Task 1: Defining custom topologies

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> net
h1 h1-eth0:s3-eth2
h2 h2-eth0:s3-eth3
h3 h3-eth0:s4-eth2
h4 h4-eth0:s4-eth3
h5 h5-eth0:s6-eth2
h6 h6-eth0:s6-eth3
h7 h7-eth0:s7-eth2
h8 h8-eth0:s7-eth3
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s5-eth1
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:s4-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h1-eth0 s3-eth3:h2-eth0
s4 lo:  s4-eth1:s2-eth3 s4-eth2:h3-eth0 s4-eth3:h4-eth0
s5 lo:  s5-eth1:s1-eth2 s5-eth2:s6-eth1 s5-eth3:s7-eth1
s6 lo:  s6-eth1:s5-eth2 s6-eth2:h5-eth0 s6-eth3:h6-eth0
s7 lo:  s7-eth1:s5-eth3 s7-eth2:h7-eth0 s7-eth3:h8-eth0
```

2. What is the output of "h7 ifconfig"

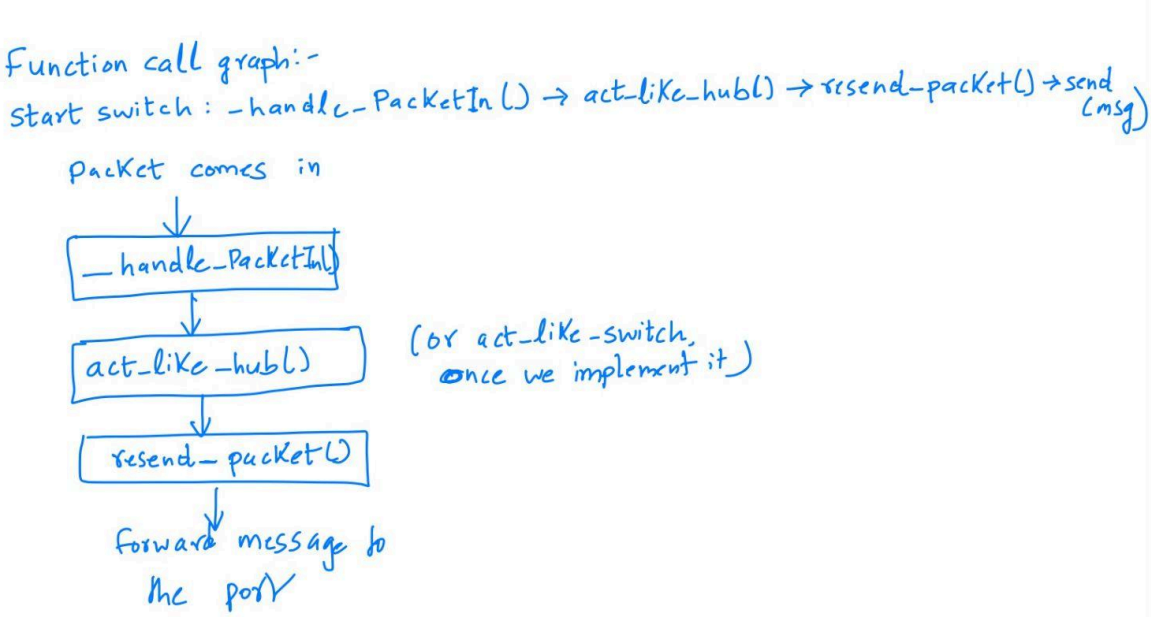
```
mininet> h7 ifconfig
h7-eth0   Link encap:Ethernet  HWaddr b6:70:32:c7:93:c5
          inet addr:10.0.0.7  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::b470:32ff:fec7:93c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24118 (24.1 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Task 2: Analyze the “of_tutorial’ controller

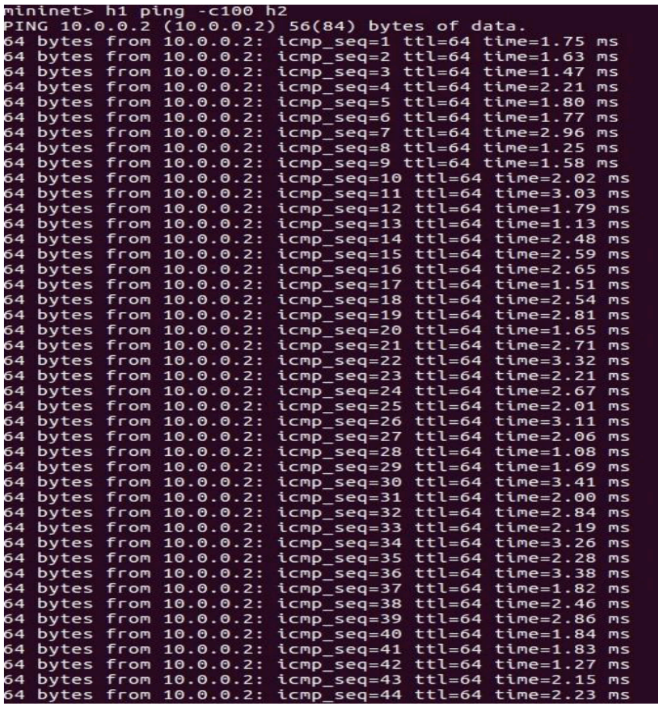
1. Draw the function call graph of this controller. For example, once a packet comes to the controller, which function is the first to be called, which one is the second, and so forth?

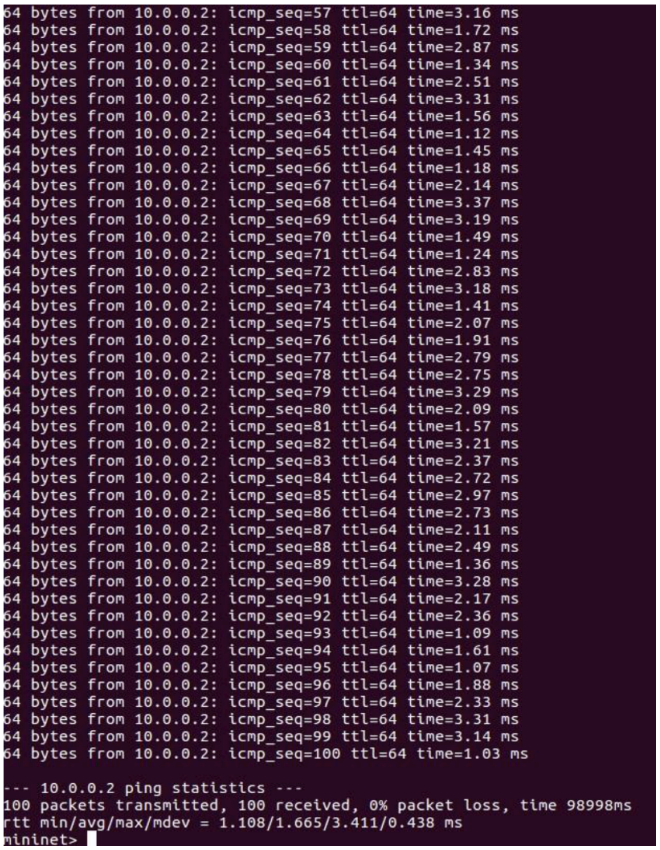
When initiating the POX listener with the command `./pox.py log.level --DEBUG misc.of_tutorial`, the process begins with the activation of the 'start switch' functionality. This initial step leads to the invocation of the `_handle_PacketIn()` method, which is responsible for handling incoming packets from the switch. Following this, the `_handle_PacketIn()` method calls the `act_like_hub()` method. This method mimics the actions of a network hub by forwarding packets to all ports, excluding the one it was received on. Subsequent to this, the `resend_packet()` method is called. This method adjusts the packet data accordingly and applies the designated action to it. To complete the process, the switch is instructed to forward the altered packet to a specified port, as per the command issued.



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

h1 ping -c100 h2:





h1 ping -c100 h8:

```
mininet> h1 ping -c100 h8
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=7.67 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=10.48 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=4.93 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=6.83 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=6.62 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=11.27 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=7.13 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=11.39 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=10.43 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=8.03 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=7.78 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=8.71 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=6.61 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=9.38 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=6.04 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=8.29 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=5.13 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=6.53 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=8.42 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=7.04 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=5.04 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=7.07 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=10.59 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=6.89 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=5.26 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=5.93 ms
64 bytes from 10.0.0.2: icmp_seq=27 ttl=64 time=6.68 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=5.25 ms
64 bytes from 10.0.0.2: icmp_seq=29 ttl=64 time=11.03 ms
64 bytes from 10.0.0.2: icmp_seq=30 ttl=64 time=8.45 ms
64 bytes from 10.0.0.2: icmp_seq=31 ttl=64 time=7.19 ms
64 bytes from 10.0.0.2: icmp_seq=32 ttl=64 time=7.76 ms
64 bytes from 10.0.0.2: icmp_seq=33 ttl=64 time=8.26 ms
64 bytes from 10.0.0.2: icmp_seq=34 ttl=64 time=8.11 ms
64 bytes from 10.0.0.2: icmp_seq=35 ttl=64 time=5.71 ms
64 bytes from 10.0.0.2: icmp_seq=36 ttl=64 time=11.68 ms
64 bytes from 10.0.0.2: icmp_seq=37 ttl=64 time=5.61 ms
64 bytes from 10.0.0.2: icmp_seq=38 ttl=64 time=11.22 ms
64 bytes from 10.0.0.2: icmp_seq=39 ttl=64 time=9.11 ms
64 bytes from 10.0.0.2: icmp_seq=40 ttl=64 time=9.98 ms
64 bytes from 10.0.0.2: icmp_seq=41 ttl=64 time=6.88 ms
64 bytes from 10.0.0.2: icmp_seq=42 ttl=64 time=7.42 ms
64 bytes from 10.0.0.2: icmp_seq=43 ttl=64 time=5.76 ms
64 bytes from 10.0.0.2: icmp_seq=44 ttl=64 time=7.37 ms
```

```
64 bytes from 10.0.0.2: icmp_seq=57 ttl=64 time=5.15 ms
64 bytes from 10.0.0.2: icmp_seq=58 ttl=64 time=5.34 ms
64 bytes from 10.0.0.2: icmp_seq=59 ttl=64 time=6.63 ms
64 bytes from 10.0.0.2: icmp_seq=60 ttl=64 time=9.31 ms
64 bytes from 10.0.0.2: icmp_seq=61 ttl=64 time=9.01 ms
64 bytes from 10.0.0.2: icmp_seq=62 ttl=64 time=11.15 ms
64 bytes from 10.0.0.2: icmp_seq=63 ttl=64 time=6.01 ms
64 bytes from 10.0.0.2: icmp_seq=64 ttl=64 time=4.55 ms
64 bytes from 10.0.0.2: icmp_seq=65 ttl=64 time=8.23 ms
64 bytes from 10.0.0.2: icmp_seq=66 ttl=64 time=10.69 ms
64 bytes from 10.0.0.2: icmp_seq=67 ttl=64 time=5.91 ms
64 bytes from 10.0.0.2: icmp_seq=68 ttl=64 time=6.98 ms
64 bytes from 10.0.0.2: icmp_seq=69 ttl=64 time=11.22 ms
64 bytes from 10.0.0.2: icmp_seq=70 ttl=64 time=10.93 ms
64 bytes from 10.0.0.2: icmp_seq=71 ttl=64 time=7.71 ms
64 bytes from 10.0.0.2: icmp_seq=72 ttl=64 time=7.93 ms
64 bytes from 10.0.0.2: icmp_seq=73 ttl=64 time=9.14 ms
64 bytes from 10.0.0.2: icmp_seq=74 ttl=64 time=10.36 ms
64 bytes from 10.0.0.2: icmp_seq=75 ttl=64 time=8.86 ms
64 bytes from 10.0.0.2: icmp_seq=76 ttl=64 time=6.58 ms
64 bytes from 10.0.0.2: icmp_seq=77 ttl=64 time=8.71 ms
64 bytes from 10.0.0.2: icmp_seq=78 ttl=64 time=11.77 ms
64 bytes from 10.0.0.2: icmp_seq=79 ttl=64 time=11.37 ms
64 bytes from 10.0.0.2: icmp_seq=80 ttl=64 time=10.74 ms
64 bytes from 10.0.0.2: icmp_seq=81 ttl=64 time=7.98 ms
64 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=7.32 ms
64 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=8.68 ms
64 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=7.69 ms
64 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=5.18 ms
64 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=9.25 ms
64 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=8.21 ms
64 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=5.82 ms
64 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=7.46 ms
64 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=8.17 ms
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=7.43 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=7.81 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=6.41 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=8.92 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=11.47 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=8.91 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=6.67 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=6.78 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=10.88 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=6.45 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99855ms
rtt min/avg/max/mdev = 4.551/5.466/12.144/1.530 ms
mininet>
```

a. How long does it take (on average) to ping for each case?
h1 ping h2 > 1.665 ms
h1 ping h8 > 5.466 ms

b. What is the minimum and maximum ping you have observed?
h1 ping h2 > Minimum 1.108 ms, maximum 3.411 ms
h1 ping h8 > Minimum 4.551 ms, maximum 12.144 ms

c. What is the difference, and why?
The time required for a ping from h1 to h8 is greater than the time it takes to ping from h1 to h2. This difference is attributed to the network topology; h1 and h2 are connected through a single switch, s3, making the path shorter. In contrast, packets traveling from h1 to h8 must pass through several intermediate devices, including switches s3, s2, s1, s5, and s7, resulting in a longer journey.

3. Run "iperf h1 h2" and "iperf h1 h8"
a. What is "iperf" used for?
Iperf is a free tool that helps network administrators evaluate network performance and connection quality by assessing bandwidth. It uses two hosts equipped with iperf to saturate the network link, which facilitates the measurement of data transfer rates between any two points in the network.

b. What is the throughput for each case?

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.7 Mbits/sec', '12.5 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.15 Mbits/sec', '3.52 Mbits/sec'
mininet>
```

c. What is the difference, and explain the reasons for the difference.
The improved data transfer rate observed between h1 and h2, as opposed to that between h1 and h8, can be attributed to network congestion and latency issues (which likewise lead to increased ping durations). The direct path from h1 to h2, involving fewer intermediate nodes, enables quicker data transmission. Conversely, the more extensive route from h1 to h8, with its additional hops, restricts the volume of data that can be sent over the same period.

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).
Incorporating the code `log.info("Switch observing traffic: %s" % (self.connection))` at line 107 in the "of_tutorial" controller enables us to monitor traffic flow. From the insights gained, it becomes evident that every switch in the network observes the traffic, especially under conditions where they are inundated with packet floods. The `_handle_PacketIn` function acts as an event listener, triggering whenever a packet arrives.

Task 3: MAC Learning Controller

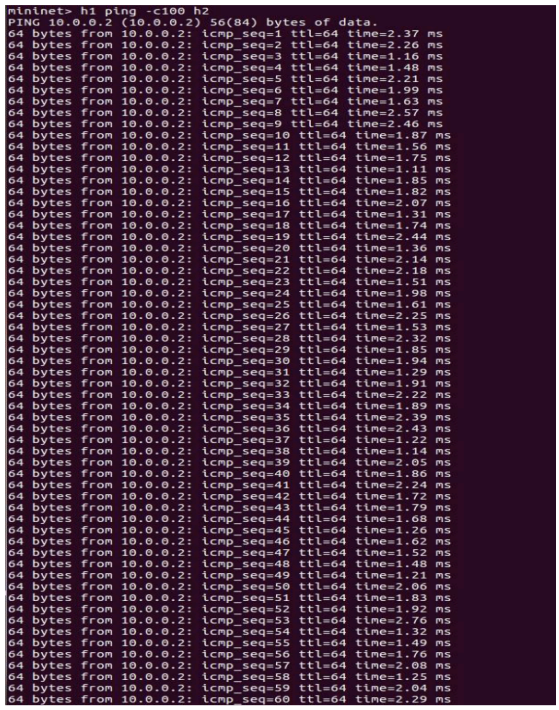
1. Describe how the above code works, such as how the "MAC to Port" map is established. You could use a ‘ping’ example to describe the establishment process (e.g., h1 ping h2).
The `act_like_switch` function possesses the capability to "learn" or associate MAC addresses with specific ports, allowing the controller to route packets more efficiently towards their designated destinations. Upon recognizing a target MAC address, the controller assigns it to a particular port to facilitate direct packet delivery. This mechanism improves the controller's efficiency in handling packets directed to familiar addresses by eliminating the need to broadcast the packet across multiple ports. However, when faced with an unfamiliar destination address, the function resorts to broadcasting the packet to all possible destinations. By reducing unnecessary broadcasting, the MAC Learning Controller significantly improves ping response times and data transfer rates. Observing the output from a single packet ping between h1 and h2 can shed light on how this learning process is implemented.

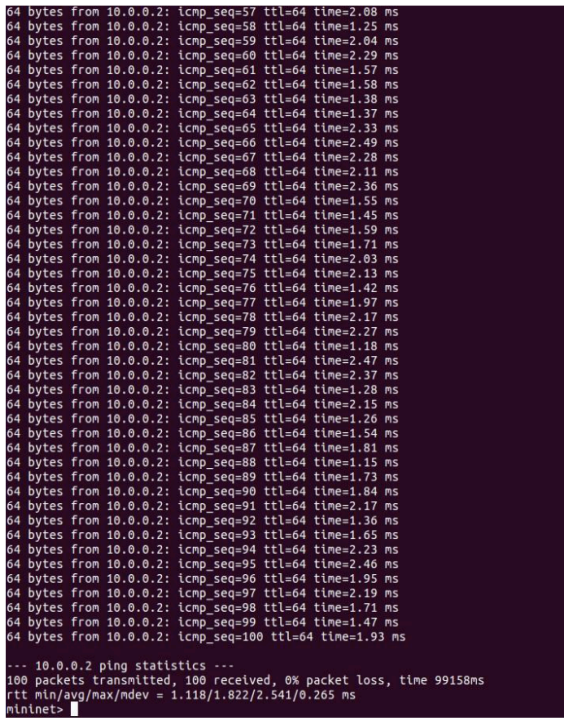
```
root@ef2a84b8z17e:~/pox# ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.5.2/Nov 12 2018 13:43:14)
DEBUG:core:Platform is Linux-4.4.0-21-generic-x86_64-with-Ubuntu-16.04-xenial
WARNING:version:You're running Python 3.5.
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:[00-00-00-00-00-06 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
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 1
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 12:3b:c3:3e:0b:62 is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that 12:3b:c3:3e:0b:62 is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that 12:3b:c3:3e:0b:62 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that 12:3b:c3:3e:0b:62 is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that 12:3b:c3:3e:0b:62 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that 12:3b:c3:3e:0b:62 is attached at port 1
```

```
33:33:00:00:00:16 not known, resend to everybody
Learning that a6:e8:e9:dd:39:24 is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
33:33:ff:0c:09:dc not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 3
33:33:00:00:00:16 not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 2
33:33:00:00:00:16 not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
Learning that 1a:d1:21:83:fe:bb is attached at port 1
33:33:00:00:00:16 not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 2
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 3
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 1
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 2
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 1
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 1
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:ff:e2:a7:63 not known, resend to everybody
Learning that fa:94:b9:5b:2f:4d is attached at port 1
33:33:ff:5b:2f:4d not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
```

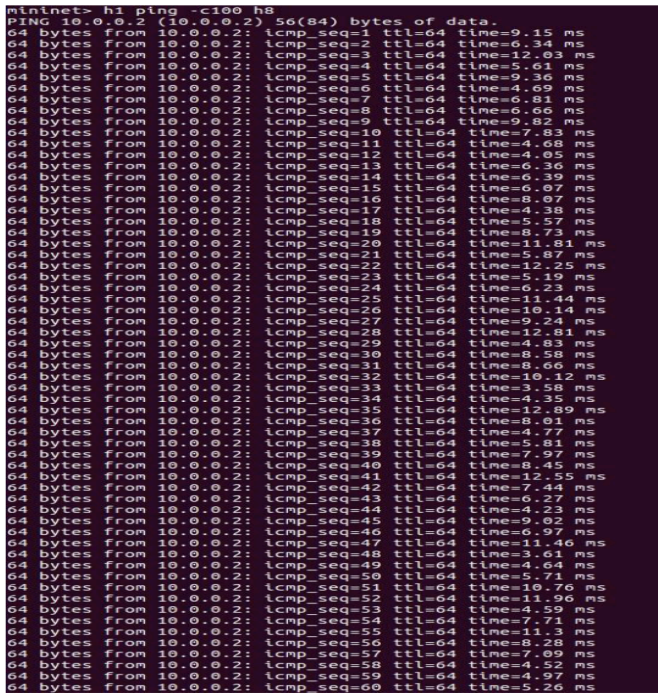
2. (Comment out all prints before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

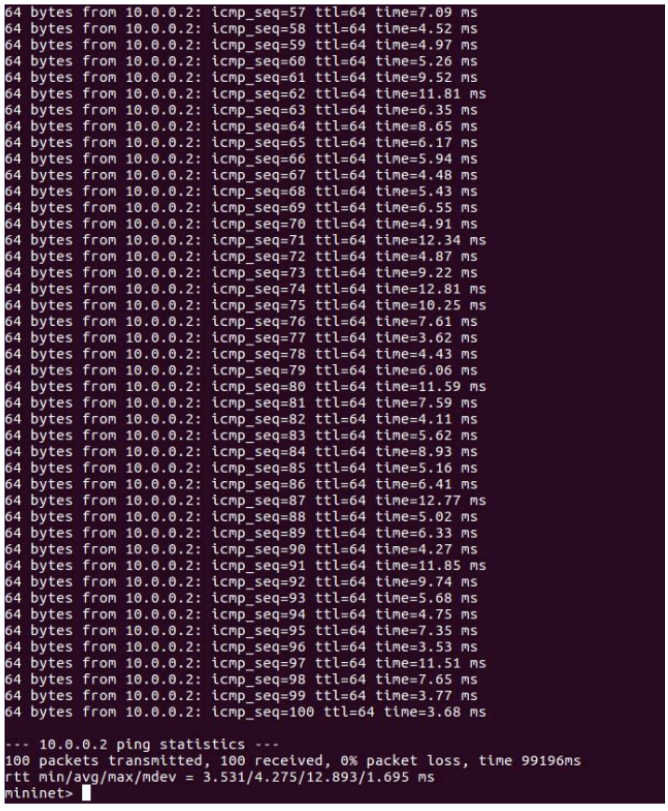
h1 ping -c100 h2:





h1 ping -c100 h8:





a. How long did it take (on average) to ping for each case?
h1 ping h2 > 1.822 ms
h1 ping h8 > 4.275 ms

b. What is the minimum and maximum ping you have observed?
h1 ping h2 > Minimum 1.118 ms, maximum 2.541 ms
h1 ping h8 > Minimum 3.531 ms, maximum 12.893 ms

c. Any difference from Task 2 and why do you think there is a change if there is?
Relative to task 2, the duration for h1 to ping h2 in task 3 is marginally reduced, with the variance being minimal. Nonetheless, when comparing the ping times between h1 and h8, a significant discrepancy is observed, which can be ascribed to the increased count of switches involved. The improvement in ping times during task 3 stems from the strategy of flooding only the initial packets. Following the identification and association of the destination MAC address, the switches proceed to direct packets exclusively to the pre-determined port specified in the "mac_to_port" mapping. This approach leads to diminished network congestion for later pings, resulting in quicker transmission times.

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: ['30.1 Mbits/sec', '34.2 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.54 Mbits/sec', '4.15 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 instances, task 3 yields a higher data transfer rate compared to task 2, chiefly because of the decreased network congestion seen in task 3. This reduction in congestion is a consequence of limiting packet flooding once the `mac_to_port` mapping has successfully learned all necessary ports, thereby reducing the load on the switches. Between h1 and h2, there's a notable tripling in throughput in task 3 versus task 2, attributed to the controller's ability to utilize pre-determined and learned paths more effectively. While the enhancement in throughput from h1 to h8 isn't as pronounced, there's still a marginal improvement, possibly influenced by the increased number of intermediate switches and potential packet loss.