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
mininet>
```

2. What is the output of “h7 ifconfig”

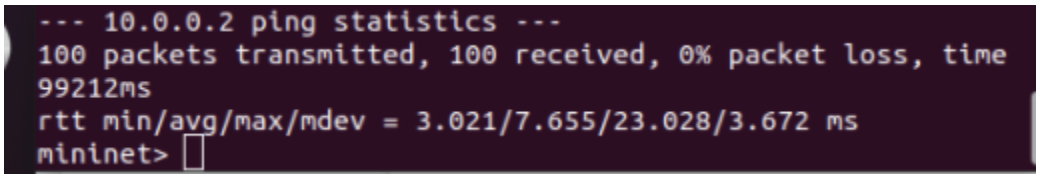
```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::4883:e3ff:fe9f:7d0f  prefixlen 64  scopeid 0x20<link>
        ether 4a:83:e3:9f:7d:0f  txqueuelen 1000  (Ethernet)
        RX packets 62  bytes 4688 (4.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 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

mininet>
```

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?

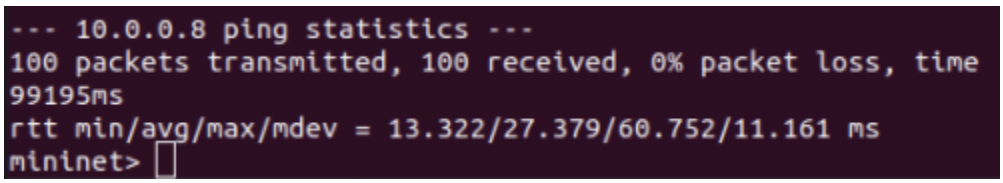
The function call graph of the controller is as below:

- The controller starts with launch() function which starts the component
- And then we have the listener which will call start_switch when connection is up
- Tutorial(connection) constructor - will create a tutorial object for each switch that it connects to
- _handle_packetIn will handle packet in messages from the switch
- Parse the packet.Get the actual ofp_packet_in message
- Call act_like_hub
- Act_like_hub calls resend_packet with of.OFOPP_ALL to send all packets to all ports besides input port
- Resend_packe with out_port as OFOPP_ALL will instruct the switch to resend the packet to all ports
- self.connection.send(msg) - Sends the message

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



h1 ping -c 100 h2
h1 ping -c 100 h8



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

H1 -> h2 7.655 ms
H1 -> h8 27.379 ms

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

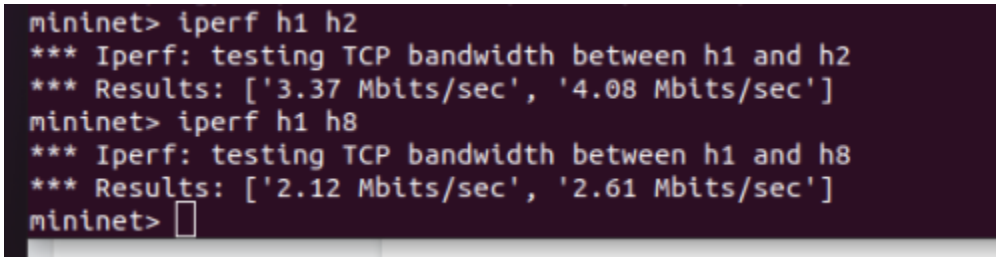
H1 -> h2 min: 3.021 ms | max: 23.028 ms
H1 -> h8 min: 13.322 ms | max: 60.752 ms

c. What is the difference, and why?

h1-> h2 finishes faster than h1->h8 as h1 to h2 there is only one switch s3 whereas h1->h8, there are multiple switches and hence multiple hops and in each switch flooding happens h1 -> s3 -> s2 -> s1 -> s5 -> s7 -> h8. The time takes is less for h1->h2

3.

Run “iperf h1 h2” and “iperf h1 h8”



a.

What is “iperf” used for?

Iperf is used to test the bandwidth(TCP bandwidth) for performance evaluation

b.

What is the throughput for each case?

h1-> h2 4.08 Mbits/sec
h1->h8 2.61 Mbits/sec

c.

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

The throughput for h1->h2 is higher compared to h1->h8. This is because reaching h1 from h2 is Quicker and has less number of hops as there is only one switch s3 in between. But for h1 to h8 , There are a lot of hops and switches to go through which increases the time; latency increases And as the traffic for the switch increases, congestion is also bound to happen. As this is a TCP bandwidth test, for each request there is a response ACK which adds on to the traffic

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 the switches observe traffic as this is act_like_hub behaviour. All packets get sent to all Switches. Simple way of getting the dpid of the switch and printing it in the handle_packet_in function. You will see that all the switches keep getting printed showing that all switches observer traffic

switch_n = “s”+str(self.connection.dpid)

log.debug(“At switch … ”+switch_n)

To observe what the traffic is in each switch, we can print out details of the packet in each by Printing Event.parsed and event.ofp in handle_packetIn function. I have not printed those as the question is only to identify the switches

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).

First h1 ping -c 1 h2

- The of_tutorial acts like a switch
- Mac_to_port dictionary will store list of mac address to port as when the controller learns
- First when h1 ping h2 is executed via switch s3, the mac_to_port is empty
- We look if the mac address to port mapping exists for h1 in mac_to_port
- It’s not there. Hence we add it
- Next, we check if the destination’s mac address to port mapping is available in mac_to_port to
- know which port to send it to
- As it’s not present, the switch will send the packet (flooding) to all ports except the input port
- All the other ports wont send any response
- H2 from its port will send an ACK to h1 via switch
- For switch, h2 is the source now. It will follow the previous steps, and see if the src (h2) mac and
- port is available in mac_to_port dictionary
- It’s not available. Hence it will add.
- For this ACK, the destination is h1 and its mac to port is already added
- Switch figures the port is known and sends the reply directly to h1.

Second h1 ping -c 1 h2

the mac_to_port already contains h1 mac to port. Hence it will
not add. It will check if the destination mac to port is known. It is available in mac_to_port as it was already added before in the previous ping. It will send the packet directly to h2 without flooding. i.e without acting like a hub.

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?

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time
99213ms
rtt min/avg/max/mdev = 2.447/6.873/12.318/2.367 ms
mininet>
```

H1 -> h2 avg: 6.783 ms

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time
99176ms
rtt min/avg/max/mdev = 11.165/27.377/54.693/9.071 ms
mininet>
```

H1 -> h8 27.377 ms

b.

What is the minimum and maximum ping you have observed?

H1 -> h2 min: 2.447 ms | max: 12.318 ms
H1 -> h8 min: 11.165ms | max: 54.693 ms

c.

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

Yes, task 3 performs better overall with smaller min and max time. Min is 3 and 13 ms whereas it's 2 and 11 ms in task 3. maximum it took was 12 ms and 54 ms with switch whereas it was 16 ms and 60 ms for hub like behavior. Switch performs better as it learns every time and doesn't flood the network all the time like a hub. Hub increases the network traffic and broadcast the packet all the time whereas switch once it saves the mac to port, it sends the message to the right port

3.

Q.3 Run "iperf h1 h2" and "iperf h1 h8".

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['12.8 Mbits/sec', '14.7 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.47 Mbits/sec', '3.03 Mbits/sec']
mininet>
Interrupt
```

a.

What is the throughput for each case?

H1 -> h2 : 14.7 Mbit/sec
H1 -> h8 : 3.03 Mbit/sec

b.

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

The throughput is higher in task 3. 14 and 3 mbit/sec vs 4 and 2 mbit/sec in task 2. This is because, task 3 has switch-like behavior and keeps populating mac_to_port dictionary. Because of this, once the destination port is known, the useful bandwidth can be used for transmitting packets to proper destination instead of flooding all the hosts. This decreases the network traffic, congestion and latency. Hence throughput increases.