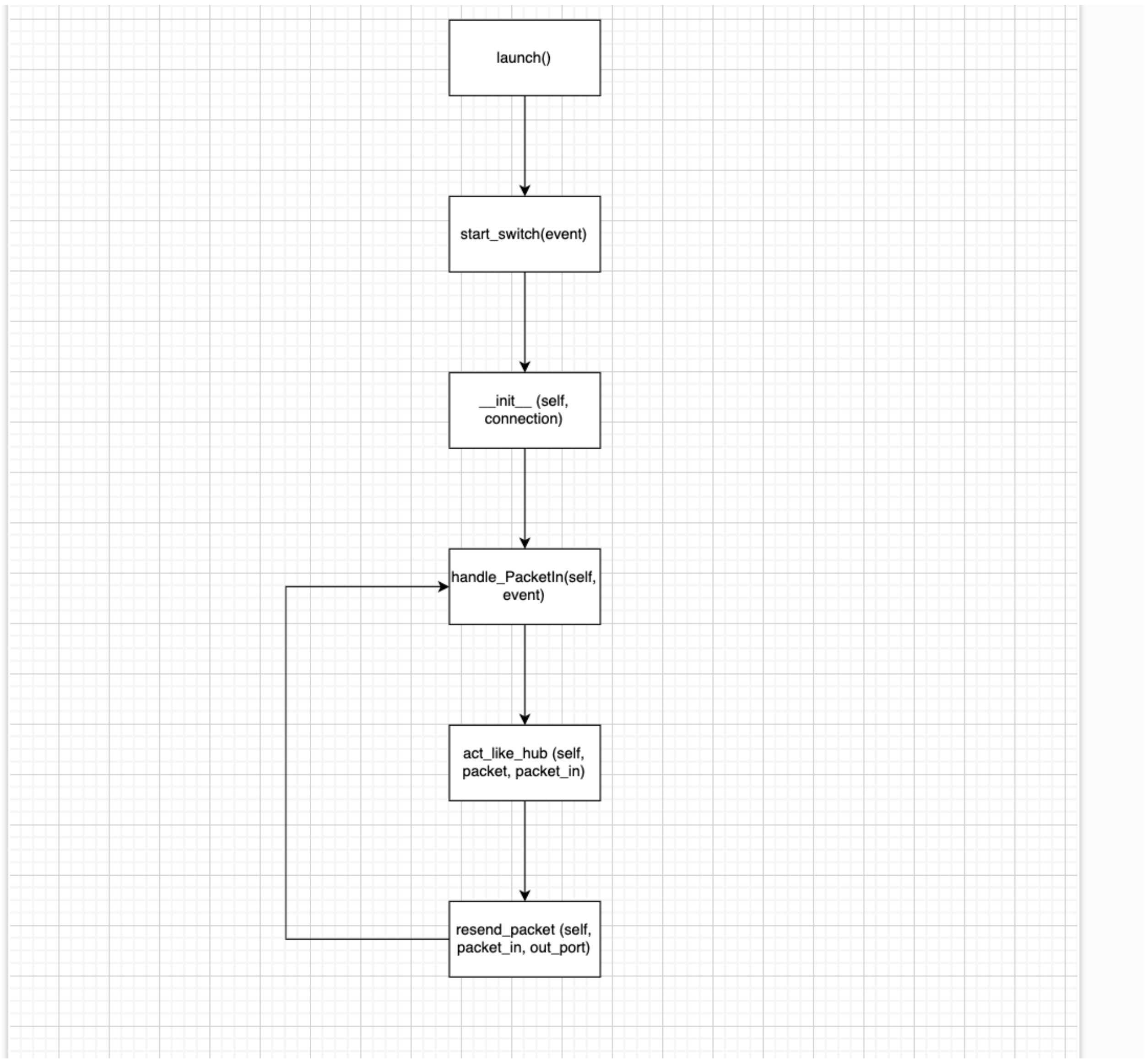
Task II: Study the “of_tutorial” Controller

Q.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)
__init__ (self, connection)
handle_PacketIn(self, event)
act_like_hub (self, packet, packet_in)
resend_packet (self, packet_in, out_port)

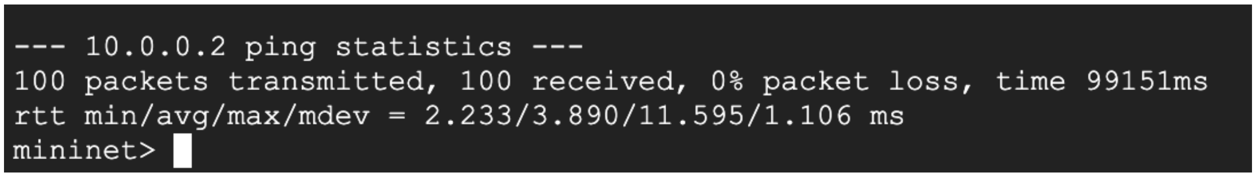
- The above figure shows the packets first enter launch function.
- Then they will enter the start_switch function

- The init function from Tutorial class is called where the connection gets established
- The handle_packetsin function checks id the packet is empty or not
- From there the act_like_hub function is called which behaves like hub and sends packet to all ports besides input port
- Finally, resend_packet is called which instructs switch to resend a packet that it had sent.

- Q.2 Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 h2).
  How long does it take (on average) to ping for each case? What is the difference, and why?

Answer:

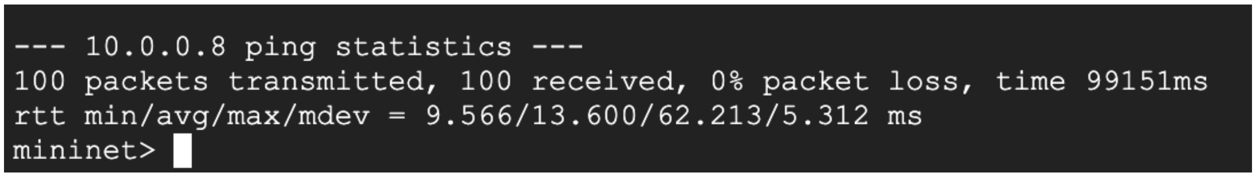
the output of h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 2.233/3.890/11.595/1.106 ms
mininet>
```

The avg time taken is 3.890ms and the total time is 99151ms

the output of h1 ping -c100 h8:

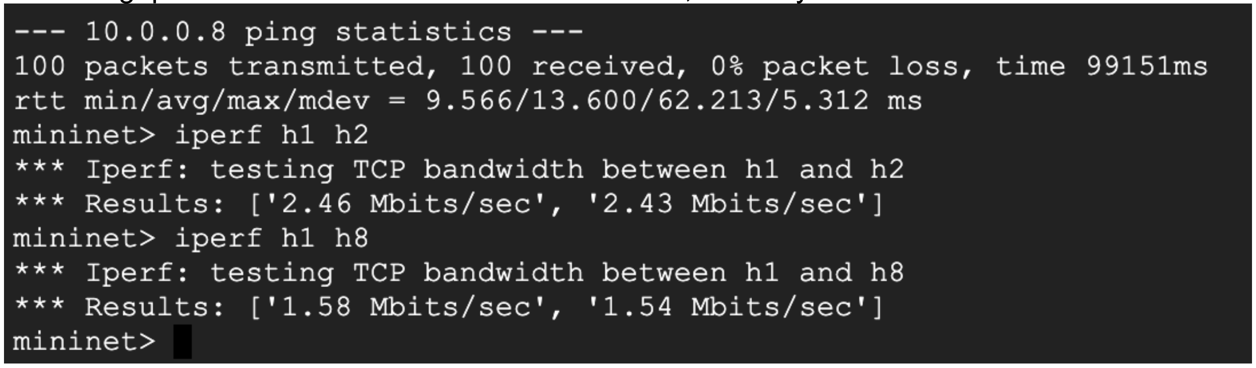


```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 9.566/13.600/62.213/5.312 ms
mininet>
```

The avg time taken is 13.600ms and the total time is 99151ms

As per the topology, nodes h1 and h2 are much close compared to h1 and h8, hence it takes more time for the packet to be transferred from h1 to h8 than h1 to h2

- Q.3 Run “iperf h1 h2” and “iperf h1 h8”. What is “iperf” used for? What is the throughput for each case? What is the difference, and why?



```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 9.566/13.600/62.213/5.312 ms
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['2.46 Mbits/sec', '2.43 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['1.58 Mbits/sec', '1.54 Mbits/sec']
mininet>
```

The throughput between h1 and h2 is 9.77 Mbits/sec whereas the throughput between h1 and h8 is 2.85 Mbits/sec.

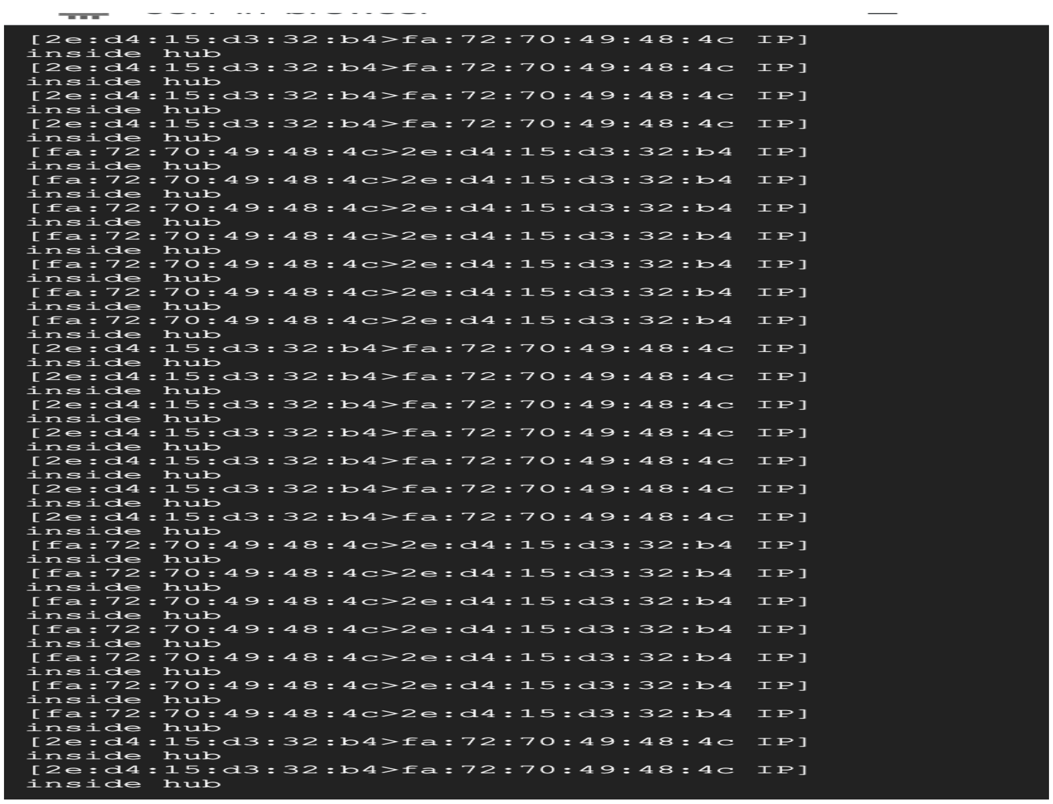
iperf is a widely used open-source tool designed for measuring and analyzing network performance. Its primary purpose is to assess the bandwidth and quality of a network connection by generating TCP data streams between two devices.

The reduction in throughput is due to the proximity of nodes h1 and h2, which only need to traverse a single link and switch (s1). However, reaching h8 requires passing through multiple switches, resulting in a decrease in throughput due to the added complexity of the network path.

- Q.4 Which of the switches observe traffic? Please describe your way for observing such traffic on switches (hint: adding some "print" functions in the "of_tutorial" controller).

Answer:

Currently all the switches will observe traffic because the switches act as hubs. So, the resend_packet will send the packets to all the switches. The behaviour can be verified from the prints put in handle_packetIn and act_as_hub:



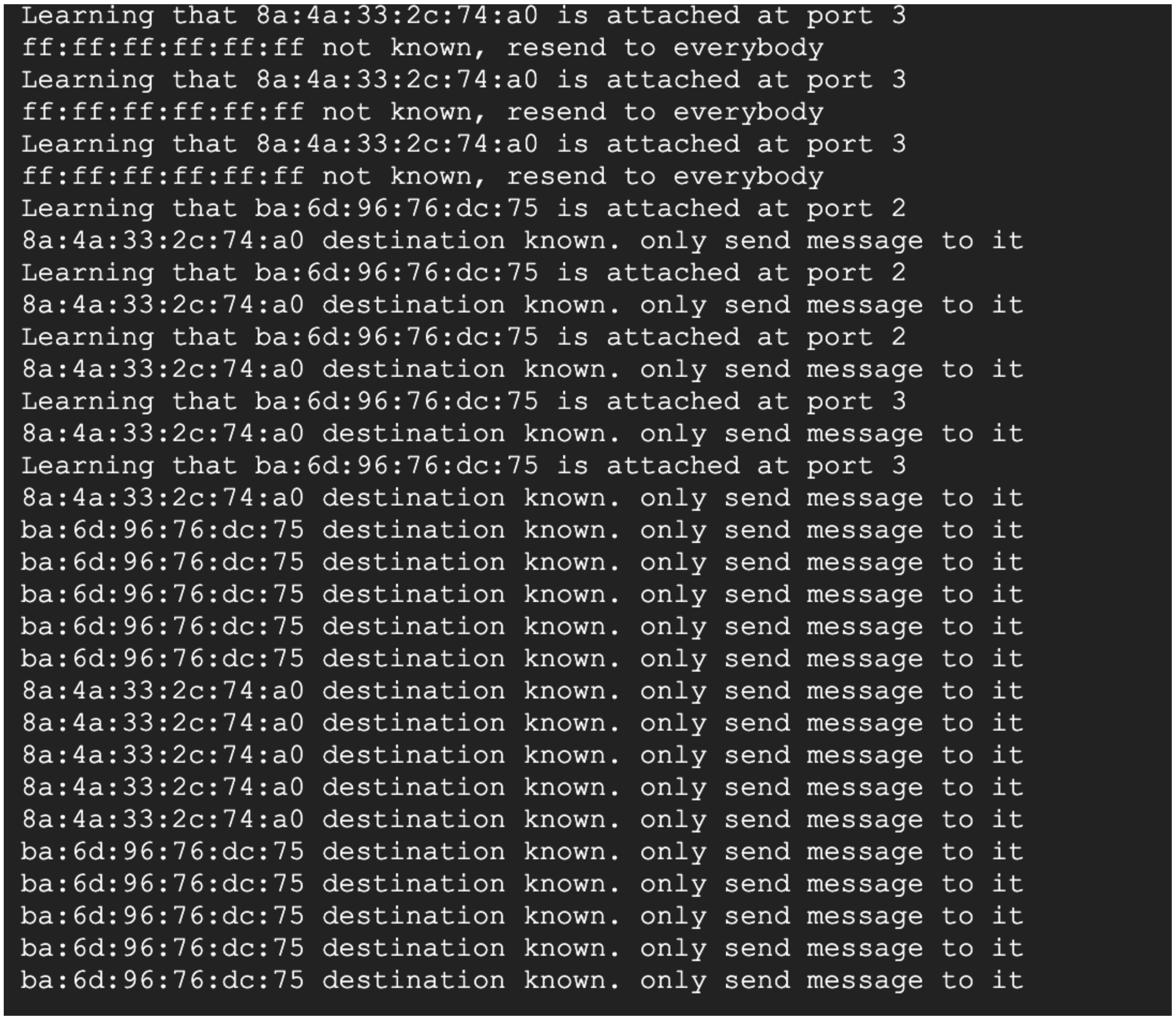
```
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[fa:72:70:49:48:4c>2e:d4:15:d3:32:b4 IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
[2e:d4:15:d3:32:b4>fa:72:70:49:48:4c IP]
inside hub
```

C. Task III: MAC learning controller

- Q.1 Please 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).

Answer:

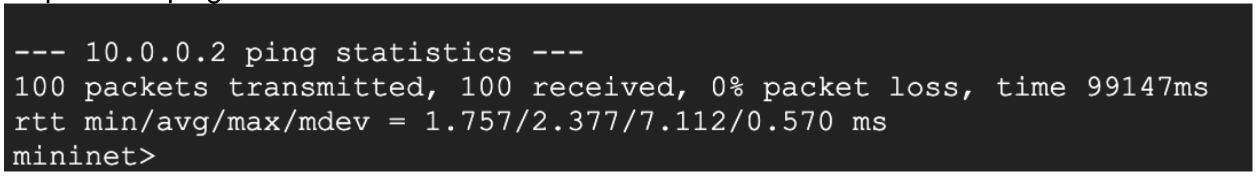
- In the initialization process of the Tutorial, a dictionary mac_to_port is created. When a packet is received, the source MAC address (packet.src) is checked in the dictionary. If it's not present, the function learns the association by adding an entry for the source MAC address and its associated input port (packet_in.in_port).
- Next, the function checks if the destination MAC address (packet.dst) is known. If known, the packet is sent out only to the associated port.
- If the destination MAC address is not known, the function floods the packet to all ports except the input port (of.OFPP_ALL), replicating the broadcast behavior of a switch when the destination is unknown.



```
Learning that 8a:4a:33:2c:74:a0 is attached at port 3
ff:ff:ff:ff:ff:ff not known, resend to everybody
Learning that 8a:4a:33:2c:74:a0 is attached at port 3
ff:ff:ff:ff:ff:ff not known, resend to everybody
Learning that 8a:4a:33:2c:74:a0 is attached at port 3
ff:ff:ff:ff:ff:ff not known, resend to everybody
Learning that ba:6d:96:76:dc:75 is attached at port 2
8a:4a:33:2c:74:a0 destination known. only send message to it
Learning that ba:6d:96:76:dc:75 is attached at port 2
8a:4a:33:2c:74:a0 destination known. only send message to it
Learning that ba:6d:96:76:dc:75 is attached at port 2
8a:4a:33:2c:74:a0 destination known. only send message to it
Learning that ba:6d:96:76:dc:75 is attached at port 3
8a:4a:33:2c:74:a0 destination known. only send message to it
Learning that ba:6d:96:76:dc:75 is attached at port 3
8a:4a:33:2c:74:a0 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
8a:4a:33:2c:74:a0 destination known. only send message to it
8a:4a:33:2c:74:a0 destination known. only send message to it
8a:4a:33:2c:74:a0 destination known. only send message to it
8a:4a:33:2c:74:a0 destination known. only send message to it
8a:4a:33:2c:74:a0 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
ba:6d:96:76:dc:75 destination known. only send message to it
```

- Q.2 (Please disable your output functions, i.e., print, before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2). How long did it take (on average) to ping for each case? Any difference from Task II (the hub case)?

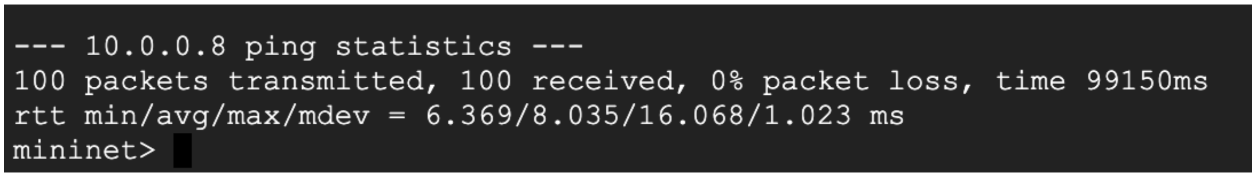
Answer:
output of h1 ping -c100 h2



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99147ms
rtt min/avg/max/mdev = 1.757/2.377/7.112/0.570 ms
mininet>
```

The average time is 2.3777 ms and total time is 99147 ms.

output of h1 ping -c100 h8

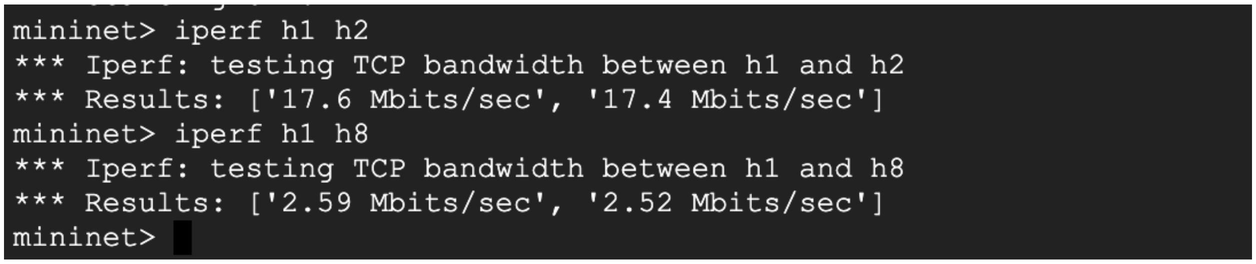


```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99150ms
rtt min/avg/max/mdev = 6.369/8.035/16.068/1.023 ms
mininet>
```

The average time is 8.035 ms and total time is 99150 ms.

There is a small decrease in the average time of packet transfer than that of Task II.

- Q.3 Run "iperf h1 h2" and "iperf h1 h8". What is the throughput for each case? What is the difference from Task II?



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['17.6 Mbits/sec', '17.4 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.59 Mbits/sec', '2.52 Mbits/sec']
mininet>
```

The throughput is 17.6 Mbits /sec for the iperf from h1 to h2.

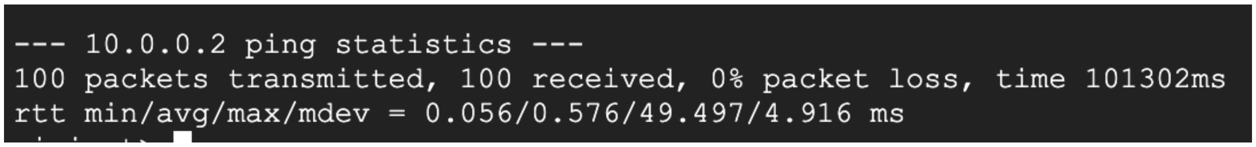
The throughput is 2.59 Mbits /sec for the iperf from h1 to h8.

We can see that there is a substantial increase in the throughput from h1 to h2 which is almost three times of that in task II and there is also slight increase in the throughput from h1 to h8.

D. Task IV: MAC learning controller with OpenFlow rules

- Q.1 Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 h2). How long does it take (on average) to ping for each case? Any difference from Task III (the MAC case without inserting flow rules)?

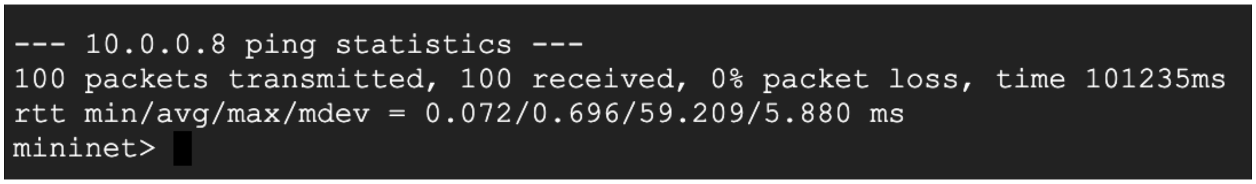
Answer:
output of h1 ping -c100 h2



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101302ms
rtt min/avg/max/mdev = 0.056/0.576/49.497/4.916 ms
```

The average time is 0.576 ms and total time is 101302 ms.

output of h1 ping -c100 h8

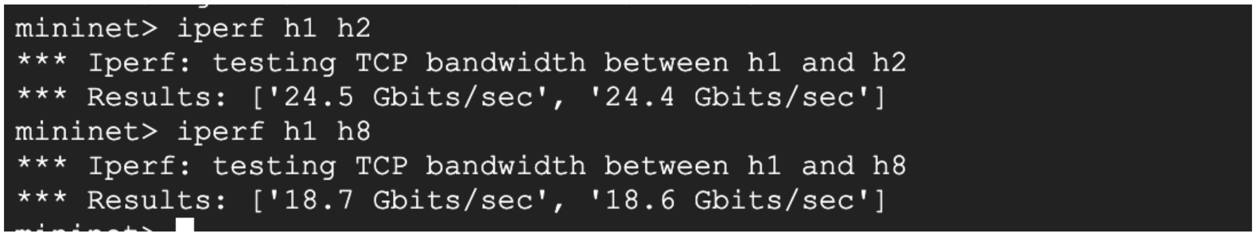


```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101235ms
rtt min/avg/max/mdev = 0.072/0.696/59.209/5.880 ms
mininet> 
```

The average time is 0.696 ms and total time is 101235 ms.

We can see that there is a substantial decrease in the avg time for both cases

- Q.2 Run “iperf h1 h2” and “iperf h1 h8”. What is the throughput for each case? What is the difference from Task III?



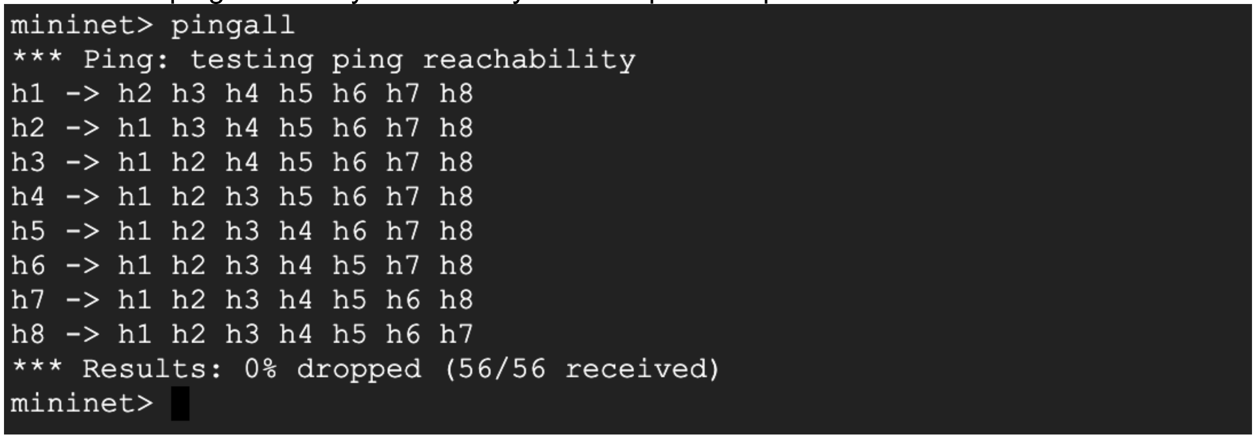
```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['24.5 Gbits/sec', '24.4 Gbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['18.7 Gbits/sec', '18.6 Gbits/sec']
```

We can see that there is a substantial increase in the throughput for both cases

- Q.3 Please explain the above results — why the results become better or worse?

Answer: The ofp_flow_mod messages enable the controller to install flow rules in switches which help the results to become better. These flow rules help in decision making for the controller by defining how packets should be handled without sending sech packet to controller. Switches that have flow rules installed can forward packets in accordance with the pre-established rules, which leads to quicker and more effective packet forwarding.

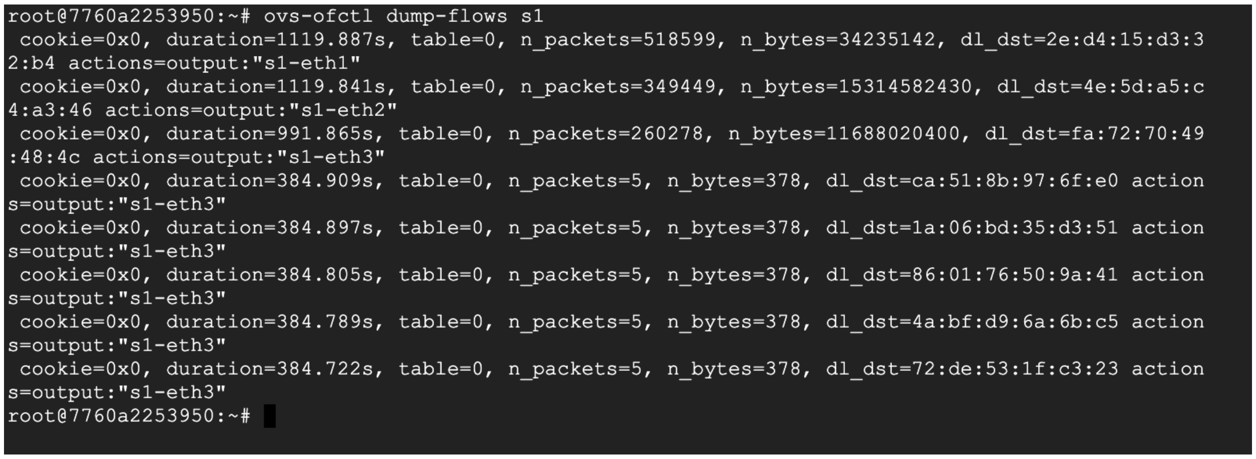
- Q.4 Run pingall to verify connectivity and dump the output.



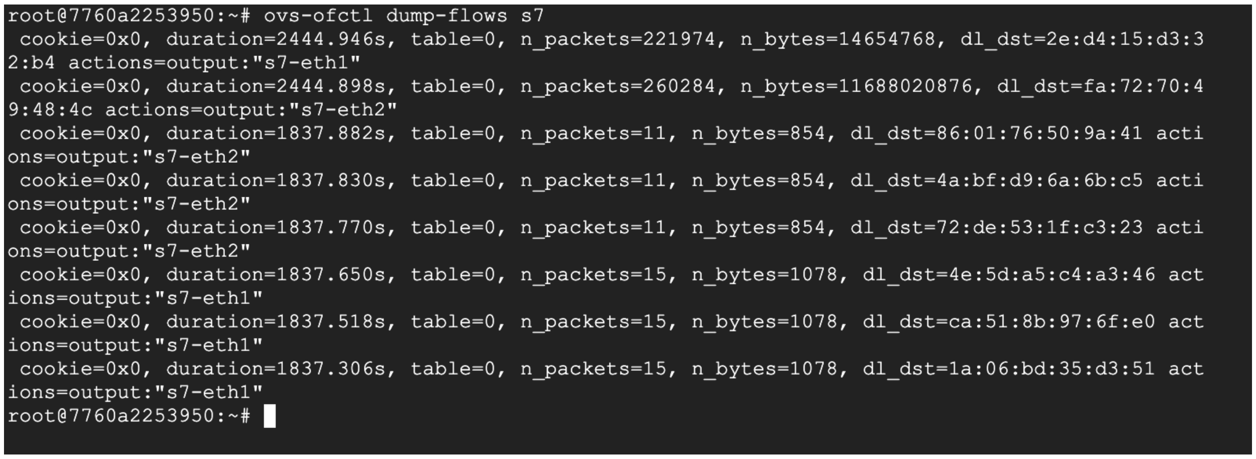
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> 
```

- Q.5 Dump the output of the flow rules using “ovs-ofctl dump-flows” (in your container, not mininet). How many rules are there for each OpenFlow switch, and why? What does each flow entry mean (select one flow entry and explain)?

Answer:



```
root@7760a2253950:~# ovs-ofctl dump-flows s1
 cookie=0x0, duration=1119.887s, table=0, n_packets=518599, n_bytes=34235142, dl_dst=2e:d4:15:d3:3
2:b4 actions=output:"s1-eth1"
 cookie=0x0, duration=1119.841s, table=0, n_packets=349449, n_bytes=15314582430, dl_dst=4e:5d:a5:c
4:a3:46 actions=output:"s1-eth2"
 cookie=0x0, duration=991.865s, table=0, n_packets=260278, n_bytes=11688020400, dl_dst=fa:72:70:49
:48:4c actions=output:"s1-eth3"
 cookie=0x0, duration=384.909s, table=0, n_packets=5, n_bytes=378, dl_dst=ca:51:8b:97:6f:e0 action
s=output:"s1-eth3"
 cookie=0x0, duration=384.897s, table=0, n_packets=5, n_bytes=378, dl_dst=1a:06:bd:35:d3:51 action
s=output:"s1-eth3"
 cookie=0x0, duration=384.805s, table=0, n_packets=5, n_bytes=378, dl_dst=86:01:76:50:9a:41 action
s=output:"s1-eth3"
 cookie=0x0, duration=384.789s, table=0, n_packets=5, n_bytes=378, dl_dst=4a:bf:d9:6a:6b:c5 action
s=output:"s1-eth3"
 cookie=0x0, duration=384.722s, table=0, n_packets=5, n_bytes=378, dl_dst=72:de:53:1f:c3:23 action
s=output:"s1-eth3"
root@7760a2253950:~# 
```



```
root@7760a2253950:~# ovs-ofctl dump-flows s7
 cookie=0x0, duration=2444.946s, table=0, n_packets=221974, n_bytes=14654768, dl_dst=2e:d4:15:d3:3
2:b4 actions=output:"s7-eth1"
 cookie=0x0, duration=2444.898s, table=0, n_packets=260284, n_bytes=11688020876, dl_dst=fa:72:70:4
9:48:4c actions=output:"s7-eth2"
 cookie=0x0, duration=1837.882s, table=0, n_packets=11, n_bytes=854, dl_dst=86:01:76:50:9a:41 acti
ons=output:"s7-eth2"
 cookie=0x0, duration=1837.830s, table=0, n_packets=11, n_bytes=854, dl_dst=4a:bf:d9:6a:6b:c5 acti
ons=output:"s7-eth2"
 cookie=0x0, duration=1837.770s, table=0, n_packets=11, n_bytes=854, dl_dst=72:de:53:1f:c3:23 acti
ons=output:"s7-eth2"
 cookie=0x0, duration=1837.650s, table=0, n_packets=15, n_bytes=1078, dl_dst=4e:5d:a5:c4:a3:46 act
ions=output:"s7-eth1"
 cookie=0x0, duration=1837.518s, table=0, n_packets=15, n_bytes=1078, dl_dst=ca:51:8b:97:6f:e0 act
ions=output:"s7-eth1"
 cookie=0x0, duration=1837.306s, table=0, n_packets=15, n_bytes=1078, dl_dst=1a:06:bd:35:d3:51 act
ions=output:"s7-eth1"
root@7760a2253950:~# 
```

The above output shows 8 flow rules which refer to every host (h1-h8) available and the flow rules to reach them.

For Example,
cookie=0x0, duration=384.722s, table=0, n_packets=5, n_bytes=378, dl_dst=72:de:53:1f:c3:23 actions=output:"s1-eth3"

cookie: The cookie value associated with the flow entry.

duration: The duration the flow entry has been active, measured in seconds. Here the flow entry is active for 384.722 secs table: The table in which the flow entry is installed. Currently the table is 0.

n_packets: The number of packets that have matched this flow entry. In this example, the flow has matched 5 packets.

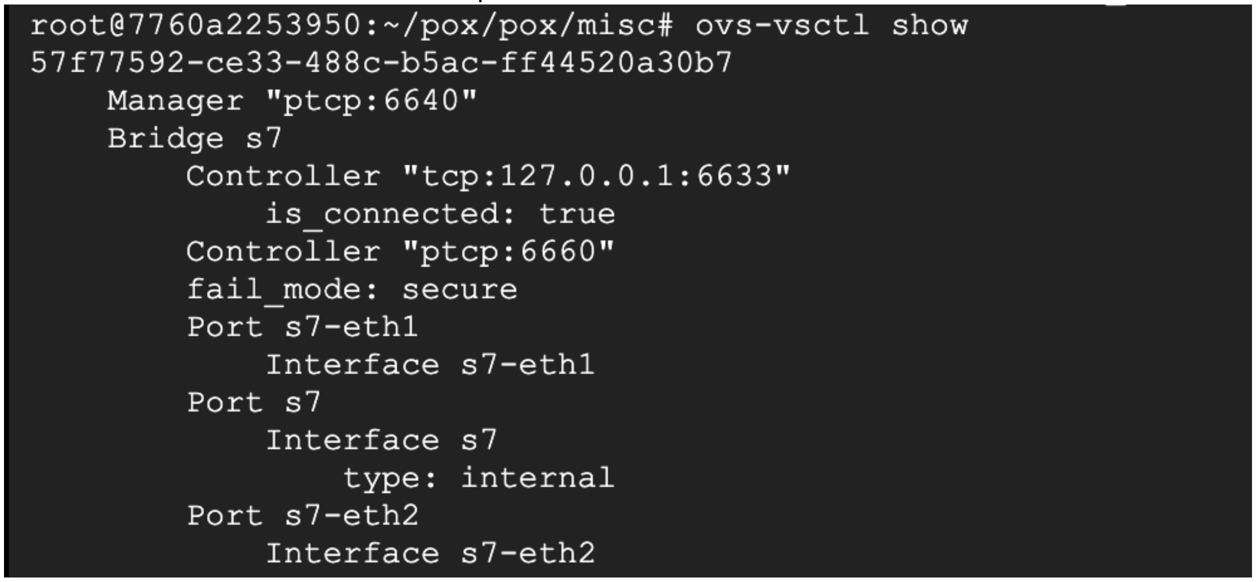
n_bytes=378: The total number of bytes that have matched this flow entry. In this example, the flow has matched 378 bytes.

dl_dst: The match criteria for the flow entry. This flow entry matches packets with a destination MAC address of 72:de:53:1f:c3:23.

actions: The action associated with the flow entry. In this case, packets matching the flow criteria are sent to the output port named "s1-eth3."

Task V: Layer-3 routing:

We can use ovs-vsctl show to find the Dpid for switch 7:



```
root@7760a2253950:~/pox/pox/misc# ovs-vsctl show
57f77592-ce33-488c-b5ac-ff44520a30b7
    Manager "ptcp:6640"
    Bridge s7
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        Controller "ptcp:6660"
        fail_mode: secure
        Port s7-eth1
            Interface s7-eth1
        Port s7
            Interface s7
                type: internal
        Port s7-eth2
            Interface s7-eth2
```

We will use the following commands to create the IP rules for switch 7:
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.5,actions=output:"s7-eth1"
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.6,actions=output: "s7-eth2"
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.7,actions=output: "s7-eth1"
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.8,actions=output: "s7-eth2"

Commands for IP-matching h2 with h5, h6, h7, h8 through switch 7
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.2,nw_dst=10.0.0.5,actions=output: "s7-eth2"
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.2,nw_dst=10.0.0.6,actions=output: "s7-eth1"

```
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.2,nw_dst=10.0.0.7,actions=output: ”s7-eth1”
ovs-ofctl add-flow s7 priority=100,ip,nw_src=10.0.0.2,nw_dst=10.0.0.8,actions=output: ”s7-eth2”
```

This will install IP rules for switch 7 and the traffic will flow through the IP rules now. We can verify that by running the dump-flows command.

The above commands uses the source IP and destination IP address and sets the flow based on it on the specified output port eth1 eth2. This commands has to be run after starting the mininet cluster and the controller on a different shell.