Assignment 2- Report

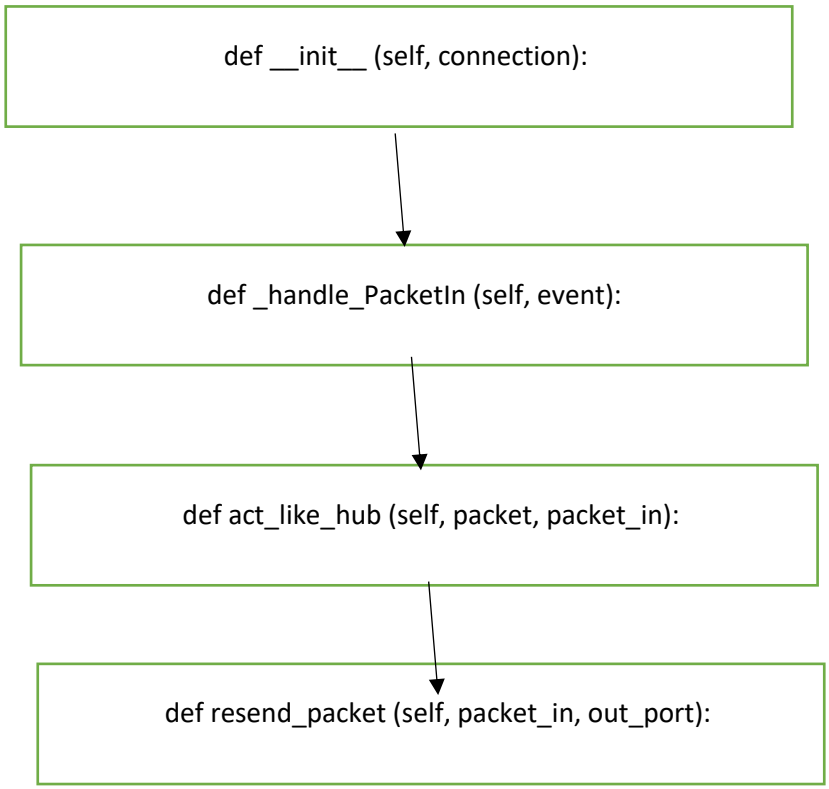
Task 1

Below is the proof of tree topology.

```
root@99fdb6e3e717:~# mn --custom binary_tree.py --topo mytopo
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (h7, s4) (h8, s4) (s1, s5)
 (s2, s5) (s3, s6) (s4, s6) (s5, s7) (s6, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> h1 ping h8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=9.59 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.545 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.112 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.071 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.067 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.091 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.074 ms
^C
--- 10.0.0.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7005ms
rtt min/avg/max/mdev = 0.062/1.326/9.590/3.127 ms
```

Task 2

Ans 1: Function call graph for the controller:



Ans 2:

h1 ping –c100 h2 - Average: 28.424 ms

h1 ping –c100 h8 – Average: 35.864 ms

Difference: 7.44 ms (Case 1 is faster)

Reason: There is a difference of about 7 ms between the two cases because for the first case the packet has to travel through only one switch(s1) and it reaches h2, whereas, for the second case the packet has to go through all the switches, until it reaches s4 which has h8 as a child. This is why it takes more time for h1 to ping h8

Ans 3:

iperf h1 h2 – 7.90 Mbits/sec, 9.09 Mbits

iperf h1 h8- 2.14 Mbits/sec , 2.54 Mbits.

Iperf is a cross-platform tool used to produce standardized performance measurements for a network. It measures throughput between two ends (client and server).

There is a significant difference between the two (around 5Mbits and 7Mbits) because the average time for ping from h1 to h8 is greater than h1 to h2. As the throughput decreases if the average time increases, the throughput for h1 h8 is lesser compared to h1 h2.

Ans 4: All switches observe the traffic.
The traffic can be observed by looking at the number of packets transferred.

Task 3

Ans 1:

Using h1 ping h2:
When a packet is set from h1 to h2, it goes to s1 and from there it goes to the controller which learns and stores the port for h1. Then, because the port for h2 is not known, it will flood the packet to every port except the input port.

Ans 2:

h1 ping –c100 h2 - Average 28.413 ms

h1 ping –c100 h8 – Average 38.173 ms

There is a very minute difference between the average of Task 2 and Task 3.

Ans 3:

iperf h1 h2 – ['9.81 Mbits/sec', '10.4 Mbits/sec']

iperf h1 h8- ['2.48 Mbits/sec', '2.99 Mbits/sec']

There is again a minute difference between the throughput for Task 2 and Task 3

Task 4

Ans 1:

h1 ping –c100 h2 - Average is 1.112 ms

h1 ping –c100 h8 – Average is 1.920 ms

Yes, there is a significant difference between the average for Task 3 and Task 4.

Ans 2:

iperf h1 h2 – ['28.3 Gbits/sec', '28.4 Gbits/sec']

iperf h1 h8- ['24.2 Mbits/sec', '24.3 Mbits/sec']

Yes, there is a significant difference between the throughputs for Task 3 and Task 4.

Ans 3:

The results for task 4 is much better as compared to task 4.
Reason: After transmitting the first packet, the controller learns the destination for it, and also forwards that information to the switch. Therefore, the next time the packet with same destination appears on the switch, it just reads the rule from the switch and it doesn't have to go to the controller, saving one hop. This leads to the average time for ping and the throughput being better than task 3.

Ans 4:

Result after pingall:

```
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)
```

After dumping the output:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=2800>
<Host h2: h2-eth0:10.0.0.2 pid=2802>
<Host h3: h3-eth0:10.0.0.3 pid=2804>
<Host h4: h4-eth0:10.0.0.4 pid=2806>
<Host h5: h5-eth0:10.0.0.5 pid=2808>
<Host h6: h6-eth0:10.0.0.6 pid=2810>
<Host h7: h7-eth0:10.0.0.7 pid=2812>
<Host h8: h8-eth0:10.0.0.8 pid=2814>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=2819>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=2822>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=2825>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=2828>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None pid=2831>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None pid=2834>
<OVSSwitch s7: lo:127.0.0.1,s7-eth1:None,s7-eth2:None pid=2837>
<RemoteController c0: 127.0.0.1:6633 pid=2792>
```

Ans 5:

For each OpenFlow switch, there are 8 rules.
Because there are a total of 8 hosts and each switch has flow entry of each host.
Below are the flow rules on switch S1.

```
root@f562c8645d11:~# ovs-ofctl dump-flows s1
 cookie=0x0, duration=1319.291s, table=0, n_packets=315169, n_bytes=20814586,
dl_dst=5a:a1:be:1f:9e:58 actions=output:"s1-eth1"
 cookie=0x0, duration=1319.290s, table=0, n_packets=341927, n_bytes=15194771526,
dl_dst=66:52:31:72:3e:b9 actions=output:"s1-eth3"
 cookie=0x0, duration=534.511s, table=0, n_packets=24, n_bytes=1680,
dl_dst=2a:ee:43:76:5f:46 actions=output:"s1-eth2"
 cookie=0x0, duration=534.510s, table=0, n_packets=4, n_bytes=280,
dl_dst=fa:b9:23:ba:17:69 actions=output:"s1-eth3"
 cookie=0x0, duration=529.553s, table=0, n_packets=1, n_bytes=42,
dl_dst=0a:8c:75:ca:ea:24 actions=output:"s1-eth3"
 cookie=0x0, duration=528.684s, table=0, n_packets=0, n_bytes=0,
dl_dst=9e:d9:8b:bf:e8:96 actions=output:"s1-eth3"
 cookie=0x0, duration=528.418s, table=0, n_packets=0, n_bytes=0,
dl_dst=42:50:d4:b2:9c:6d actions=output:"s1-eth3"
 cookie=0x0, duration=528.354s, table=0, n_packets=1, n_bytes=42,
dl_dst=2e:92:a8:20:36:d2 actions=output:"s1-eth3"
```

Taking an example for the first flow entry:

cookie=0x0, duration=1319.291s, table=0, n_packets=315169, n_bytes=20814586, dl_dst=5a:a1:be:1f:9e:58 actions=output:"s1-eth1"

Here,

Duration – refers to the time taken

dl_dst – means the mac address of the destination host

n_packets- refers to the number of packets transferred

n_bytes- refers to the number of bytes transferred

actions – refers to the action to be taken, here action to tbe take is output to port s1-eth1

Task 5

Below is an example of the command used to set up IP Matching rule:

ovs-ofctl add-flow s7
priority=65535,dl_type=0x0800,nw_src=10.0.0.1,nw_dst=10.0.0.8,actions=normal

This command adds flow rule to switch s7.
dl_type refers to the type of address (ip/Mac), here it represents IP.
Nw_src is the source IP and nw_ dst is the destination IP.
Actions refers to the type of action for this flow.

For the above command, a new rule is added to s7, where any packet coming from 10.0.0.1 is forwarded by switch 7 to 10.0.0.8.