CS580-MiniProject 3

Name- Parneet Lnu

BNumber- B00816285

Email- plnu2@binghamton.edu

TASK-2

Q1. 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?

Ans. The functions are called as follows:

1. The launch() function is called and in this start_switch() is called. In this a new tutorial object is created and connection_up event is fired.
2. Then the __init__() function is called where the connection to switch is created.
3. After this _handle_PacketIn() function is called.
4. From above function act_like_hub() (or act_like_switch()) is called.
5. Then at last resend_packet() is called.

Q2. Have h1 ping h2, and h1 ping h5 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?

Ans. To do h1 ping h2, avg time taken is 1.690 ms.

To do h1 ping h5, avg time taken is 5.551 ms.

Since, h1 and h2 have only one switch between them and hence the routing distance is less so the ping time less than h1-h5 where there are multiple switches (namely 5) in the path

Q3. Run "iperf h1 h2" and "iperf h1 h5". What is "iperf" used for? What is the throughput for each case? What is the difference, and why?

Ans. "iperf" is a tool used for active measurements of the maximum achievable bandwidth.

For "iperf h1 h2" throughput is 10.2 Mbits/sec and "iperf h1 h5" throughput is 3.04 Mbits/sec.

The difference between the two throughputs is of 7.16 where throughput of h1-h2 path is higher.

As mentioned, that there is longer path or multiple switches in h1-h5 than h1-h2, hence there is a higher probability of packet loss. So, the overall throughput is lower in h1-h5 path.

Q4. Which of the switches observe traffic (s1 _ s6)? Please describe the way for observing such traffic on switches (e.g., adding some "print" functions in the "of_tutorial" controller).

Ans. All the switches i.e. s1-s6 observe traffic since they all act as a hub. I used the following code in _handle_PacketIn() function to observe this:

```
p = str(self.connection.dpid)
log.debug("Using switch" + p)
```

TASK-3

Q1. 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).

Ans. When h1 ping h2 is done, the packet is transferred from h1 to the switch s1. After that the following steps take place in the code:

- The switch s1 learns from the port from source MAC. We check if the port is not in the forwarding table then we update it.
- Next, we check if the port associated with destination MAC is in the table. If yes, then we print a relevant message and forward the packet to destination i.e. to host h2.
- If not, then we print the relevant message and flood the packet to all the ports(i.e. to all other switches) except the input port.

Q2. (Please disable your output functions, i.e., print, before doing this experiment) Have h1 ping h2, and h1 ping h5 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)?

Ans. To do h1 ping h2, avg time taken is 2.349 ms.

To do h1 ping h5, avg time taken is 7.573 ms.

There is not a significant difference in ping time than in Task 2.

Q3. Run "iperf h1 h2" and "iperf h1 h5". What is the throughput for each case? What is the difference from Task II?

Ans. For "iperf h1 h2" throughput is 16.5 Mbits/sec and "iperf h1 h5" throughput is 3.08 Mbits/sec.

There is a little to no increase in throughput for each case.

TASK-4

Q1. 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)?

Ans. To do h1 ping h2, avg time taken is 0.055 ms.

To do h1 ping h5, avg time taken is 0.075 ms.

The average time for ping has decreased significantly as compared to Task3. Time has dropped to almost negligent against Task 3 results.

Q2. Run "iperf h1 h2" and "iperf h1 h8". What is the throughput for each case? What is the difference from Task III?

Ans. For "iperf h1 h2" throughput is 32.6 Gbits/sec and "iperf h1 h5" throughput is 26 Gbits/sec.

The throughput has improved by significant measure in Task3.

Q3. Please explain the above results—why the results become better or worse?

Ans. The results have become better because initially (i.e. in Task3) all the packets were still going to the controller even though the switch learns the destination port after first ping. After installing flow rules, the switch becomes smarter and all the packets are directly sent to the destination. This results in faster ping time and also much better throughput.

Q4. Run pingall to verify connectivity and dump the output.

Ans. This is a screenshot of the output:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=21076>
<Host h2: h2-eth0:10.0.0.2 pid=21078>
<Host h3: h3-eth0:10.0.0.3 pid=21080>
<Host h4: h4-eth0:10.0.0.4 pid=21082>
<Host h5: h5-eth0:10.0.0.5 pid=21084>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=21089>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=21092>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=21095>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=21098>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None pid=21101>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None pid=21104>
<RemoteController c0: 127.0.0.1:6633 pid=21068>
mininet>
```

Q5. 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)?

Ans. There are 5 rules for each OpenFlow switch because there are 5 other switches and it gives forwarding rules for all the other switches.
Here is an entry for switch s1:
cookie=0x0, duration=242.001s, table=0, n_packets=31, n_bytes=1974, dl_dst=f6:72:e5:3a:2e:66 actions=output:"s1-eth2"

- cookie=0x0 gives information if any cookies are stored in this flow entry. There are none in this case.
- duration=242.001s tells about how long this flow entry has been in flow table.
- table=0 tells us the table where flow entry is installed on.
- n_packets=31, it tells that 31 packets have matched this entry
- n_bytes=1974, it tells that 1974 bytes have matched this entry
- dl_dst=f6:72:e5:3a:2e:66, it gives information about the destination MAC address of the rule.
- actions=output:"s1-eth2", tells to transfer to port s1-eth2.

TASK-5

The steps to set-up IP-matching rules using ovs-ofctl commands is:

- Open mininet and enter the following commands for each of the IP-address of hosts(h1~h5) on switch s6:

```
sh ovs-ofctl add-flow s6 arp,nw_dst=10.0.0.1,actions=normal
sh ovs-ofctl add-flow s6 arp,nw_dst=10.0.0.2,actions=normal
sh ovs-ofctl add-flow s6 arp,nw_dst=10.0.0.3,actions=normal
sh ovs-ofctl add-flow s6 arp,nw_dst=10.0.0.4,actions=normal
sh ovs-ofctl add-flow s6 arp,nw_dst=10.0.0.5,actions=normal
```

- Then check open-flow rules for switch s6:

```
sh ovs-ofctl dump-flows s6
```

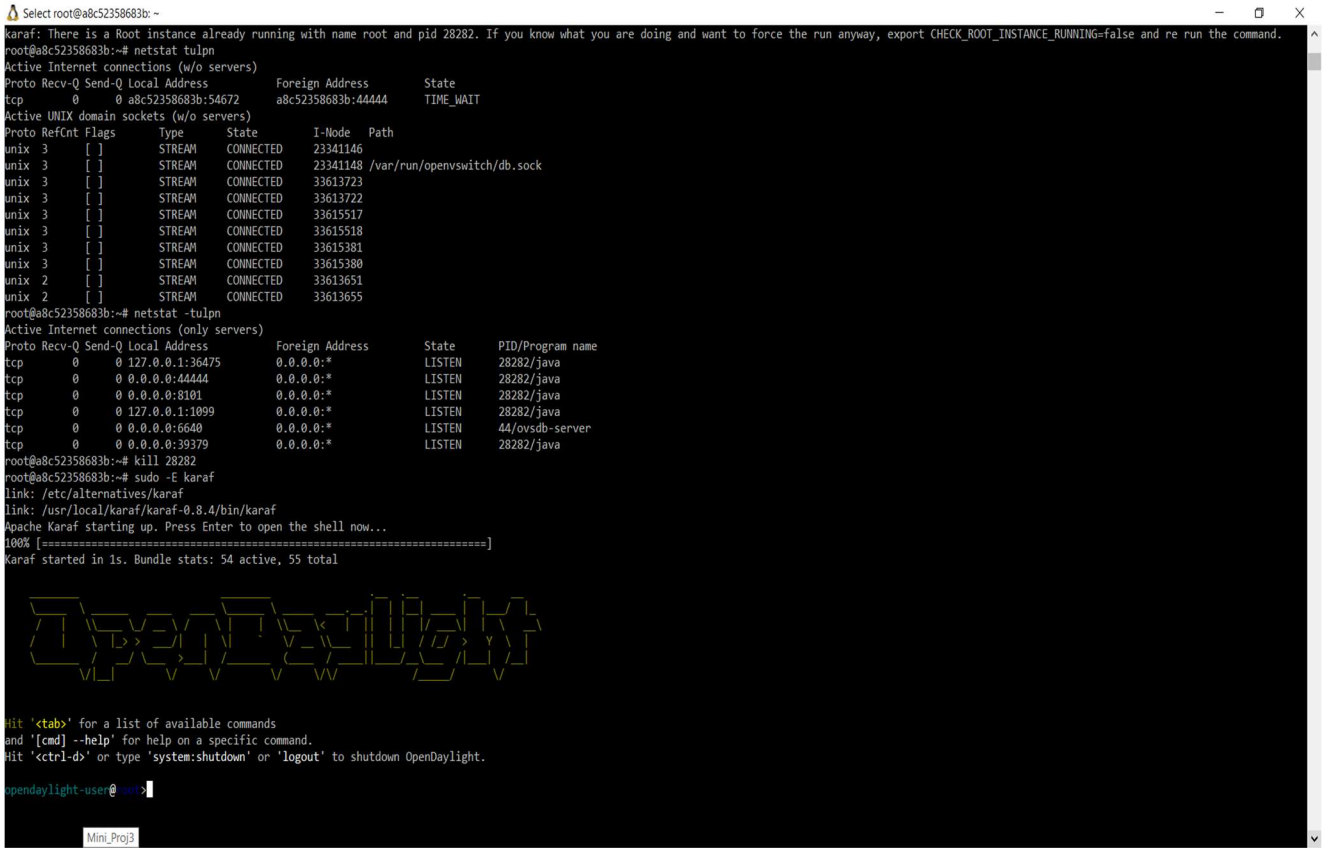
  It will show all the rules which are added in the previous switch.

- At end, check the connection using "pingall" and see if all packets are received.

TASK-1

```
root@1f8ab671f199:~# 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
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(s1, h1) (s1, h2) (s2, h3) (s2, h4) (s3, h5) (s4, s1) (s4, s2) (s5, s3) (s6, s4) (s6, s5)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=10.8 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.645 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.059 ms
^C
--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.059/3.845/10.831/4.945 ms
mininet>
```

Bonus Task



I have downloaded OpenDaylight but was unable to integrate with the Mininet as the odl-l2-switch UI was not working because it needed more memory than container has.