Cloud Computing Mini Project 3

- 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?

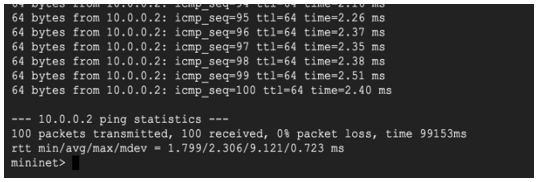
Answer:

_receive_packet(event) ⇒_handle_PacketIn(event) ⇒ act_like_hub(packet, packet_in) OR act_like_switch(packet, packet_in) ⇒ resend_packet(packet_in, out_port) OR (if act_like_switch) self.mac_to_port, msg.match, and resend_packet(packet_in, out_port) ⇒ self.connection.send(msg)

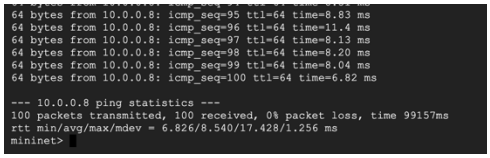
Here, flow starts from the __init__ function goes to _handle_packet and act_like_hub and lastly to resend_packet as _handle_packet and act_like_hub will call resend_packet.

- 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:
h1 ping-c100 h2



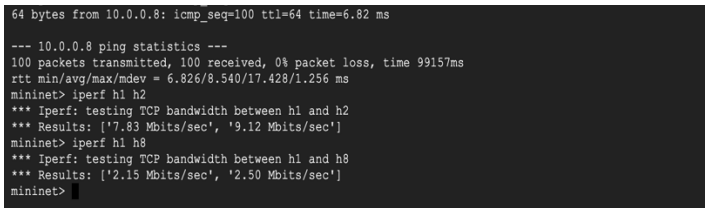
h1 ping-c100 h8



h8 is far away from h1 so, in order to travel to h8, multiple jumps and hops are taken. So h8 takes more time also h1 and h8 are both connected to two different switches i.e. s1 and s4 where as with h2 it is connected by the same switch in the topology so the packets redirect to each other very quick as compared to h1 and h8 with respect to the binary tree given.

- 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:



Iperf is used for bandwidth testing. It is a cross-platform tool which is widely used for the network performance measurement and tuning. There is a huge difference because for h1 and h2 as per the binary tree they are connect by one switch i.e. s1 and as for h1 and h8 they are connected by multiple switches hence it has lower throughput than h1 and h2.

- 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:

All the switches from s1 to s7 face the issue of traffic. Add print(f"***Sw: {self.connection.dpid}***") in the function act_like_hub() function. By running the mininet in one terminal and then pinging the same on the another terminal method, can be used to observe the traffic.

Task 3:

- 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:

The function accepts three parameters: self, packet, and packet_in. Self is an instance of a switch object, packet is the incoming packet to be processed, and packet_in is the packet's associated input port.

The initial effort made by the function is to identify the port connected to the incoming packet's source MAC address. The switch adds a record to the packet_in.in_port table associating the source MAC address with the packet's input port (self.mac_to_port) if the source MAC address is not already there.

The function then checks the switch's MAC-to-port database (self.mac_to_port) to see if the packet's destination MAC address is already known. The packet is sent out of the connected device if the destination MAC address is known.

The function floods the packet out to all ports except the input port using the self if the destination MAC address is unknown in the MAC-to-port table.the of the resend_packet() method.Flag OFPP_ALL. In order to manage packets with unknown destinations, network switches frequently employ this technique, which broadcasts the packet to all other ports in the hopes that the intended recipient will react.

For debugging reasons, the function additionally provides print statements that show information about the discovered MAC-to-port relationships and the actions executed for incoming packets with known or unknowable destinations.

We use ping command to send packets to other host

Switch is used to forward packet to the controller

The tasks performed by the controller as following –

If (Host sends the first packet)

MAC address and source port is saved and learnt.

Further, port number gets attached to MAC address and packet is sent using the same port.

Else (not the first time)

If(port associated with MAC is known)

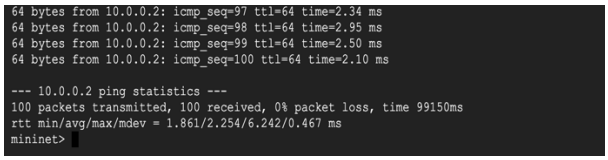
Packet is sent through that port

Else (Port is not known)

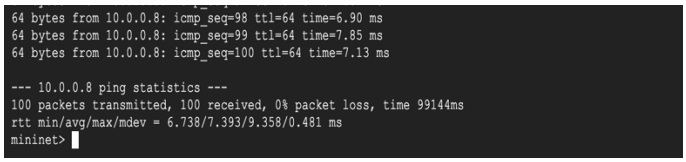
Packet is sent to all ports except the input port.

- 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)?

h1 ping -c100 h2

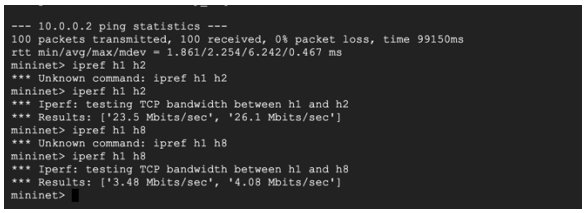


h1 ping -c100 h8



Yes, with respect to Task 2 the average has decreased down slightly.
From h1 to h2 the avg is 2.254 and for h1 to h8 it is 7.393.

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

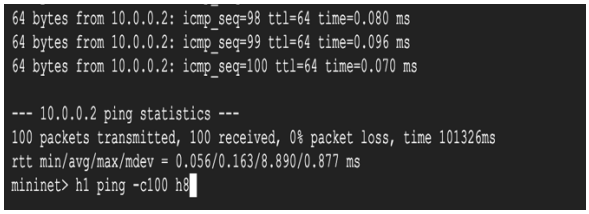


For ipref h1 h2 the throughput has increased drastically and for ipref h1 h8 the throughput has a slight increase.

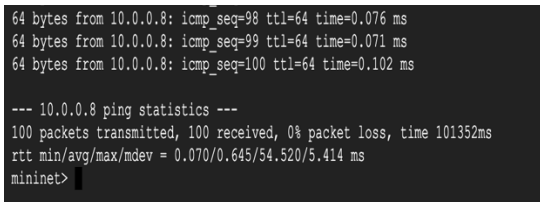
Task 4

- Q.1 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)?

h1 ping -c100 h2

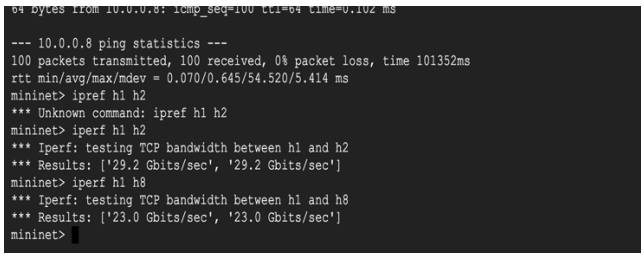


h1 ping -c100 h8



For pinging from h1 to h2 the avg is 0.163 and for h1 to h8 it is 0.645, which is a huge drop from the averages we got from task 3.

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



Throughput of task 4 is better than the throughput of task 3 as we can see the throughput of task 4 is in Gbites/sec whereas throughput of task 2 & 3 was in Mbits/sec.

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

  The result of task 4 is the best as compared to the other task i.e. from 2 and 3 also the same with performance and throughput because we have used open flow controllers which is way better than mac learning controller which installs flow rule which ahead it handles future packets and moving ahead mac learning controller is better than SDN controller.

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

```
*** Starting CLI:
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>
```

```
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=381>
<Host h2: h2-eth0:10.0.0.2 pid=383>
<Host h3: h3-eth0:10.0.0.3 pid=385>
<Host h4: h4-eth0:10.0.0.4 pid=387>
<Host h5: h5-eth0:10.0.0.5 pid=389>
<Host h6: h6-eth0:10.0.0.6 pid=391>
<Host h7: h7-eth0:10.0.0.7 pid=393>
<Host h8: h8-eth0:10.0.0.8 pid=395>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=400>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=403>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=406>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=409>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None pid=412>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None pid=415>
<OVSSwitch s7: lo:127.0.0.1,s7-eth1:None,s7-eth2:None pid=418>
<RemoteController c0: 127.0.0.1:6633 pid=373>
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)?

```
    rlist, wlist, elist = yield Select(sockets, [], sockets, 5)
GeneratorExit
DEBUG:openflow.of_01:No longer listening for connections
root@18fabd91b115:~/pox# ovs-ofctl dump-flows s1
cookie=0x0, duration=332.791s, table=0, n_packets=21, n_bytes=1890, dl_dst=33:33:00:00:00:16 actions=ALL
cookie=0x0, duration=332.540s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:91:ee:f2 actions=ALL
cookie=0x0, duration=332.513s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:8e:db:fd actions=ALL
cookie=0x0, duration=332.435s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:64:05:1c actions=ALL
cookie=0x0, duration=332.348s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:9e:e2:94 actions=ALL
cookie=0x0, duration=332.250s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:78:8f:72 actions=ALL
cookie=0x0, duration=332.129s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:e8:ed:3d actions=ALL
cookie=0x0, duration=332.092s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:8d:2d:53 actions=ALL
cookie=0x0, duration=331.961s, table=0, n_packets=0, n_bytes=0, dl_dst=33:33:ff:ab:78:2e actions=ALL
cookie=0x0, duration=331.513s, table=0, n_packets=55, n_bytes=3850, dl_dst=33:33:00:00:00:02 actions=ALL
cookie=0x0, duration=214.839s, table=0, n_packets=27, n_bytes=1134, dl_dst=ff:ff:ff:ff:ff:ff actions=ALL
cookie=0x0, duration=214.837s, table=0, n_packets=27, n_bytes=1918, dl_dst=1a:b7:a9:8e:db:fd actions=output:"s1-eth1"
cookie=0x0, duration=214.831s, table=0, n_packets=26, n_bytes=1820, dl_dst=aa:8f:9e:e8:ed:3d actions=output:"s1-eth2"
cookie=0x0, duration=214.778s, table=0, n_packets=5, n_bytes=378, dl_dst=fa:14:e4:9e:e2:94 actions=output:"s1-eth3"
cookie=0x0, duration=214.770s, table=0, n_packets=5, n_bytes=378, dl_dst=0a:74:da:8d:2d:53 actions=output:"s1-eth3"
cookie=0x0, duration=214.757s, table=0, n_packets=5, n_bytes=378, dl_dst=c2:9c:fe:91:ee:f2 actions=output:"s1-eth3"
cookie=0x0, duration=214.702s, table=0, n_packets=5, n_bytes=378, dl_dst=f2:92:d0:64:05:1c actions=output:"s1-eth3"
cookie=0x0, duration=214.690s, table=0, n_packets=5, n_bytes=378, dl_dst=e6:6a:34:ab:78:2e actions=output:"s1-eth3"
cookie=0x0, duration=214.633s, table=0, n_packets=5, n_bytes=378, dl_dst=8e:05:a0:78:8f:72 actions=output:"s1-eth3"
root@18fabd91b115:~/pox#
```

There are total 8 rules. For e.g. for h1 sends packets to destination address to other hubs.

This indicates that the switch is using MAC-based forwarding to direct traffic within the network. This is a dump of the OpenFlow flows installed in the switch named "s1". The flows are organized in tables, and this dump shows the flows installed in table 0. cookie: a value that can be used to identify the flow

duration: how long the flow has been active

table: the table where the flow is installed

n_packets: the number of packets matched by the flow

n_bytes: the number of bytes matched by the flow

dl_dst: the destination MAC address to match against

Task 5:

ovs-vsctl add-br s7

ovs-vsctl set Bridge s7 protocols=OpenFlow13

ovs-ofctl --protocol=OpenFlow13 add-flow s7
priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.5,actions=output:2

ovs-ofctl --protocol=OpenFlow13 add-flow s7
priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.6,actions=output:2

ovs-ofctl --protocol=OpenFlow13 add-flow s7
priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.7,actions=output:2

ovs-ofctl --protocol=OpenFlow13 add-flow s7
priority=100,ip,nw_src=10.0.0.1,nw_dst=10.0.0.8,actions=output:2