Assignment 3

Sai Krishna Kasarapu

B00979383

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 :

Function call graph:

The order in which the controller functions executed are:

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 first 3 functions launch, start_switch , __init__ executes when we start the controller, the remaining 3 functions _handle_PacketIn, act_like_hub, resend_packet are the functions that execute whenever a packet is received, so each time a packet is received these 3 functions will be executed in order.

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?

Q 2 Answer:

When h1 pings h2(h1 ping h2), the average is 2.429 ms

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99142ms
rtt min/avg/max/mdev = 1.907/2.429/7.674/0.642 ms
mininet>
```

When h1 pings h8(h1 ping h8), the average is 8.794 ms

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99154ms
rtt min/avg/max/mdev = 7.837/8.794/12.298/0.725 ms
mininet>
```

The difference is when h1 pings h2 it took less average compared to the average when h1 pings h8.The reason is here in our topology the hosts h1 and h2 are connected to the same switch which results in quicker learning of source and destination MAC addresses and also packets took less hops and less time to reach from source to destination as they are connected to same switch, in other case(when h1 pings h8), here h1 is connected to switch 1 and h8 is connected to switch 4, so the packets require multiple hops to go from h1 to h8 which introduces latency and obviously more time.

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?

Q 3 Answer :

Iperf is used to measure network performance and tuning, and also used to measure the throughput of a network. In other words it is used to measure the maximum TCP and UDP bandwidth performance in a network.

The throughput for h1,h2 is [7.80 Mbps, 7.72 Mbps]

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['7.80 Mbits/sec', '7.72 Mbits/sec']
mininet>
```

The throughput for h1, h8 is [2.49 Mbps, 2.47 Mbps]

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.49 Mbits/sec', '2.47 Mbits/sec']
mininet>
```

The difference is for iperf h1 h2 has higher throughput compared to throughput of iperf h1 h8. The reason is as h1 and h2 are connected to same switch, the traffic between hosts h1 and h2 wont traverse multiple switches, and packets wont need to travel by multiple links and hops to go from h1 to h2 which results is in higher data transfer rate.On the other hand for iperf h1 h8 , both hosts connected to different switches(h1->s1, h8->s4), so the packets needs to travel through all the switches in its data path towards s4 and also by multiple hops and links, so this is results is less throughput compared to the other case.

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 that are in the topology observe traffic.we can find which switches observing traffic by adding the print function in handle_PacketIn (self, event) function in the of_tutorial.py 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]
INFO:openflow.of_01:[00-00-00-00-00-05 8] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-05 8]
switch observing traffic is  4
switch observing traffic is  3
switch observing traffic is  6
switch observing traffic is  6
switch observing traffic is  7
switch observing traffic is  3
switch observing traffic is  5
switch observing traffic is  1
switch observing traffic is  2
switch observing traffic is  7
switch observing traffic is  4
switch observing traffic is  5
switch observing traffic is  1
switch observing traffic is  2
switch observing traffic is  3
switch observing traffic is  6
switch observing traffic is  4
switch observing traffic is  7
switch observing traffic is  5
switch observing traffic is  1
switch observing traffic is  2
switch observing traffic is  1
switch observing traffic is  5
switch observing traffic is  7
switch observing traffic is  2
switch observing traffic is  6
switch observing traffic is  4
switch observing traffic is  3
switch observing traffic is  1
switch observing traffic is  5
```

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).

```
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:

How above code works:

→So, whenever a packet receives the switch,in the first if case, the above controller code checks whether the Source MAC address(MAC address of host from which packets are coming) is already mapped in the mac_to_port dictionary or not,if it is not present in the mac_to_port then it creates an entry in the dictionary by mapping source MAC address to the port on which the packet is received.

→In the second if case, the controller checks whether the destination MAC address is already known or present in the mac_to_port dictionary or not, if it presents then it forwards the

packet to the corresponding port number associated wit that destination host MAC address.

→else, the controller floods the packet to the all the ports except the port from which packet is received.

Example: Assume host h1 is sending packet to host h2,also assume initially mac_to_port is empty.So, when h1 pings h2, host h1 send a packet to the host h2,so the switch connected to host1 recieves the packet, now the switch checks whether the h1 MAC address is already mapped in the dictionary or not, if not then it adds an entry for h1 MAC address to the incoming port from which the packet is received, now the controller checks the h2 MAC address(destination MAC address) is mapped in the dictionary or not, if it is present then controller floods the packet to the port associated with host2 MAC address, if mapping is not present then controller floods packet to all the switches and then the switch connected to host2 will learning mapping and when next time when packet comes to host2 it will directly send to that port number instead of flooding to all the switches.

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:

When h1 pings h2 average is 2.295 ms

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99149ms
rtt min/avg/max/mdev = 1.861/2.295/2.993/0.210 ms
mininet>
```

When h1 pings h8, average is 8.281 ms

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99148ms
rtt min/avg/max/mdev = 7.156/8.281/15.861/1.072 ms
mininet>
```

The difference is compared to task 2(hub case), this switch case have got a slightly better average.

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

Answer:

Throughput for h1, h2 is [14.7 Mbps, 14.5 Mbps]

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['14.7 Mbits/sec', '14.5 Mbits/sec']
mininet>
```

Throughput for h1, h8 is [3.27 Mbps, 3.19Mbps]

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.27 Mbits/sec', '3.19 Mbits/sec']
mininet>
```

The difference of this switch case from task 2(hub case) is that compared to task2 here we got better throughput.

Compared to task 2 , in task 3 we got better average and better throughput, because in task2 the controller is flooding the packet to all the switches whenever it receives a packet as the switch in this case is acting like a hub, but in task3 as we modified the code the switch ,now it learns the mapping and floods the packets to the destination port only if it knowns the mapping is already present in the dictionary else it

floods to all the switches, so here we have reduced traffic compared to task2 , that's why we are getting better results in this case.

Task 4:

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:

The average when h1 pings h2 for 100 times is 0.077ms .Yes, there is a difference when compared to task 3, here we got a very far better results. Overall we got better ping compared to task 3.

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101374ms
rtt min/avg/max/mdev = 0.055/0.077/0.100/0.012 ms
mininet>
```

The average when h1 pings h8 for 100 times is 0.100 ms. Here also we got a far better results in the ping compared to task 3.

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101370ms
rtt min/avg/max/mdev = 0.064/0.100/0.597/0.053 ms
mininet>
```

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

Answer:

The throughput for h1 , h2 is 24.0 Gbps

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['24.0 Gbits/sec', '24.0 Gbits/sec']
mininet>
```

The throughput for h1, h8 is 19.9 Gbps

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['19.9 Gbits/sec', '19.9 Gbits/sec']
mininet>
```

Compared to task3 here we got a far better throughput , as we can notice that the bandwidth in task4 is in Gbps where as in task3 it is in Mbps, which is a very better throughput.

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

Answer : The results got better. The reason is in task 3 controller code, whenever a packet arrives it learns the mapping and checks the destination MAC address is present in the mapping table or not , if not present it floods packet to all switches, if present the switch will forward it to the corresponding port associated with that destination MAC address, but again when another packet with same destination comes it will again checks the mapping and if present it will directly send the packet to that port, but in task 4 whenever the controller learns the mapping it will do one more thing along with forwarding the packet, the thing is it will install flow rules in the switch and instructs the switch how to handle the future packets also. So, whenever packets with that particular destination address comes,the switch will automatically forwards the packets to the destination port in efficient

path . The switch will independently handles the packets with known destination MAC addresses. This results in faster packet transmission and as a result we got better results in task 4.

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 :

Below are the openflow entries in the flow table of openFlow switch.For each openFlow switch we have 8 flow entries. We have 8 entries because each represents a hub.

For example if we take a flow entry of switch s1:

Cookie=0x0,duration=150.653s,table=0,n_packets=27,n_bytes=1918, dl_dst=da:0d:e3:1c:d:91 actions=output:"s1-eth1"

In each flow entry we have various parameters as observed below, where cookie field carries some metadata about the flow entry, the duration field represents the from till now for how much time the flow entry is in, n_packets represents how many packets have matched this flow entry similarly how many bytes matched with this flow entry for n_bytes, dl_dst field represents the destination host MAC address to which the packets will be forwarded, table field represents table id, actions field represents the output port of switch by which the packets needs to be outputed.

```
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s1
 cookie=0x0, duration=150.653s, table=0, n_packets=27, n_bytes=1918, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s1-eth1"
 cookie=0x0, duration=150.609s, table=0, n_packets=26, n_bytes=1820, dl_dst=4a:5f:22:52:e7:0a actions=output:"s1-eth2"
 cookie=0x0, duration=150.553s, table=0, n_packets=5, n_bytes=378, dl_dst=16:d2:9c:85:4a:ff actions=output:"s1-eth3"
 cookie=0x0, duration=150.503s, table=0, n_packets=5, n_bytes=378, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s1-eth3"
 cookie=0x0, duration=150.445s, table=0, n_packets=5, n_bytes=378, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s1-eth3"
 cookie=0x0, duration=150.390s, table=0, n_packets=5, n_bytes=378, dl_dst=8e:c3:99:72:b7:75 actions=output:"s1-eth3"
 cookie=0x0, duration=150.333s, table=0, n_packets=5, n_bytes=378, dl_dst=06:58:15:ea:1d:ea actions=output:"s1-eth3"
 cookie=0x0, duration=150.264s, table=0, n_packets=5, n_bytes=378, dl_dst=7e:49:58:a6:48:3c actions=output:"s1-eth3"
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s2
 cookie=0x0, duration=197.625s, table=0, n_packets=7, n_bytes=518, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s2-eth1"
 cookie=0x0, duration=197.614s, table=0, n_packets=25, n_bytes=1778, dl_dst=16:d2:9c:85:4a:ff actions=output:"s2-eth2"
 cookie=0x0, duration=197.563s, table=0, n_packets=24, n_bytes=1736, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s2-eth3"
 cookie=0x0, duration=197.241s, table=0, n_packets=6, n_bytes=420, dl_dst=4a:5f:22:52:e7:0a actions=output:"s2-eth1"
 cookie=0x0, duration=197.049s, table=0, n_packets=5, n_bytes=378, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s2-eth1"
 cookie=0x0, duration=197.037s, table=0, n_packets=5, n_bytes=378, dl_dst=8e:c3:99:72:b7:75 actions=output:"s2-eth1"
 cookie=0x0, duration=196.945s, table=0, n_packets=5, n_bytes=378, dl_dst=06:58:15:ea:1d:ea actions=output:"s2-eth1"
 cookie=0x0, duration=196.925s, table=0, n_packets=5, n_bytes=378, dl_dst=7e:49:58:a6:48:3c actions=output:"s2-eth1"
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s3
 cookie=0x0, duration=253.597s, table=0, n_packets=7, n_bytes=518, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s3-eth1"
 cookie=0x0, duration=253.549s, table=0, n_packets=23, n_bytes=1694, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s3-eth2"
 cookie=0x0, duration=253.489s, table=0, n_packets=22, n_bytes=1652, dl_dst=8e:c3:99:72:b7:75 actions=output:"s3-eth3"
 cookie=0x0, duration=253.261s, table=0, n_packets=6, n_bytes=420, dl_dst=4a:5f:22:52:e7:0a actions=output:"s3-eth1"
 cookie=0x0, duration=253.129s, table=0, n_packets=6, n_bytes=420, dl_dst=16:d2:9c:85:4a:ff actions=output:"s3-eth1"
 cookie=0x0, duration=252.913s, table=0, n_packets=6, n_bytes=420, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s3-eth1"
 cookie=0x0, duration=252.741s, table=0, n_packets=5, n_bytes=378, dl_dst=06:58:15:ea:1d:ea actions=output:"s3-eth1"
 cookie=0x0, duration=252.719s, table=0, n_packets=5, n_bytes=378, dl_dst=7e:49:58:a6:48:3c actions=output:"s3-eth1"
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s4
 cookie=0x0, duration=344.245s, table=0, n_packets=7, n_bytes=518, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s4-eth1"
 cookie=0x0, duration=344.201s, table=0, n_packets=21, n_bytes=1610, dl_dst=06:58:15:ea:1d:ea actions=output:"s4-eth2"
 cookie=0x0, duration=344.145s, table=0, n_packets=20, n_bytes=1568, dl_dst=7e:49:58:a6:48:3c actions=output:"s4-eth3"
 cookie=0x0, duration=343.969s, table=0, n_packets=6, n_bytes=420, dl_dst=4a:5f:22:52:e7:0a actions=output:"s4-eth1"
 cookie=0x0, duration=343.793s, table=0, n_packets=6, n_bytes=420, dl_dst=16:d2:9c:85:4a:ff actions=output:"s4-eth1"
 cookie=0x0, duration=343.627s, table=0, n_packets=6, n_bytes=420, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s4-eth1"
 cookie=0x0, duration=343.509s, table=0, n_packets=6, n_bytes=420, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s4-eth1"
 cookie=0x0, duration=343.401s, table=0, n_packets=6, n_bytes=420, dl_dst=8e:c3:99:72:b7:75 actions=output:"s4-eth1"
```

```
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s5
 cookie=0x0, duration=459.957s, table=0, n_packets=23, n_bytes=1638, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s5-eth1"
 cookie=0x0, duration=459.913s, table=0, n_packets=21, n_bytes=1498, dl_dst=16:d2:9c:85:4a:ff actions=output:"s5-eth2"
 cookie=0x0, duration=459.861s, table=0, n_packets=21, n_bytes=1498, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s5-eth2"
 cookie=0x0, duration=459.805s, table=0, n_packets=11, n_bytes=854, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s5-eth3"
 cookie=0x0, duration=459.748s, table=0, n_packets=11, n_bytes=854, dl_dst=8e:c3:99:72:b7:75 actions=output:"s5-eth3"
 cookie=0x0, duration=459.691s, table=0, n_packets=11, n_bytes=854, dl_dst=06:58:15:ea:1d:ea actions=output:"s5-eth3"
 cookie=0x0, duration=459.593s, table=0, n_packets=11, n_bytes=854, dl_dst=7e:49:58:a6:48:3c actions=output:"s5-eth3"
 cookie=0x0, duration=459.533s, table=0, n_packets=23, n_bytes=1638, dl_dst=4a:5f:22:52:e7:0a actions=output:"s5-eth1"
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s6
 cookie=0x0, duration=465.396s, table=0, n_packets=15, n_bytes=1078, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s6-eth1"
 cookie=0x0, duration=465.383s, table=0, n_packets=19, n_bytes=1414, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s6-eth2"
 cookie=0x0, duration=465.325s, table=0, n_packets=19, n_bytes=1414, dl_dst=8e:c3:99:72:b7:75 actions=output:"s6-eth2"
 cookie=0x0, duration=465.236s, table=0, n_packets=17, n_bytes=1330, dl_dst=06:58:15:ea:1d:ea actions=output:"s6-eth3"
 cookie=0x0, duration=465.176s, table=0, n_packets=17, n_bytes=1330, dl_dst=7e:49:58:a6:48:3c actions=output:"s6-eth3"
 cookie=0x0, duration=465.064s, table=0, n_packets=15, n_bytes=1078, dl_dst=4a:5f:22:52:e7:0a actions=output:"s6-eth1"
 cookie=0x0, duration=464.932s, table=0, n_packets=15, n_bytes=1078, dl_dst=16:d2:9c:85:4a:ff actions=output:"s6-eth1"
 cookie=0x0, duration=464.712s, table=0, n_packets=15, n_bytes=1078, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s6-eth1"
root@2b4eeea97b9a:~# ovs-ofctl dump-flows s7
 cookie=0x0, duration=472.313s, table=0, n_packets=15, n_bytes=1078, dl_dst=da:0d:e3:1c:0d:91 actions=output:"s7-eth1"
 cookie=0x0, duration=472.272s, table=0, n_packets=11, n_bytes=854, dl_dst=c2:d9:10:1a:5f:d5 actions=output:"s7-eth2"
 cookie=0x0, duration=472.214s, table=0, n_packets=11, n_bytes=854, dl_dst=8e:c3:99:72:b7:75 actions=output:"s7-eth2"
 cookie=0x0, duration=472.157s, table=0, n_packets=11, n_bytes=854, dl_dst=06:58:15:ea:1d:ea actions=output:"s7-eth2"
 cookie=0x0, duration=472.060s, table=0, n_packets=11, n_bytes=854, dl_dst=7e:49:58:a6:48:3c actions=output:"s7-eth2"
 cookie=0x0, duration=471.940s, table=0, n_packets=15, n_bytes=1078, dl_dst=4a:5f:22:52:e7:0a actions=output:"s7-eth1"
 cookie=0x0, duration=471.808s, table=0, n_packets=15, n_bytes=1078, dl_dst=16:d2:9c:85:4a:ff actions=output:"s7-eth1"
 cookie=0x0, duration=471.592s, table=0, n_packets=15, n_bytes=1078, dl_dst=7e:fb:04:7d:8a:40 actions=output:"s7-eth1"
root@2b4eeea97b9a:~#
```