Dimple Singh
B00957272
dsingh27@binghamton.edu

Mini Project 3 Question Answers

TASK 2

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?

Answer 1: The Controller is started when we execute the below-mentioned command:
python3 pox.py log.level --DEBUG misc.of_tutorial
After execution of the same in the pox folder the pox.py file is executed and it takes three arguments which are:-
1) log.level – to check and pass the level of logs we can access
2) DEBUG – to run the code in debug mode
3) misc.of_tutorial – takes log data as input and receives it through port 0.0.0.0:6633. It then parses and separates the logs that are specifically related to Mininet. Finally, the program executes various functions mentioned below using the extracted logs.

1. The entry point of of_tutorial file is launch() function which is executed first when we start the component. In launch(), it connects to a switch and start_switch is called which creates an object Tutorial for the switch.

```
def launch ():
    """
    Starts the component
    """
    def start_switch (event):
        log.debug("Controlling %s" % (event.connection,))
        Tutorial(event.connection)
    core.openflow.addListenerByName("ConnectionUp", start_switch)
```

2)After the creation of object of Tutorial the __initi__() function is called it sets up event listeners and maintains a table to keep track of which ethernet address is on MAC addresses with their switch ports.

```
class Tutorial (object):
    """
    A Tutorial object is created for each switch that connects.
    A Connection object for that switch is passed to the
__init__function.
    """
    def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection
    # This binds our PacketIn event listener
    connection.addListeners(self)
    # Use this table to keep track of which ethernet address is on
    # which switch port (keys are MACs, values are ports).
    self.mac_to_port = {}
```

3)After packet is received from the switch, the _handle_PacketIn() function is called:

```
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("Source MAC: ", packet.src)
            print("Switch: ",packet_in.in_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)
```

4) The _handle_PacketIn() is responsible for parsing incoming packets and subsequently invoking act_like_hub() or act_like_switch() functions based on the intended behavior.

5) The act_like_hub() function resends packets to all ports besides input port and implements hub-like behaviour.

```
        def act_like_hub (self, packet, packet_in):
        """
        Implement hub-like behavior -- send all packets to all ports besides
        the input port.
        """
        # We want to output to all ports -- we do that using the special
        # OFPP_ALL port as the output port.  (We could have also used
        # OFPP_FLOOD.)
        self.resend_packet(packet_in, of.OFPP_ALL)
        # Note that if we didn't get a valid buffer_id, a slightly better
        # implementation would check that we got the full data before
        # sending it (len(packet_in.data) should be == packet_in.total_len)).
```

6) The act_like_switch() function implements switch-like behaviour. It learns the port for the source MAC address and then resends the packet to the learned port or floods it to all ports besides the input port.

```
        def act_like_switch (self, packet, packet_in):
            """
            Implement switch-like behavior.
            """
            """ # DELETE THIS LINE TO START WORKING ON THIS (AND
        THE ONE BELOW!) #
            # Here's some psuedocode to start you off implementing a learning
            # switch.  You'll need to rewrite it as real Python code.
            # Learn the port for the source MAC
            self.mac_to_port ... <add or update entry>
            if the port associated with the destination MAC of the packet
        is known:
            # Send packet out the associated port
            self.resend_packet(packet_in, ...)
            # Once you have the above working, try pushing a flow entry
            # instead of resending the packet (comment out the above and
            # uncomment and complete the below.)
            log.debug("Installing flow...")
            # Maybe the log statement should have source/destination/port?
            #msg = of.ofp_flow_mod()
            #
            ## Set fields to match received packet
            #msg.match = of.ofp_match.from_packet(packet)
            #
            #< Set other fields of flow_mod (timeouts? buffer_id?) >
```

```
                #
                #< Add an output action, and send -- similar to resend_packet() >
                else:
                        # Flood the packet out everything but the input port
                        # This part looks familiar, right?
                        self.resend_packet(packet_in, of.OFPP_ALL)
```

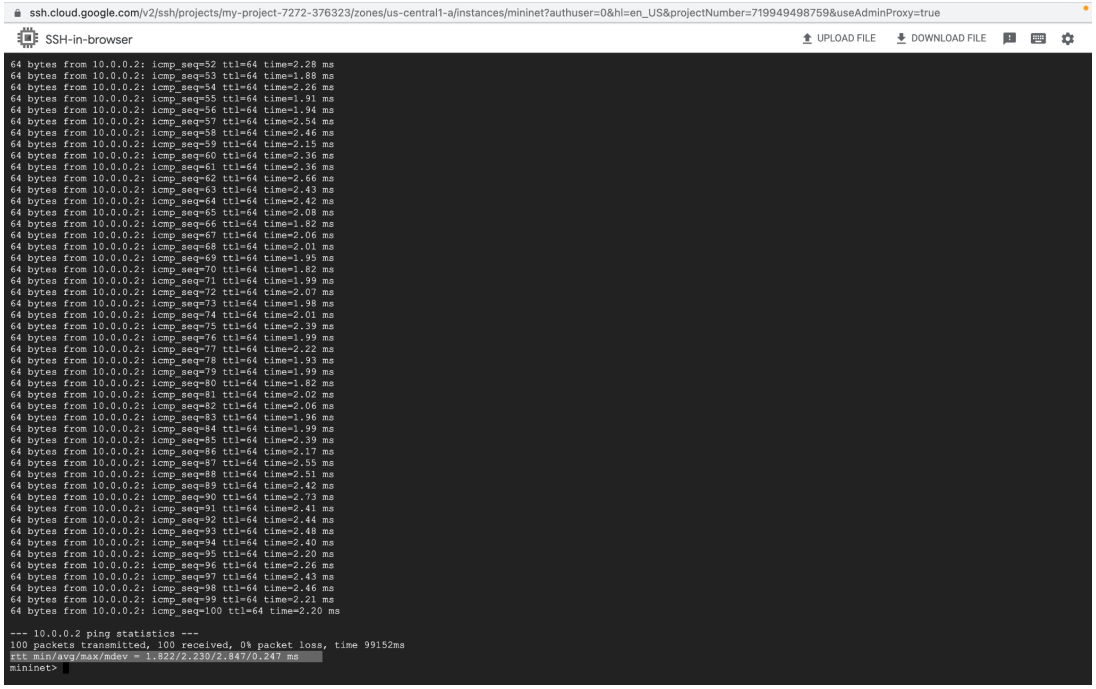
7) Lastly, The resend_packet() function is called which implements hub-like behavior by sending packet back to the switch with a specified output port or implements switch-like behavior by resending a learned packet.

```
        def resend_packet (self, packet_in, out_port):
                """
                Instructs the switch to resend a packet that it had sent to us.
                "packet_in" is the ofp_packet_in object the switch had sent to the
                controller due to a table-miss.
                """
                msg = of.ofp_packet_out()
                msg.data = packet_in
                # Add an action to send to the specified port
                action = of.ofp_action_output(port = out_port)
                msg.actions.append(action)
                # Send message to switch
                self.connection.send(msg)
```

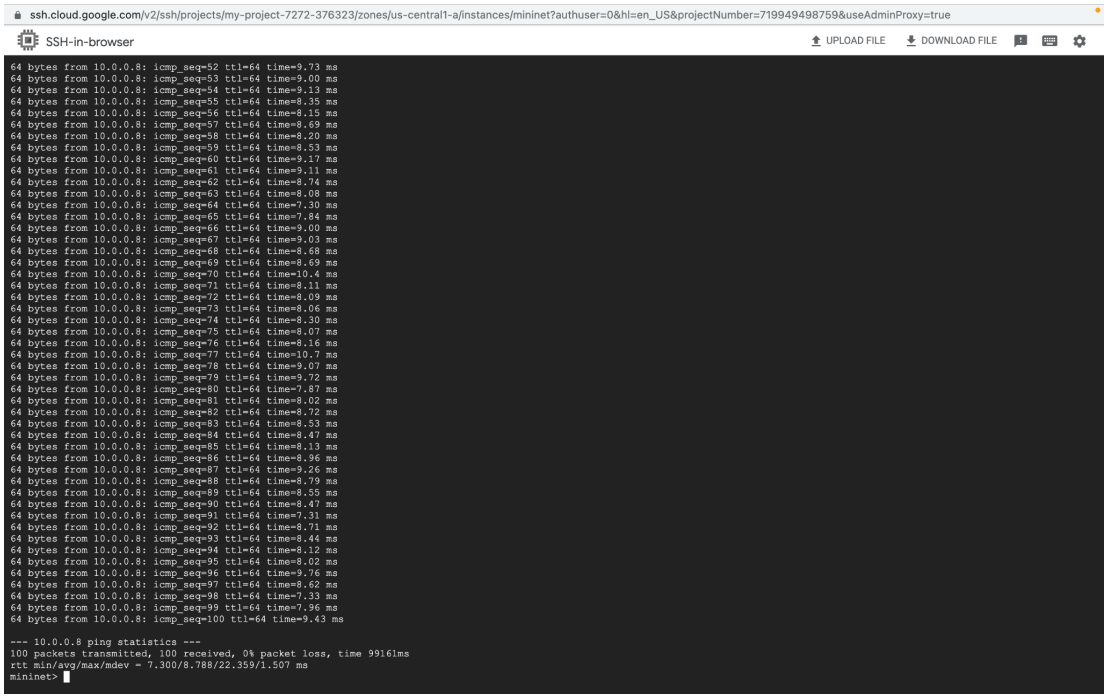
Q.2 Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2). How long does it take (on average) to ping for each case? What is the difference, and why?

Answer 2:

mininet> h1 ping -c 100 h2
result is below:



Average Time : 2.230ms
mininet> h1 ping -c 100 h8



Average Time : 8.788ms

The difference in average time in both cases is due to various reasons which can affect the time taken for ping packets to travel from one host to another namely :

1) Physical distance between the hosts
2) Network Topology
3) Routing path
4) Network congestion
5) Quality of network links

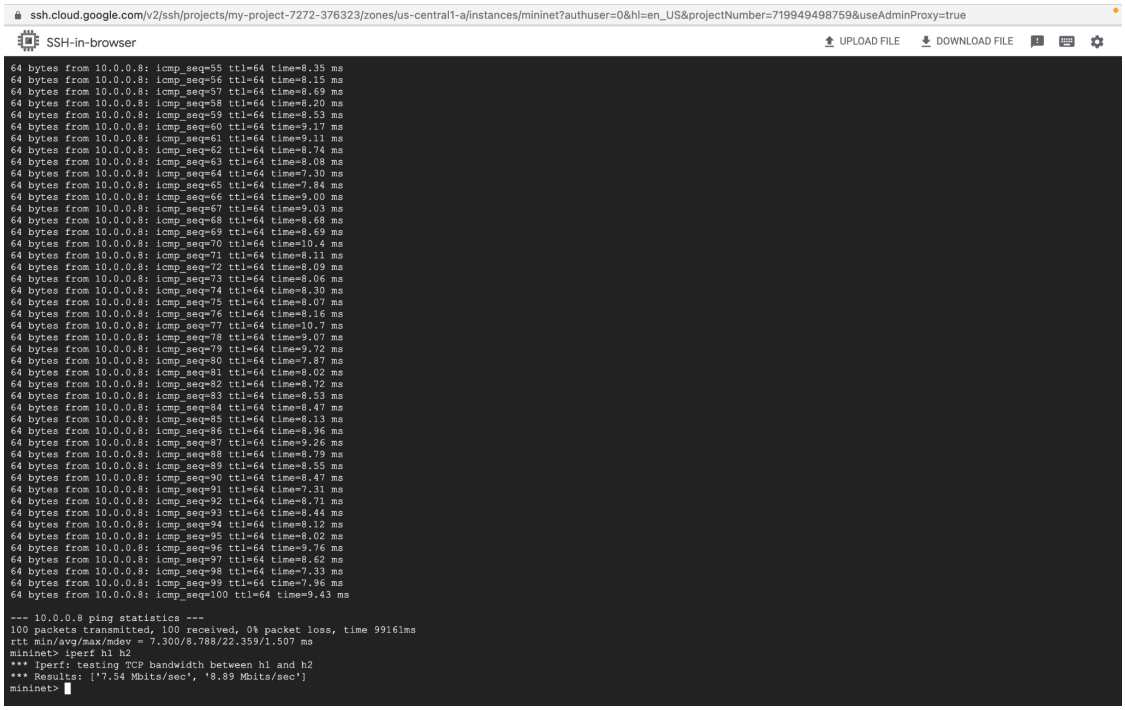
Note: Ping times can vary over time and under different network conditions.

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?

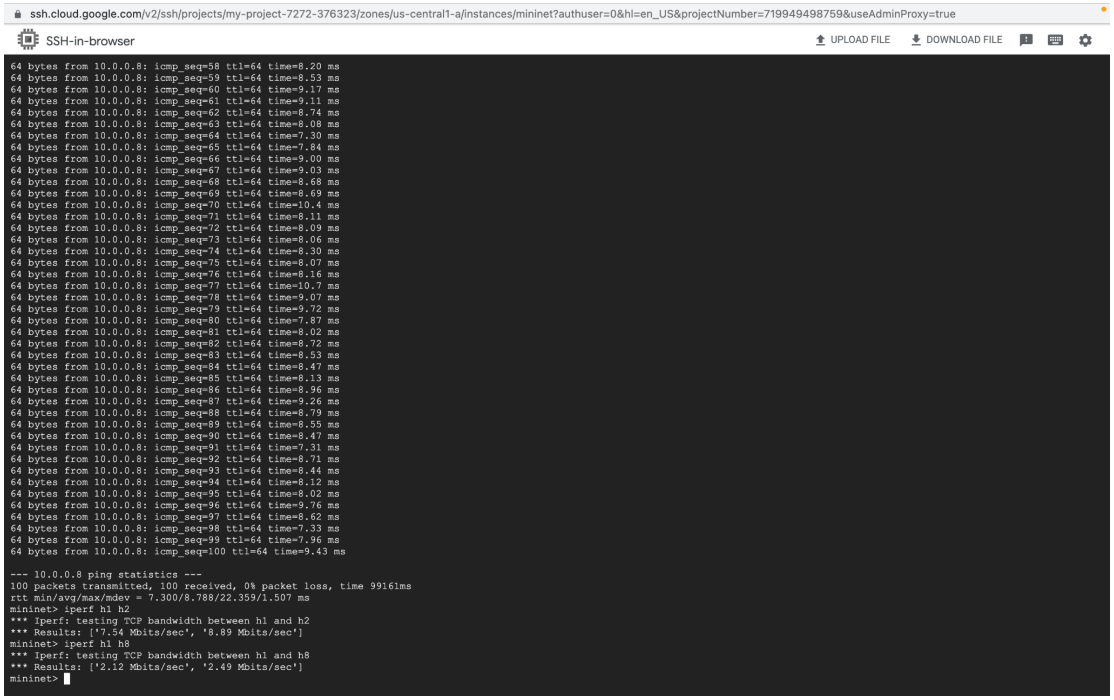
Answer 3:

mininet> iperf h1 h2

Result - 7.54 Mbits/sec , 8.89 Mbits/sec



mininet> iperf h1 h8
Result - 2.12 Mbits/sec , 2.49Mbits/sec
Result below:



The difference in throughput between the two cases (h1 to h2) and (h1 to h8) could be due to various reasons namely:

1. Network congestion
2. Host processing capabilities
3. Network configuration

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 4:
Results / Observations:
1) More traffic flowing from switches 2 and 3 moreover on switch 3 is greater than switch 2.
2) This is probably, when the flooding of port happens and then every time the for the connection is established it flood around the switches and port and the traffic is observed on Switch 7 which is connecting and main switch between the connection topology of binary tree.

Code changes added in bold :

```
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("Source MAC: ", packet.src)
print("Switch: ",packet_in.in_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)
```

ssh.cloud.google.com/v2/ssh/projects/my-project-7272-376323/zones/us-central1-a/instances/mininet?authuser=0&hl=en_US&projectNumber=719949498759&useAdminProxy=true

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
      # Once you have the above working, try pushing a flow entry
      # instead of resending the packet (comment out the above and
      # uncomment and complete the below.)

      log.debug("Installing flow...")
      # Maybe the log statement should have source/destination/port?

      #msg = of.ofp_flow_mod()
      #
      ## Set fields to match received packet
      #msg.match = of.ofp_match.from_packet(packet)
      #
      #< Set other fields of flow_mod (timeouts? buffer_id?) >
      #
      #< Add an output action, and send -- similar to resend_packet() >

    else:
      # Flood the packet out everything but the input port
      # This part looks familiar, right?
      self.resend_packet(packet_in, of.OFPP_ALL)

    """ # DELETE THIS LINE TO START WORKING ON THIS #


  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("Switch: ",packet_in.in_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)


def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Tutorial(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
root@7fdfe7fcd559:~/pox/pox/misc#
```

ssh.cloud.google.com/v2/ssh/projects/my-project-7272-376323/zones/us-central1-a/instances/mininet?authuser=0&hl=en_US&projectNumber=719949498759&useAdminProxy=true

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
INFO:openflow.of_01:[00-00-00-00-00-02 7] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-02 7]
Switch:  1
Switch:  1
Switch:  1
Switch:  3
Switch:  3
Switch:  3
Switch:  2
Switch:  3
Switch:  2
Switch:  2
Switch:  3
Switch:  3
Switch:  3
Switch:  3
Switch:  1
Switch:  1
Switch:  1
Switch:  3
Switch:  3
Switch:  3
Switch:  3
Switch:  2
Switch:  2
Switch:  2
Switch:  3
Switch:  3
Switch:  3
Switch:  3
Switch:  1
Switch:  1
Switch:  3
Switch:  1
Switch:  3
Switch:  3
Switch:  2
Switch:  2
Switch:  3
Switch:  2
Switch:  3
Switch:  3
Switch:  3
Switch:  1
Switch:  1
Switch:  1
Switch:  3
Switch:  3
Switch:  3
Switch:  3
Switch:  2
Switch:  2
Switch:  2
```

TASK 3

Q.5 Please describe how the above code works, such as how the "MAC toPort" map is established. You could use a 'ping' example to describe the establishment process (e.g.,h1 ping h2).

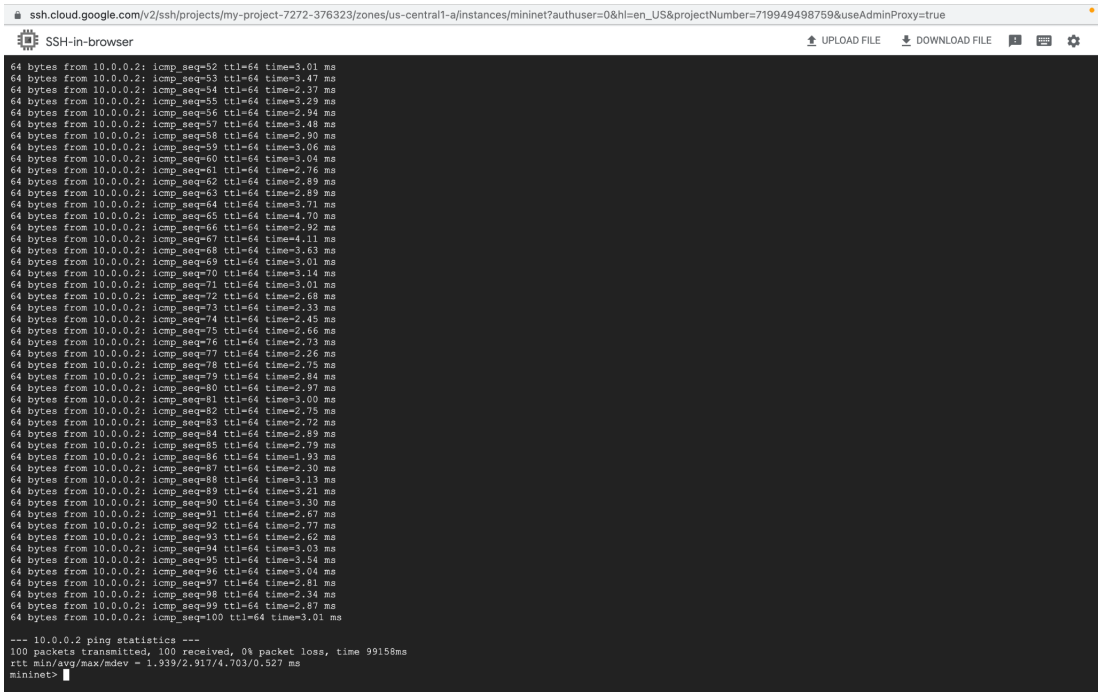
Answer 5:

```
def act_like_switch (self, packet, packet_in):
# Learn the port for the source MAC
# print "Src: ",str(packet.src),":", packet_in.in_port,"Dst:", str(packet.dst) if packet.src not in self.mac_to_port:

print "Learning that " + str(packet.src) + " is attached at port " + str(packet_in.in_port) self.mac_to_port[packet.src] =
packet_in.in_port

# if the port associated with the destination MAC of the packet is known: if packet.dst in self.mac_to_port:

# Send packet out the associated port
print str(packet.dst) + " destination known. only send message to it" self.resend_acket(packet_in,
self.mac_to_port[packet.dst])

else:
# Flood the packet out everything but the input port
# This part looks familiar, right?
print str(packet.dst) + " not known, resend to everybody" self.resend_packet(packet_in, of.OFPP_ALL)
```

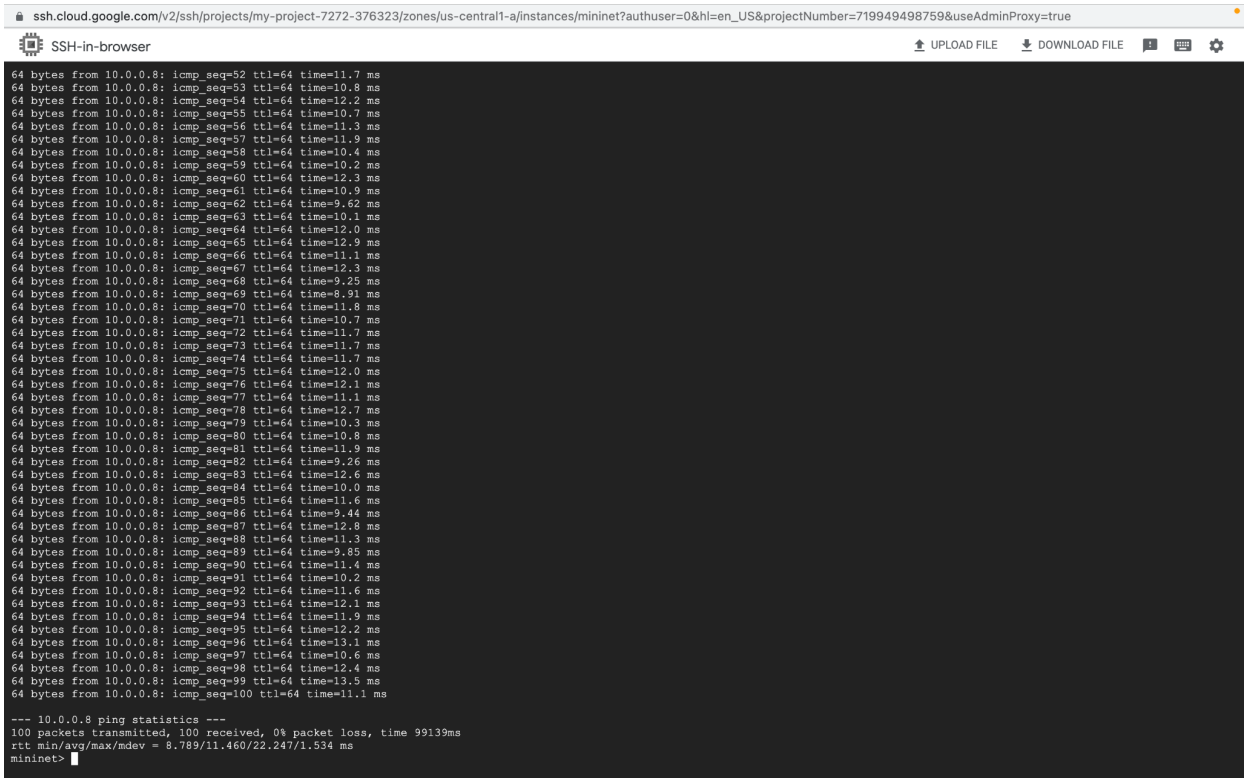
Answer 5: The above code is an implementation of a basic switching hub using the OpenFlow protocol.
The "MAC to Port" map is established using the dictionary "self.mac_to_port". Each time a packet is received, the source MAC address is extracted from the packet and checked to see if it is already in the dictionary. If not, the current port number of the incoming packet is recorded in the dictionary against the source MAC address.
When the destination MAC address of a packet is received, the code checks if it is already known in the "MAC to Port" map. If yes, the packet is forwarded out of the associated port. If not, the packet is flooded to all connected ports, except for the port where the packet was received.
The code first learns the mapping between MAC addresses and switch ports, and then forwards packets based on this mapping. If the destination MAC address is unknown, the packet is flooded to all ports except the incoming port.

Q.6 (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 6:



mininet> h1 ping -c 100 h2
Result Average Time task 3 = 2.917ms
Avg Time task 2 = 2.230ms
mininet> h1 ping h8



Result Average Time task 3= 11.460ms
Avg Time task 2= 8.788ms

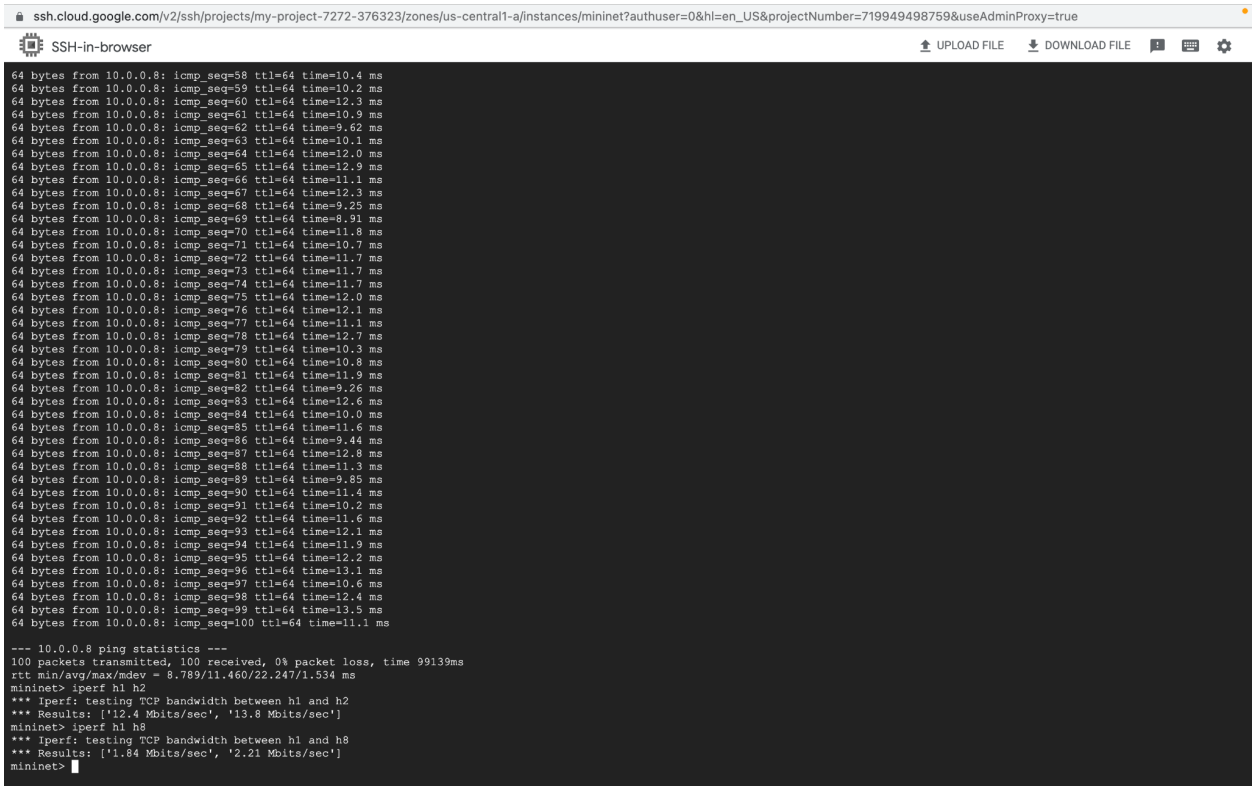
In Task 2:
The code is an implementation of a simple hub that forwards all packets received to all ports except the incoming port. The hub can be modified to act like an L2 learning switch. The learning switch learns the source port of each MAC address and forwards packets with unknown destination addresses to all ports except the incoming port.

In Task 3:
The code implements a basic learning switch that learns the port associated with the source MAC address of packets and forwards packets with known destination MAC addresses to the associated port. If the destination MAC address is unknown, the switch broadcasts the packet to all ports except the incoming port.

It's taking more time in task 3 because you are required to implement a learning switch, which involves more complex logic than the simple hub behavior implemented in task 2. In a learning switch, the switch needs to learn which MAC addresses are connected to which switch ports, so it can forward packets intelligently rather than flooding them to all ports like a hub. This involves maintaining a table of MAC-to-port mappings and updating it as packets are received. Additionally, when a packet is received with a destination MAC address for which the switch has a mapping, the switch needs to forward the packet to the correct output port, rather than flooding it to all ports like a hub. This requires more complex processing and decision-making logic than the simple hub behavior of task 2.

Q.7 Run “iperf h1 h2” and “iperf h1 h8”. What is the throughput for each case? What is the difference from Task II?

Answer 7:



Task 3
mininet> iperf h1 h2
Result = 12.4 Mbits/sec, 13.8 Mbits/sec

Task 2
Result = 7.54 Mbits/sec , 8.89 Mbits/sec

Task 3
mininet> iperf h1 h8
Result = 1.84 Mbits/sec, 2.21 Mbits/sec

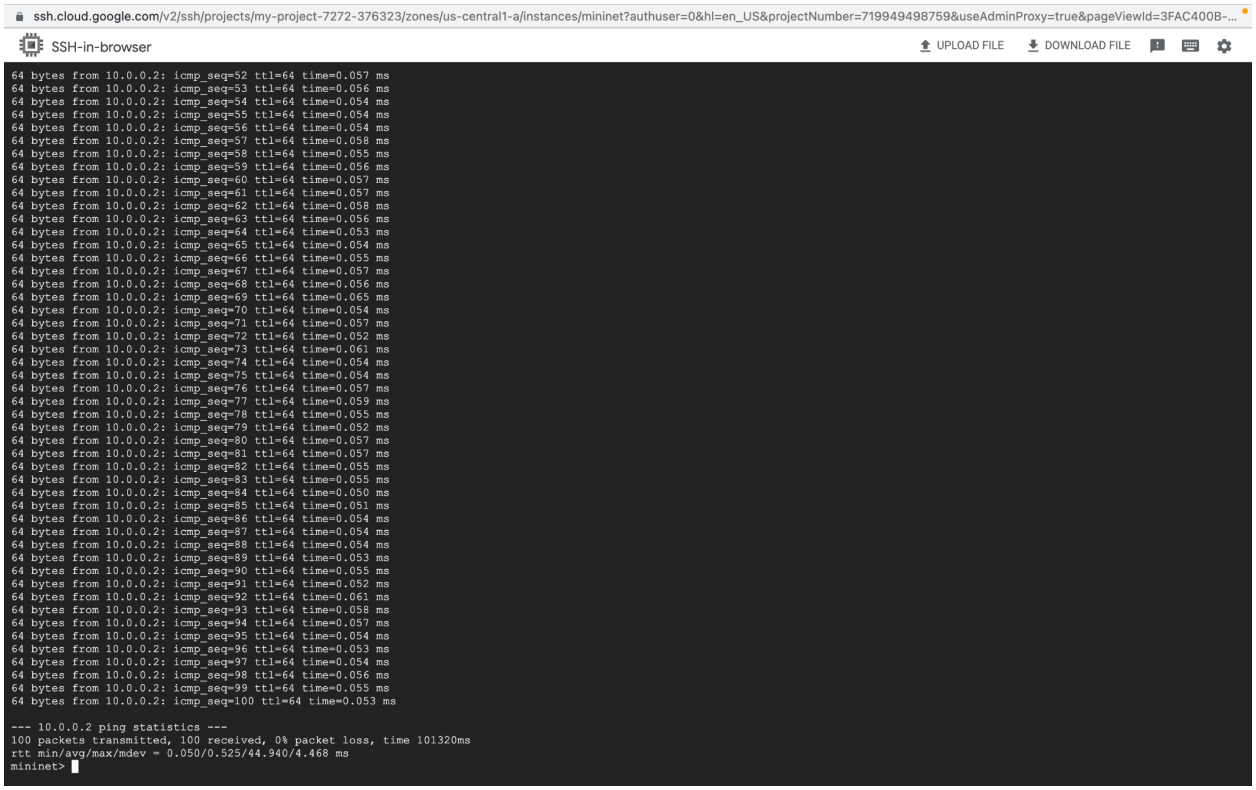
Task 2
Result = 2.12 Mbits/sec , 2.49Mbits/sec

The difference in the results between task 2 and task 3 can be due to various factors, such as the network topology, traffic routing, and network congestion. Task 3 involves testing the network traffic between two hosts connected through multiple switches, whereas task 2 involves testing traffic between two hosts connected directly to the same switch. The additional network elements involved in task 3 can contribute to increased latency, packet loss, and congestion, which can result in lower network throughput.

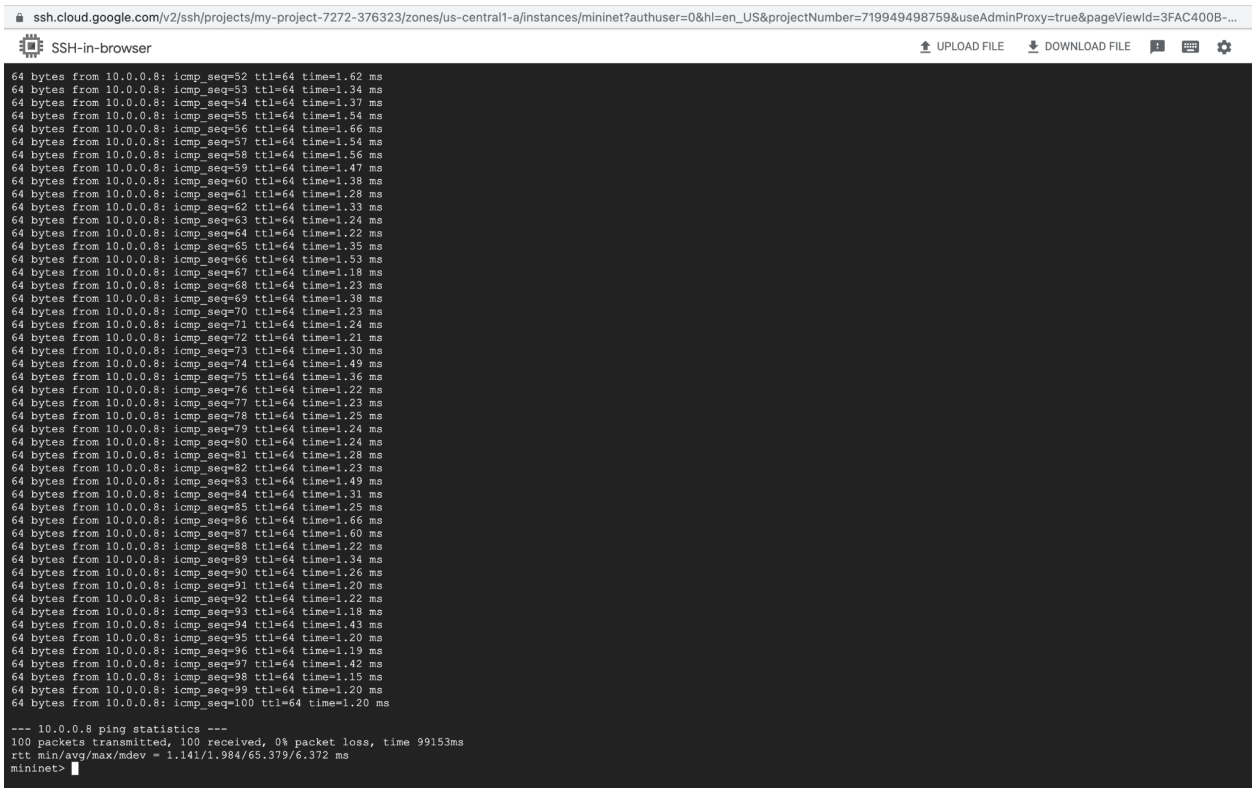
TASK 4

Q.8 Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2). 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 8:
Task 4
mininet> h1 ping -c 100 h2



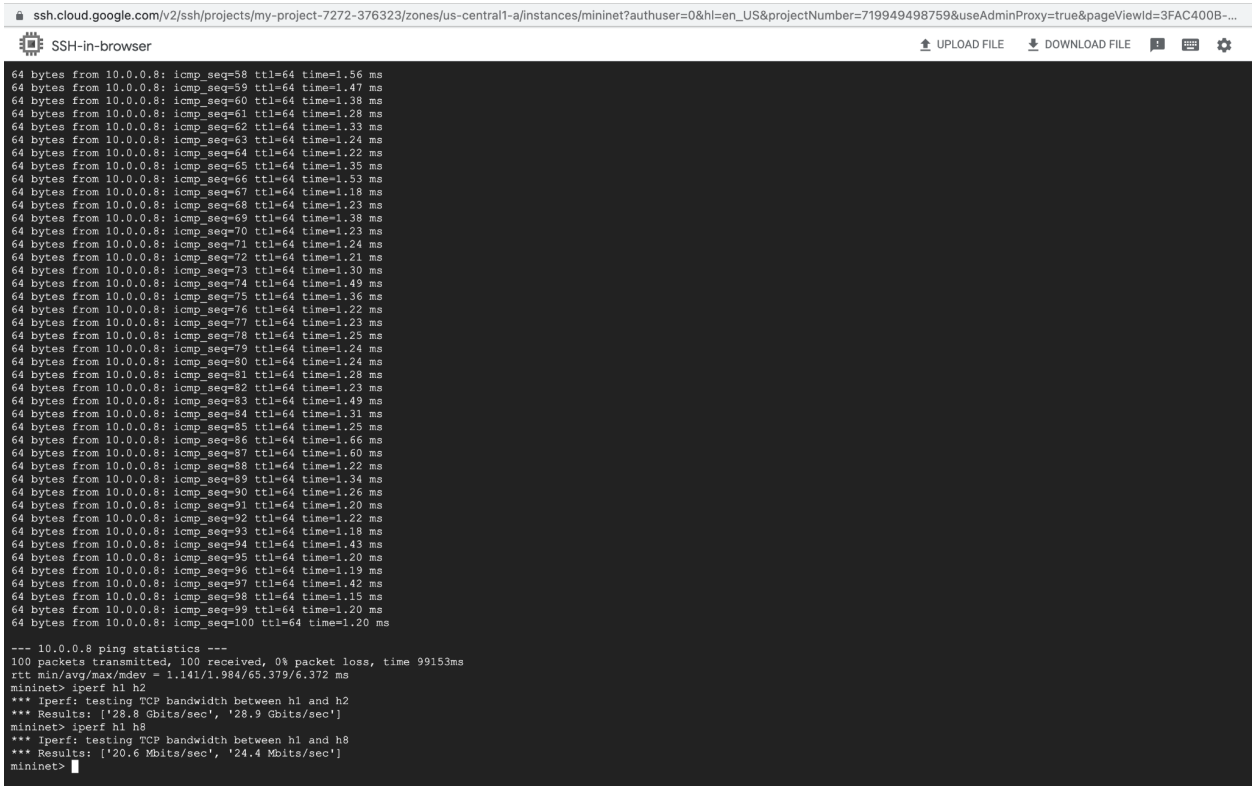
Result Average Time task 4 = 0.525ms
Result Average Time task 3 = 2.917ms
mininet> h1 ping -c 100 h8



Result Average Time task 4= 1.984ms
Result Average Time task 3= 11.460ms
The Average Time taken has decreased a lot in task 4 as compared to task 3.

Q.9 Run “iperf h1 h2” and “iperf h1 h8”. What is the throughput for each case? What is the difference from Task III?

Answer 9:
mininet> iperf h1 h2
mininet> iperf h1 h8



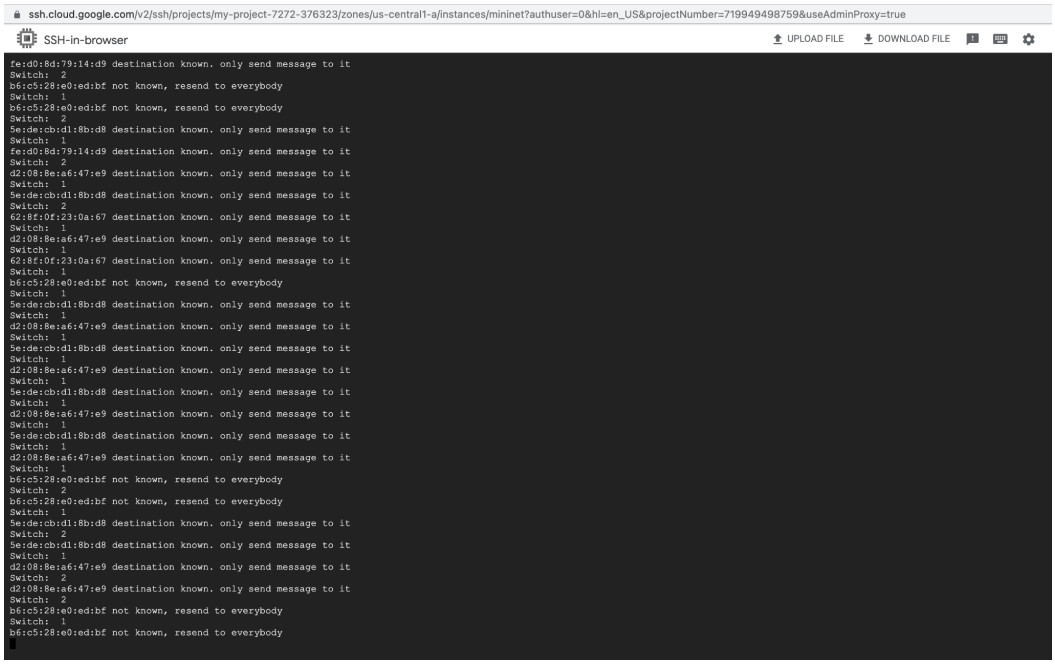
Result iperf h1 h2 task 4 = 28.8 Gbits/sec, 28.9 Gbits/sec
Result iperf h1 h2 task 3 = 12.4 Mbits/sec, 13.8 Mbits/sec
Result iperf h1 h8 task 4 = 20.6 Mbits/sec, 24.4 Mbits/sec
Result iperf h1 h8 task 3 = 1.84 Mbits/sec, 2.21 Mbits/sec
The throughput has increased in the case of task 4.

Q.10 Please explain the above results — why the results become better or worse?

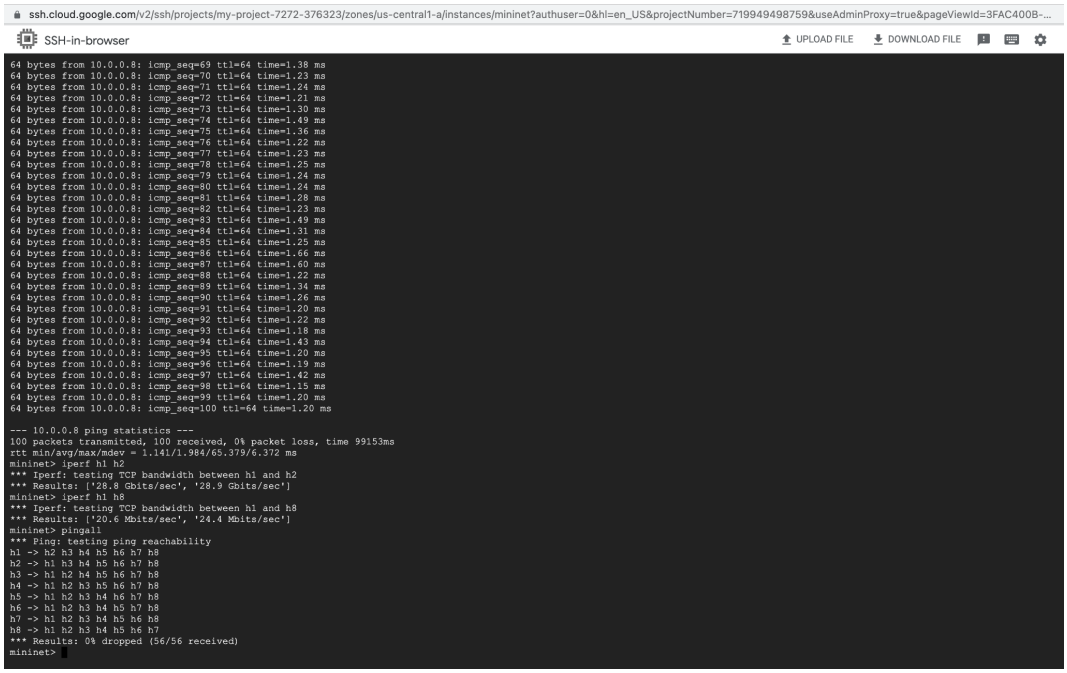
Answer 10:
The performance of a network can be impacted by the way packets are forwarded. In task 3, each packet is sent to the controller to determine where it should be forwarded, causing delays and reducing bandwidth. Task 4 improves performance by having the switch learn MAC addresses and install flow rules to forward packets without involving the controller, resulting in lower latency and higher bandwidth. Adding flow rules makes packet forwarding more efficient and improves network performance.

Q.11 Run pingall to verify connectivity and dump the output.

Controller Image:

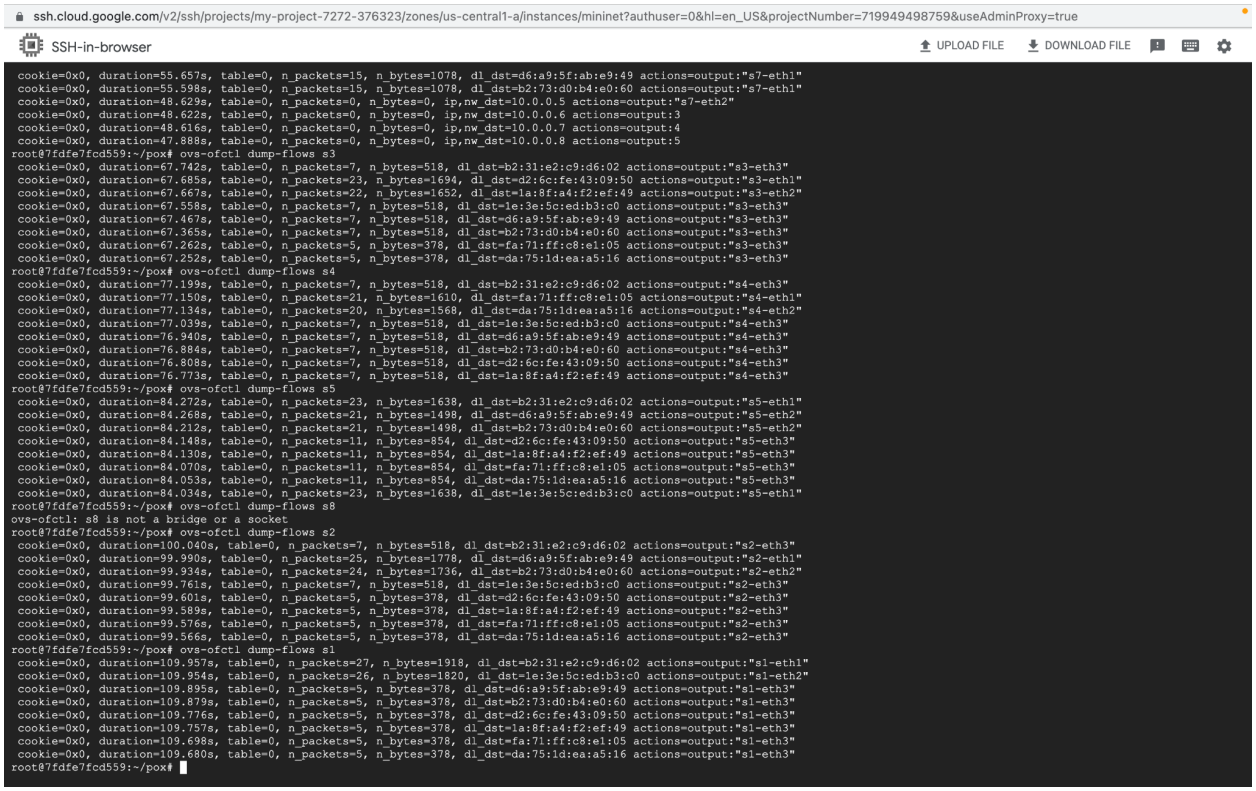


Mininet Image:



Q.12 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 12 :



Based on the output of "ovs-ofctl dump-flows s3" command, there are 8 flow rules for switch s3. Each flow entry represents a rule that the switch uses to forward packets based on their attributes, such as source/destination MAC addresses, VLAN tags, and IP addresses.

These flow entries define how packets are forwarded by the switch based on their attributes, such as MAC addresses and VLAN tags. For instance, the first flow entry has a cookie, duration, table number, and statistics related to the number of packets and bytes that matched this flow. The flow rule matches on a specific destination MAC address and specifies that packets should be forwarded out of port "s3-eth1". The other flow entries follow a similar structure, with different match criteria and actions.

Dump for s3:

cookie=0x0, duration=67.742s, table=0, n_packets=7, n_bytes=518, dl_dst=b2:31:e2:c9:d6:02 actions=output:"s3-eth3"
cookie=0x0, duration=67.685s, table=0, n_packets=23, n_bytes=1694, dl_dst=d2:6c:fe:43:09:50 actions=output:"s3-eth1"
cookie=0x0, duration=67.667s, table=0, n_packets=22, n_bytes=1652, dl_dst=1a:8f:a4:f2:ef:49 actions=output:"s3-eth2"
cookie=0x0, duration=67.558s, table=0, n_packets=7, n_bytes=518, dl_dst=1e:3e:5c:ed:b3:c0 actions=output:"s3-eth3"
cookie=0x0, duration=67.467s, table=0, n_packets=7, n_bytes=518, dl_dst=d6:a9:5f:ab:e9:49 actions=output:"s3-eth3"
cookie=0x0, duration=67.365s, table=0, n_packets=7, n_bytes=518, dl_dst=b2:73:d0:b4:e0:60 actions=output:"s3-eth3"

cookie=0x0, duration=67.262s, table=0, n_packets=5, n_bytes=378, dl_dst=fa:71:ff:c8:e1:05 actions=output:"s3-eth3"
cookie=0x0, duration=67.252s, table=0, n_packets=5, n_bytes=378, dl_dst=da:75:1d:ea:a5:16 actions=output:"s3-eth3"

For example, let us consider the first flow entry:

cookie=0x0, duration=67.742s, table=0, n_packets=7, n_bytes=518, dl_dst=b2:31:e2:c9:d6:02 actions=output:"s3-eth3"

Here:

cookie=0x0 - The cookie used by the controller to identify the rule.

duration=67.742 s - The time finished since the flow was added to the switch.

table=0 - This is the table in which this flow is installed (value 0 = default table).

n_packets=7 - This is the number of packets that have matched this flow so far.

n_bytes=518 - This is the total number of bytes that have been sent through this flow so far.

dl_dst=b2:31:e2:c9:d6:02 - This is the destination MAC address that this flow matches on.

actions=output:"s3-eth3": This is the action taken by the switch when a packet matches this flow - in this case, the packet is forwarded out of port "s3-eth3".

Task 5

ovs-ofctl add-flow s7 ip,nw_dst=10.0.0.5,actions=output:2
ovs-ofctl add-flow s7 ip,nw_dst=10.0.0.6,actions=output:3
ovs-ofctl add-flow s7 ip,nw_dst=10.0.0.7,actions=output:4
ovs-ofctl add-flow s7 ip,nw_dst=10.0.0.8,actions=output:5

The add flow rules for switch 7, the switch will use the above mentioned rules to forward IP packets that have different destination IP addresses to different output ports.

Meaning of above rules :

IP address = 10.0.0.5 then send packet out of port 2
IP address = 10.0.0.6 then send packet out of port 3
IP address = 10.0.0.7 then send packet out of port 4
IP address = 10.0.0.8 then send packet out of port 5

These rules help to control traffic forwarding in the network and are useful for configuring software defined network (SDN) where administrators control behavior of network switches.