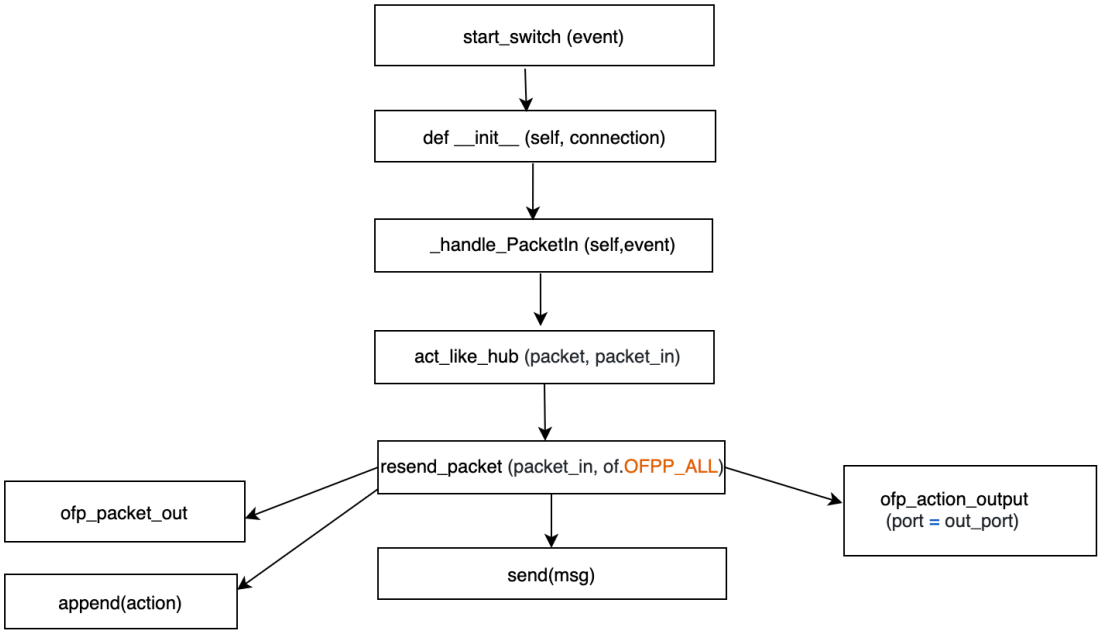
Task 2

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?



- The controller receives a packetIn message from a switch. such a packet will be sent to the SDN controller (i.e., “of_tutorial” controller) as there are no forwarding rules for the particular packet.

- The controller passes both the packet and the packet_in as arguments to act_like_hub.

- act_like_hub resends the packet out on all ports except the input port by calling the resend_packet function with parameters packet_in and constant of.OFPPP_ALL which makes to flood all ports except the input port.

- The resend_packet creates an ofp_packet_out message which is used to instruct a witch to send a packet. The controller the creates ofp_action_output and passes the port as an argument. ofp_action_output is then added to the ofp_packet_out message so the switch actually knows what to do with it. Then the controller appeds the action message. Finally, controller sends the message to the switch with the send function call with the parameter 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?

h1 ping h2 the average time is 27.28 ms

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 2.316/27.289/52.259/14.213 ms
```

h1 ping h8 the average time is 34.95 ms

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99148ms
rtt min/avg/max/mdev = 8.047/34.950/61.800/15.204 ms
```

The difference is 7.67 ms

The reason for the difference is in the first example from h1 to h2 only need to travel from h1 to switch s1 then to the controller and then back to s1 and then h2.

h1->s1->controller->s1->h2

In the second example packet travel from h1 to s1 to controller and back to s5 to the controller and back, then to s7, controller and back, then to s6, controller and back, then to s4 controller and back and then to h8. There is much longer routing than in the first example.

h1->s1->controller->s1->s5->controller->s5->s7->controller->s7->s6->controller->s6->s4->controller->s4->h8

Thus, longer time is taken in the second case due to longer routing.

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?

The results for both the cases are as follows
iperf h1 h2
Results: ['7.17 Mbits/sec', '8.04 Mbits/sec']
iperf h1 h8
Results: ['2.45 Mbits/sec', '2.93 Mbits/sec']

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['7.17 Mbits/sec', '8.04 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.45 Mbits/sec', '2.93 Mbits/sec']
```

Iperf is used to benchmark the bandwidth of a network.
Significant difference is observed between the two examples, iperf h1 h2 indicating higher bandwidth.
The bandwidth is lower for iperf h1 h8 because with TCP a response is needed while the packets travel a longer distance or can get lost in this example, resulting in lower bandwidth as compared to the first example with much lower routing.

Q.4 Which of the switches observe traffic? Please describe your way for observing such traffic on switches (e.g., adding some functions in the “of_tutorial” controller).

Add the following code to the controller

```
switch_name = "s"+tr(self.connection.dpid)

    log.debug("At Switch::"+switchname)
```

We can log the switch name logging the incoming and outcoming connections.

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).

The code adds the packets source MAC address to mac_to_port. Then it checks if the port of destination mac address is known. If it's known, we can witch and send that packet directly to that port. If it's unknown, then controller floods all outgoing ports with messages except the source port of the packet. When we ping h1 to h2 for the first time we will see traffic on all switches. However, when we ping for the second time, we only see the traffic on the first switch as the controllers checks for mac to port mapping and check if destinations MAC is already present. If it does, then direct it to h2 directly.

```
e6:9a:8e:91:16:f5 destination known. only send message to it
ff:ff:ff:ff:ff:ff not known, resend to everybody
ff:ff:ff:ff:ff:ff not known, resend to everybody
e2:d6:cb:87:11:3f destination known. only send message to it
ff:ff:ff:ff:ff:ff not known, resend to everybody
e6:9a:8e:91:16:f5 destination known. only send message to it
ff:ff:ff:ff:ff:ff not known, resend to everybody
ff:ff:ff:ff:ff:ff not known, resend to everybody
e2:d6:cb:87:11:3f destination known. only send message to it
e6:9a:8e:91:16:f5 destination known. only send message to it
e2:d6:cb:87:11:3f destination known. only send message to it
```

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)?

The results for h1 ping h2 100 times is as follows:

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99143ms
rtt min/avg/max/mdev = 1.588/25.843/50.395/14.562 ms
```

The results for h1 ping h8 100 times is as follows:

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99112ms
rtt min/avg/max/mdev = 6.901/34.208/62.315/14.849 ms
```

The difference between the two scenarios is ~8ms which is not very significant.

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

The result for iperf h1 h2 is as follows:

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['17.1 Mbits/sec', '18.2 Mbits/sec']
```

The result for iperf h1 h8 is as follows:

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['3.38 Mbits/sec', '3.93 Mbits/sec']
```

There is a difference of 9.93 Mbits/sec and 0.93 Mbits/sec from task 2. The throughput is higher for the mac learning controller. The bandwidth is more for the case with the first example as the mac to port mapping results in lower routing than the first case.

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 h2 takes on an average 0.063 ms

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99007ms
rtt min/avg/max/mdev = 0.055/0.063/0.091/0.009 ms
```

h1 ping h8 takes on an average 0.088 ms

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99000ms
rtt min/avg/max/mdev = 0.065/0.088/0.800/0.072 ms
```

The average ping time for task 4 using open flow rules much lower than the one in task 3.

Because in task 4 only the packets that switch doesn't have a flow entry for arrive at the controller. The flow rules on the switch helps handle future packet thus leading to lower ping time.

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

The results for iperf h1 h2 is 25.6 Gbits/sec

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['25.6 Gbits/sec', '25.6 Gbits/sec']
```

The result for iperf h1 h8 is 21.6 Gbits/sec

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['21.6 Gbits/sec', '21.5 Gbits/sec']
```

The difference is very huge (more than 21 Gbits/sec) as compared to task 3.

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

The bandwidth considerably improves for the scenario in task 4 with smart switching using OpenFlow rules compared to task 3.

Using OpenFlow rules the controller learns where the devices are in the topology and updates the flow tables to build end to end connectivity.

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

```
root@faf80035f469:~# mn --custom binary_tree.py  --topo mytopo --test pingall &> Pingall.txt
root@faf80035f469:~# cat Pingall.txt
*** 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 ...
*** Waiting for switches to connect
s1 s2 s3 s4 s5 s6 s7
*** 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)
*** Stopping 1 controllers
c0
*** Stopping 14 links
..............
*** Stopping 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Stopping 8 hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Done
completed in 11.154 seconds
root@faf80035f469:~#
```

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)?

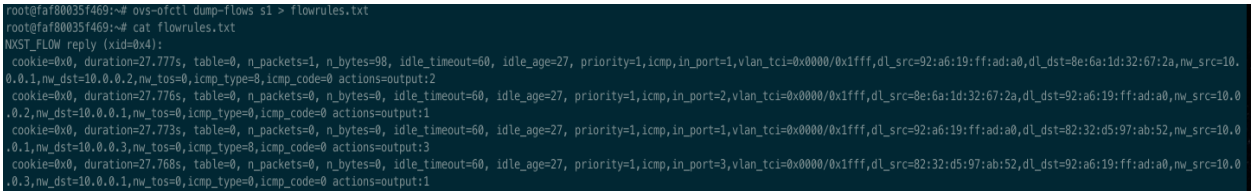
The output of the flow rules using “ovs-ofctl dump-flows s1” as flows

NXST_FLOW reply (xid=0x4):

cookie=0x0, duration=27.777s, table=0, n_packets=1, n_bytes=98, idle_timeout=60, idle_age=27, priority=1,icmp,in_port=1,vlan_tci=0x0000/0x1fff,dl_src=92:a6:19:ff:ad:a0,dl_dst=8e:6a:1d:32:67:2a,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2

cookie=0x0, duration=27.776s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, idle_age=27, priority=1,icmp,in_port=2,vlan_tci=0x0000/0x1fff,dl_src=8e:6a:1d:32:67:2a,dl_dst=92:a6:19:ff:ad:a0,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1

cookie=0x0, duration=27.773s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, idle_age=27, priority=1,icmp,in_port=1,vlan_tci=0x0000/0x1fff,dl_src=92:a6:19:ff:ad:a0,dl_dst=82:32:d5:97:ab:52,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:3



Task 5

Using ifconfig we can retrieve the ip address of hall the hosts. They range for 10.0.0.1 to 10.0.0.8.

We will add set of ovs-ofctl commands to add flows connection to the hosts on both sides of the S7 switch using their destination ip address.

```
sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.1 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.2 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.3 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.4 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.5 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.6 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.7 actions=normal"

sudo ovs-ofctl add-flow s7 "arp,nw_dst=10.0.0.8 actions=normal"
```

Bonus question

Bonus: You can get the bonus points, if you deploy OpenDaylightcontroller, and integrate it with Mininet. Please provide the proofs.

```
csvb@CS480k-sraut1:~/mininet$ cd distribution-karaf-0.4.0-Beryllium
csvb@CS480k-sraut1:~/mininet/distribution-karaf-0.4.0-Beryllium$ ./bin/karaf
karaf: JAVA_HOME not set; results may vary
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
/home/csvb/mininet/distribution-karaf-0.4.0-Beryllium/etc/config.properties (Permission denied)
csvb@CS480k-sraut1:~/mininet/distribution-karaf-0.4.0-Beryllium$ sudo ./bin/karaf
karaf: JAVA_HOME not set; results may vary
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0

    ________                        ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  | \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/           /_____/      \/


Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.
```