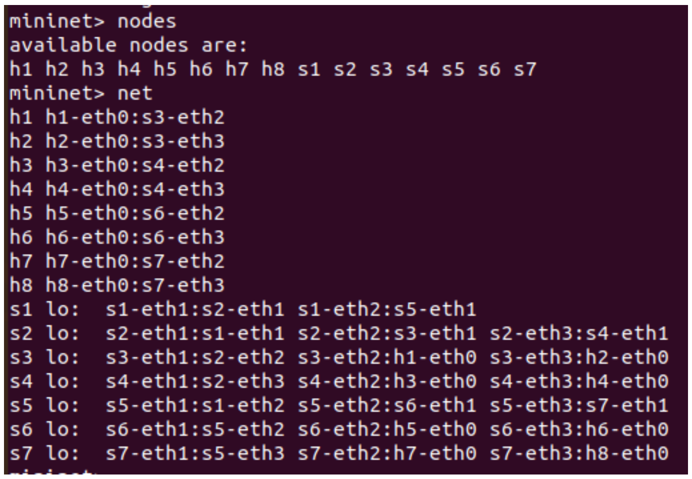
TASK1:

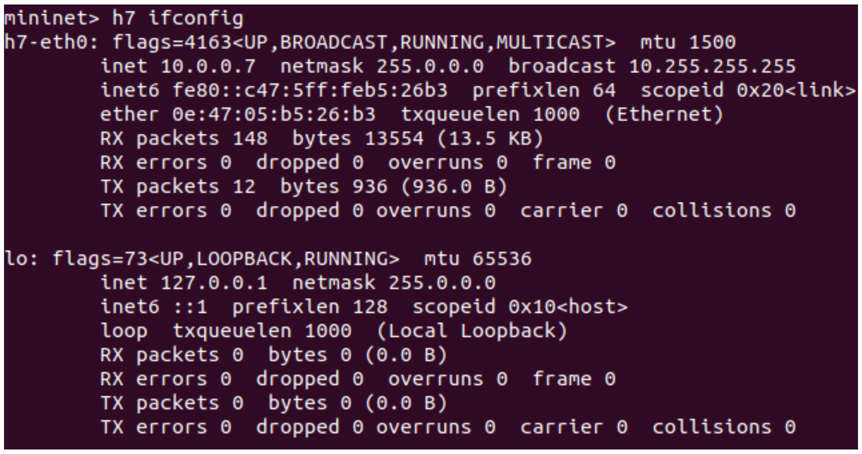
questions:

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: 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::c47:5ff:feb5:26b3  prefixlen 64  scopeid 0x20<link>
        ether 0e:47:05:b5:26:b3  txqueuelen 1000  (Ethernet)
        RX packets 148  bytes 13554 (13.5 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
```

TASK2:

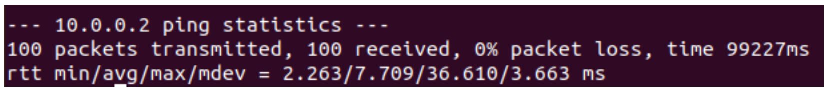
questions:

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?

Launch() -> start_switch(event) -> Tutorial(event.connection) -> __init__ (self, connection) -> _handle_PacketIn () -> act_like_hub () -> resend_packet()

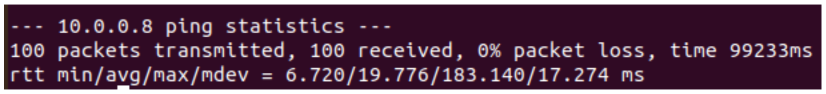
2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99227ms
rtt min/avg/max/mdev = 2.263/7.709/36.610/3.663 ms
```

h1 ping -c100 h8:



```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99233ms
rtt min/avg/max/mdev = 6.720/19.776/183.140/17.274 ms
```

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

h1 -> h2: 7.708

h1 -> h8: 19.776

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

h1 -> h2: min: 2.263 max: 36.610

h1 -> h8: min: 6.720 max: 183.140

c. What is the difference, and why?

h1 -> h2: need to go through 1 switch(h1->s3->h2)

h1 -> h8: need to go through 5 switches(h1->s3->s2->s1->s5->s7->h8)

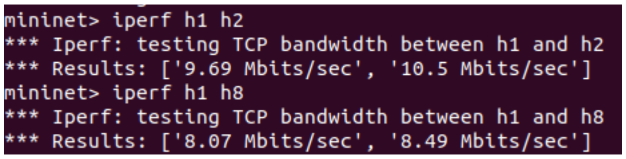
so h1 -> h8 needs more time

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

a. What is “iperf” used for?

test TCP bandwidth between two hosts

b. What is the throughput for each case?



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.69 Mbits/sec', '10.5 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['8.07 Mbits/sec', '8.49 Mbits/sec']
```

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

The throughput for h1 -> h2 is slightly better than the throughout for h1 -> h8, because h1 and h2 are a little closer, but the distance doesn’t make a big difference on TCP bandwidth in this situation.

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).

All of the switches observe traffic. Because act_like_hub() sends packets to all the switches in the network.

If we want to observe this traffic, we could get id of switch by self.connection.dpid command, and add a print statement in the _handle_PacketIn().

TASK3:

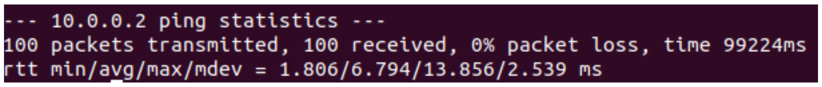
questions:

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).

When a packet comes in. the program will check if the mac and port pair already exists in mac_to_port dictionary. If so, send directly to the destination. If not, it will flood the packet to all hosts to find the destination, and it will store the mac-port pair in mac_to_port dictionary to speed up next call.

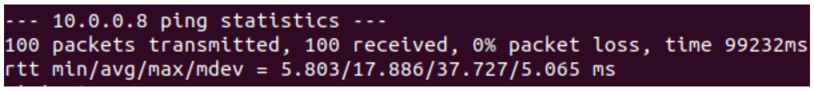
2. (Comment out all prints before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99224ms
rtt min/avg/max/mdev = 1.806/6.794/13.856/2.539 ms
```

h1 ping -c100 h8:



```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99232ms
rtt min/avg/max/mdev = 5.803/17.886/37.727/5.065 ms
```

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

h1 -> h2: 6.794

h1 -> h8: 17.886

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

h1 -> h2: min: 1.806 max: 13.856

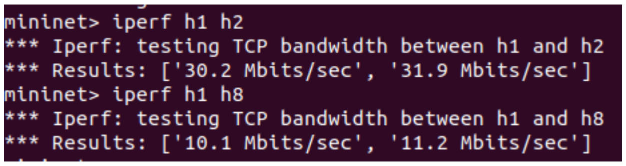
h1 -> h8: min: 5.803 max: 37.727

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

The change is the pings in Tsak3 reduce slightly comparing to that in Task2. This change is because now the controller is able to get MAC addresses so that they can send directly to destination port with the help of mac_to_port dictionary.

3. Q.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: ['30.2 Mbits/sec', '31.9 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['10.1 Mbits/sec', '11.2 Mbits/sec']
```

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

The throughput in both cases is faster than that in Task 2, especially in link h1 -> h2. Again, this is because the MAC learning which enables mapping of MAC addresses and destination port. Thus congestion is reduced and in turn increases throughput.