CS580K : Mini Project 3:

Name : Shubham Patwa

Go to:

Task 1

Task 2

Task 3

Task 4

Task 5

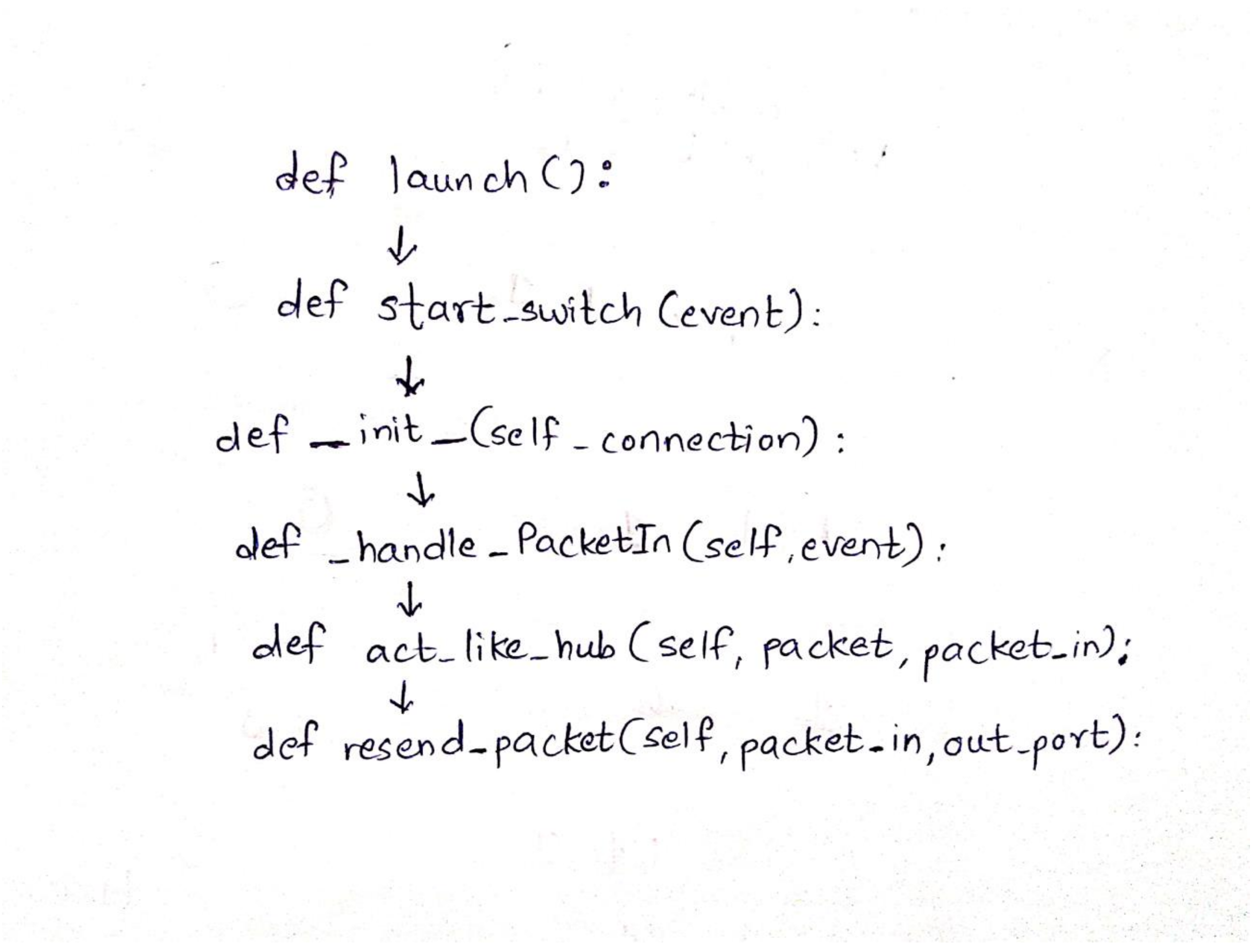
BonusTask

- Task 1:

A python script that generates a binary tree topology has been attached with the submission folder.

Task 2: Study the “of_tutorial” Controller

• 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?

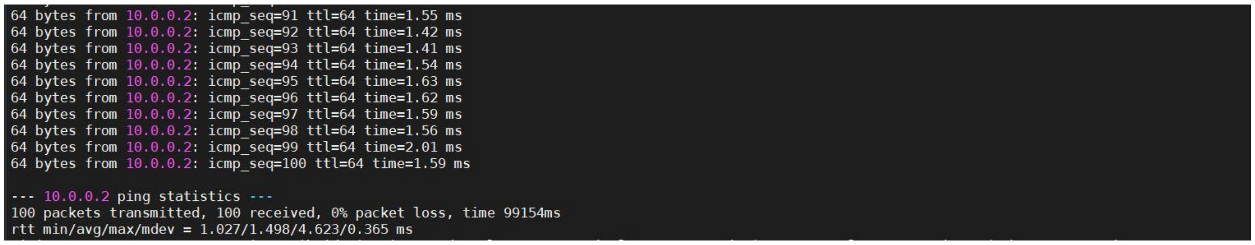

def launch():
def start_switch(event):
def __init__(self_connection):
def _handle_PacketIn(self,event):
def act_like_hub(self, packet, packet_in):
def resend_packet(self, packet_in, out_port):

After this, until mininet is exited , _handle_PacketIn is looped.

• Q.2 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?

Case : h1 ping -c100 h2:

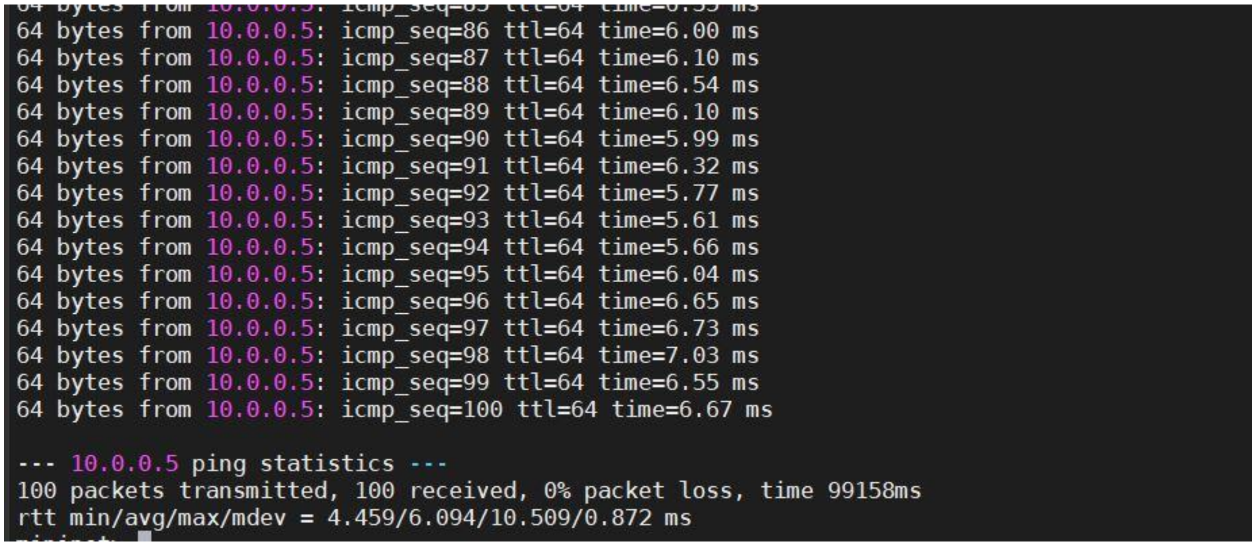
Average 1.498ms



```
64 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=1.55 ms
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=1.42 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=1.41 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=1.54 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=1.63 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=1.62 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=1.59 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=1.56 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=2.01 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=1.59 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99154ms
rtt min/avg/max/mdev = 1.027/1.498/4.623/0.365 ms
```

Case : h1 ping -c100 h5

Average : 6.094 ms



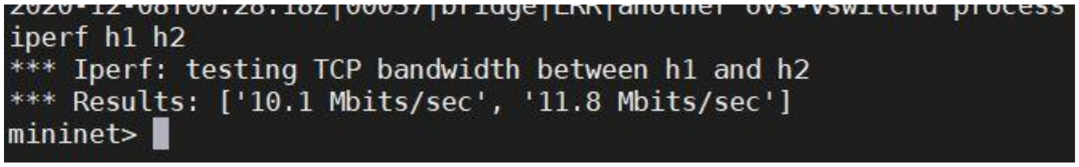
```
64 bytes from 10.0.0.5: icmp_seq=86 ttl=64 time=6.00 ms
64 bytes from 10.0.0.5: icmp_seq=87 ttl=64 time=6.10 ms
64 bytes from 10.0.0.5: icmp_seq=88 ttl=64 time=6.54 ms
64 bytes from 10.0.0.5: icmp_seq=89 ttl=64 time=6.10 ms
64 bytes from 10.0.0.5: icmp_seq=90 ttl=64 time=5.99 ms
64 bytes from 10.0.0.5: icmp_seq=91 ttl=64 time=6.32 ms
64 bytes from 10.0.0.5: icmp_seq=92 ttl=64 time=5.77 ms
64 bytes from 10.0.0.5: icmp_seq=93 ttl=64 time=5.61 ms
64 bytes from 10.0.0.5: icmp_seq=94 ttl=64 time=5.66 ms
64 bytes from 10.0.0.5: icmp_seq=95 ttl=64 time=6.04 ms
64 bytes from 10.0.0.5: icmp_seq=96 ttl=64 time=6.65 ms
64 bytes from 10.0.0.5: icmp_seq=97 ttl=64 time=6.73 ms
64 bytes from 10.0.0.5: icmp_seq=98 ttl=64 time=7.03 ms
64 bytes from 10.0.0.5: icmp_seq=99 ttl=64 time=6.55 ms
64 bytes from 10.0.0.5: icmp_seq=100 ttl=64 time=6.67 ms

--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99158ms
rtt min/avg/max/mdev = 4.459/6.094/10.509/0.872 ms
```

Reason for this difference:

In our defined topology, h5 is away from h1 compared to h2. Since, it has to travel taking multiple hops/jumps as compared with h2, it takes more time.

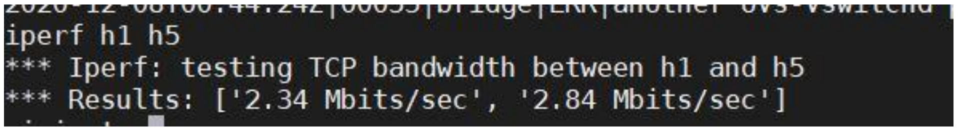
• Q.3 Run “iperf h1 h2” and “iperf h1 h5”. What is “iperf” used for? What is the throughput for each case? What is the difference, and why?

Case : iperf h1 h2



```
2020-12-08T00:28.182|00037|bridge|ERR|another ovs-vswitchd process
iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.1 Mbits/sec', '11.8 Mbits/sec']
mininet>
```

Case : iperf h1 h5



```
2020-12-08T00:44.242|00035|bridge|ERR|another ovs-vswitchd
iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['2.34 Mbits/sec', '2.84 Mbits/sec']
```

Iperf is used for testing the bandwidth.

Google**: Iperf** is a widely **used** tool for network performance measurement and tuning. It is significant as a cross-platform tool that can produce standardized performance measurements for any network.

Reason for the difference:

As previously stated, h5 is located away compared to h2 in the topology. Since it has to travel more than h2, we get reduced bandwidth.

• Q.4 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)

All switches observe traffic from s1 ~ s6.

We can add print functions in between to observe the traffic. Using multiple terminals, mininet in one terminal and then pinging the same from another terminal , we get and can observe the traffic.

Task 3: Mac Learning Controller

The python code has been attached in the submission folder.

• 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).

Packet is forwarded to the controller by the switch.

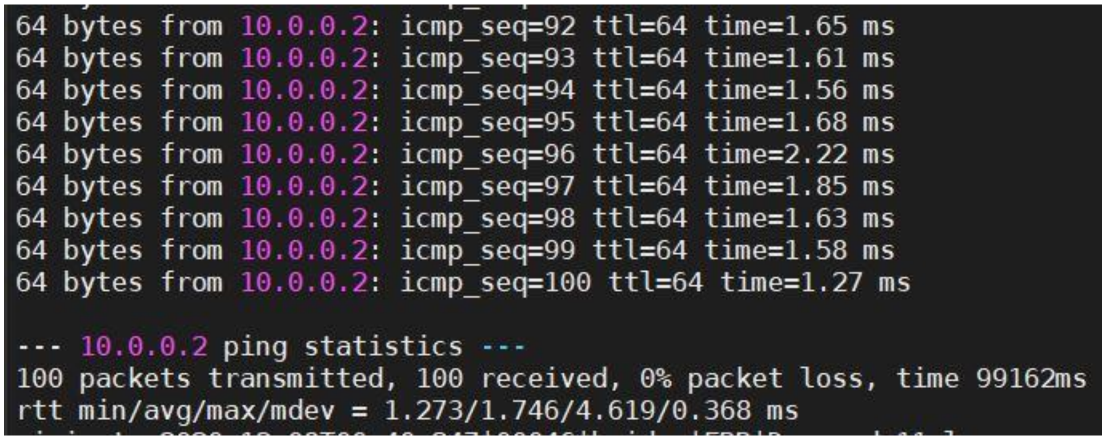
Controller performs the following steps:

```
If( host sends the first packet) :
MAC address and source port is learnt and saved.
Then, the port number gets attached to MAC address. Packet is sent through that 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 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)?

Case 1: h1 ping -c100 h2

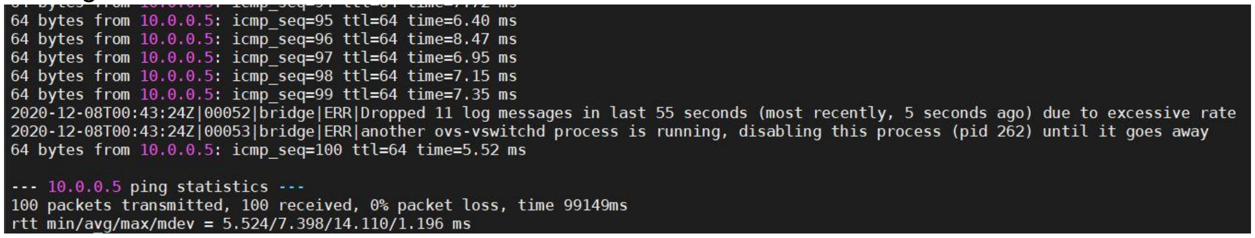
Average : 1.746 ms



```
64 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=1.65 ms
64 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=1.61 ms
64 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=1.56 ms
64 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=1.68 ms
64 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=2.22 ms
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=1.85 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=1.63 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=1.58 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=1.27 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99162ms
rtt min/avg/max/mdev = 1.273/1.746/4.619/0.368 ms
```

Case 2: h1 ping -c100 h5

Average : 7.398ms



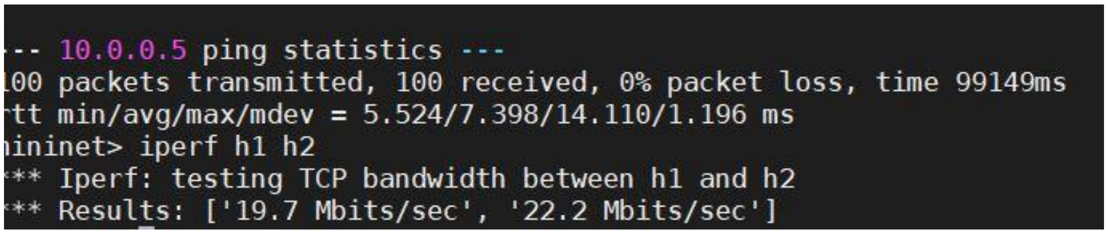
```
64 bytes from 10.0.0.5: icmp_seq=95 ttl=64 time=6.40 ms
64 bytes from 10.0.0.5: icmp_seq=96 ttl=64 time=8.47 ms
64 bytes from 10.0.0.5: icmp_seq=97 ttl=64 time=6.95 ms
64 bytes from 10.0.0.5: icmp_seq=98 ttl=64 time=7.15 ms
64 bytes from 10.0.0.5: icmp_seq=99 ttl=64 time=7.35 ms
2020-12-08T00:43:24Z|00052|bridge|ERR|Dropped 11 log messages in last 55 seconds (most recently, 5 seconds ago) due to excessive rate
2020-12-08T00:43:24Z|00053|bridge|ERR|another ovs-vswitchd process is running, disabling this process (pid 262) until it goes away
64 bytes from 10.0.0.5: icmp_seq=100 ttl=64 time=5.52 ms

--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99149ms
rtt min/avg/max/mdev = 5.524/7.398/14.110/1.196 ms
```

Difference compared to task 2: Average time has increased compared to task 2

• Q.3 Run “iperf h1 h2” and “iperf h1 h5”. What is the throughput for each case? What is the difference from Task II?

Case 1: iperf h1 h2:

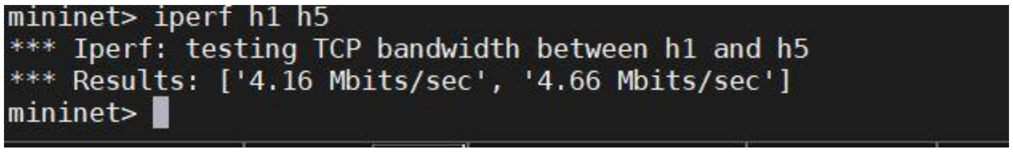
Throughput:



```
-- 10.0.0.5 ping statistics ---
00 packets transmitted, 100 received, 0% packet loss, time 99149ms
tt min/avg/max/mdev = 5.524/7.398/14.110/1.196 ms
ininet> iperf h1 h2
** Iperf: testing TCP bandwidth between h1 and h2
** Results: ['19.7 Mbits/sec', '22.2 Mbits/sec']
```

Case 2: iperf h1 h5

Throughput:



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['4.16 Mbits/sec', '4.66 Mbits/sec']
mininet>
```

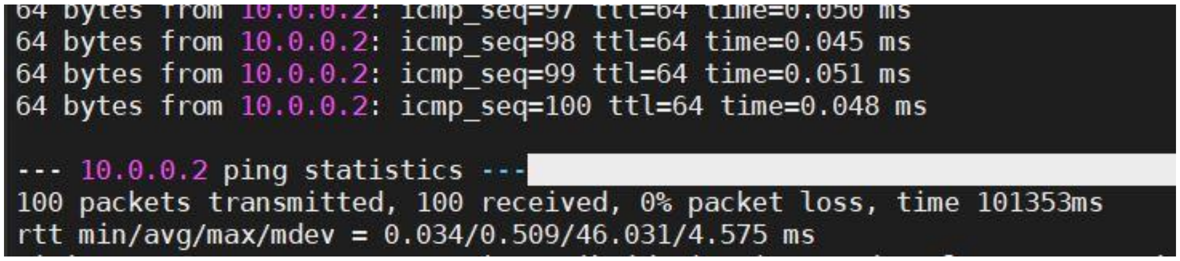
Difference form task 2 : As we can see, the throughput in both cases has increased as compared to task 2.

Task 4: Mac Learning Controller with OpenFlow Rules

• Q.1 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? Any difference from Task III (the MAC case without inserting flow rules)?

Case 1: h1 ping -c100 h2

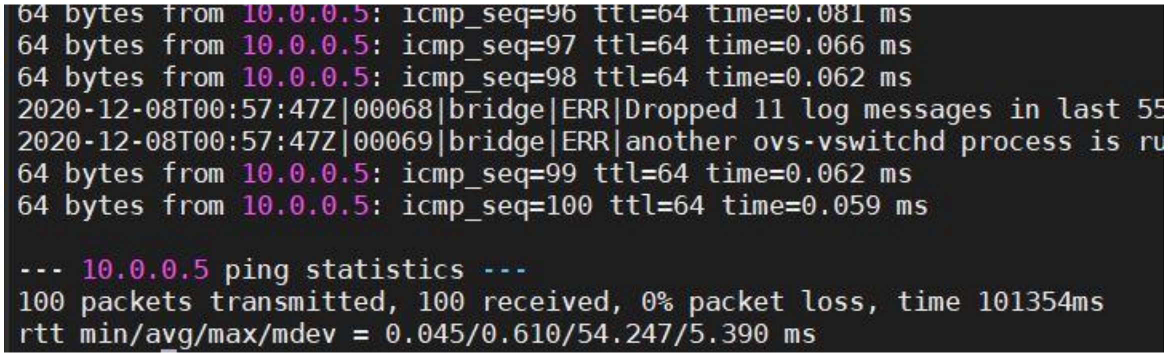
Average : 0.509ms



```
64 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=0.048 ms

--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101353ms
rtt min/avg/max/mdev = 0.034/0.509/46.031/4.575 ms
```

Case 2: h1 ping -c100 h5

Average : 0.610ms



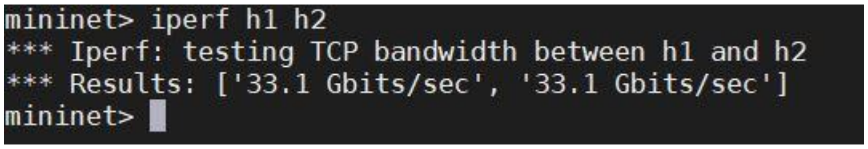
```
64 bytes from 10.0.0.5: icmp_seq=96 ttl=64 time=0.081 ms
64 bytes from 10.0.0.5: icmp_seq=97 ttl=64 time=0.066 ms
64 bytes from 10.0.0.5: icmp_seq=98 ttl=64 time=0.062 ms
2020-12-08T00:57:47Z|00068|bridge|ERR|Dropped 11 log messages in last 55
2020-12-08T00:57:47Z|00069|bridge|ERR|another ovs-vswitchd process is ru
64 bytes from 10.0.0.5: icmp_seq=99 ttl=64 time=0.062 ms
64 bytes from 10.0.0.5: icmp_seq=100 ttl=64 time=0.059 ms

--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101354ms
rtt min/avg/max/mdev = 0.045/0.610/54.247/5.390 ms
```

As compared with Task 3: We can say that there is a tremendous improvement over the ping times for both the cases with openFlow rules

Q.2 Run “iperf h1 h2” and “iperf h1 h5”. What is the throughput for each case? What is the difference from Task III?

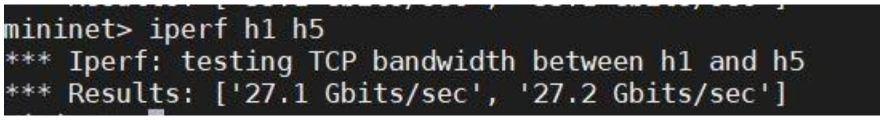
Case 1: iperf h1 h2

Throughput:



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['33.1 Gbits/sec', '33.1 Gbits/sec']
mininet>
```

Case 2: iperf h1 h5



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['27.1 Gbits/sec', '27.2 Gbits/sec']
```

Difference from Task 3: As we can see, there is again a tremendous improvement over the bandwidth compared with Task 3.

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

Explanation:

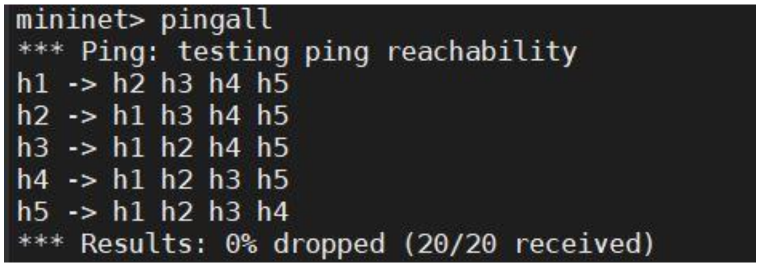
The results have become much better.

Because of the openFlow established rules, the topology switches immediately know where and how to transmit that packet

Thus there is drastic improvement over previous performance.

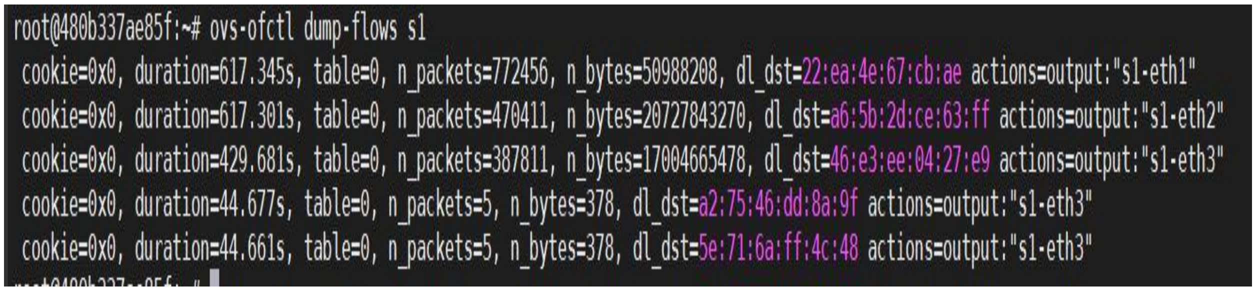
We can notice this in the earlier images as proofs.

• Q.4 Run pingall to verify connectivity and dump 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)
```

• 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)?



```
root@480b337ae85f:~# ovs-ofctl dump-flows s1
 cookie=0x0, duration=617.345s, table=0, n_packets=772456, n_bytes=50988208, dl_dst=22:ea:4e:67:cb:ae actions=output:"s1-eth1"
 cookie=0x0, duration=617.301s, table=0, n_packets=470411, n_bytes=20727843270, dl_dst=a6:5b:2d:ce:63:ff actions=output:"s1-eth2"
 cookie=0x0, duration=429.681s, table=0, n_packets=387811, n_bytes=17004665478, dl_dst=46:e3:ee:04:27:e9 actions=output:"s1-eth3"
 cookie=0x0, duration=44.677s, table=0, n_packets=5, n_bytes=378, dl_dst=a2:75:46:dd:8a:9f actions=output:"s1-eth3"
 cookie=0x0, duration=44.661s, table=0, n_packets=5, n_bytes=378, dl_dst=5e:71:6a:ff:4c:48 actions=output:"s1-eth3"
```

We can see that there are 5 entries here in the below diagram. Thus, there are 5 rules for each openFlow switch.

A flow rule is what drives a packet.

Arrival of a packet at a host for the first time configures the flow rules for all future packets for that switch.

Since there are five host here, we can conclude that there are 5 flow rules for each switch.

Task 5: Layer 3 Routing

We first install IP-matching rules on switch -6 and let other switches stay as MAC learning switches.

The goal is to allow all hosts from h1 to h4 ping h5 ad switch forwards packets via IP-matching flow rules and not MAC learning rules.

We can achieve this by editing the act like switch function using an if else condition since that function handles all the switches.

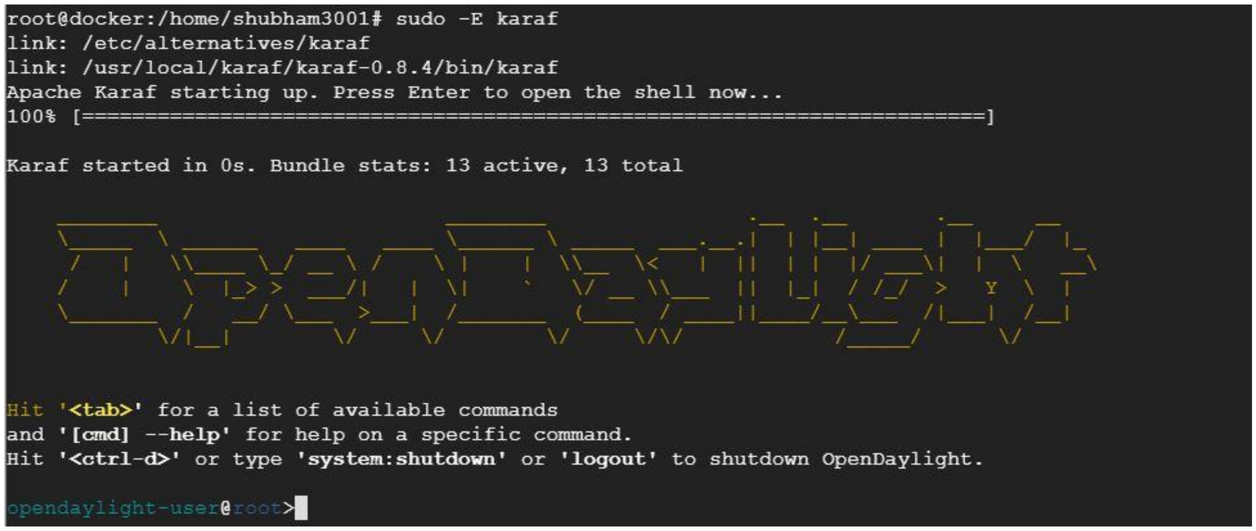
Using ovs-ofctl we can achieve the required functionality.

Bonus: Deploy OpenDaylight controller, and integrate it with Mininet. Please provide the proofs.

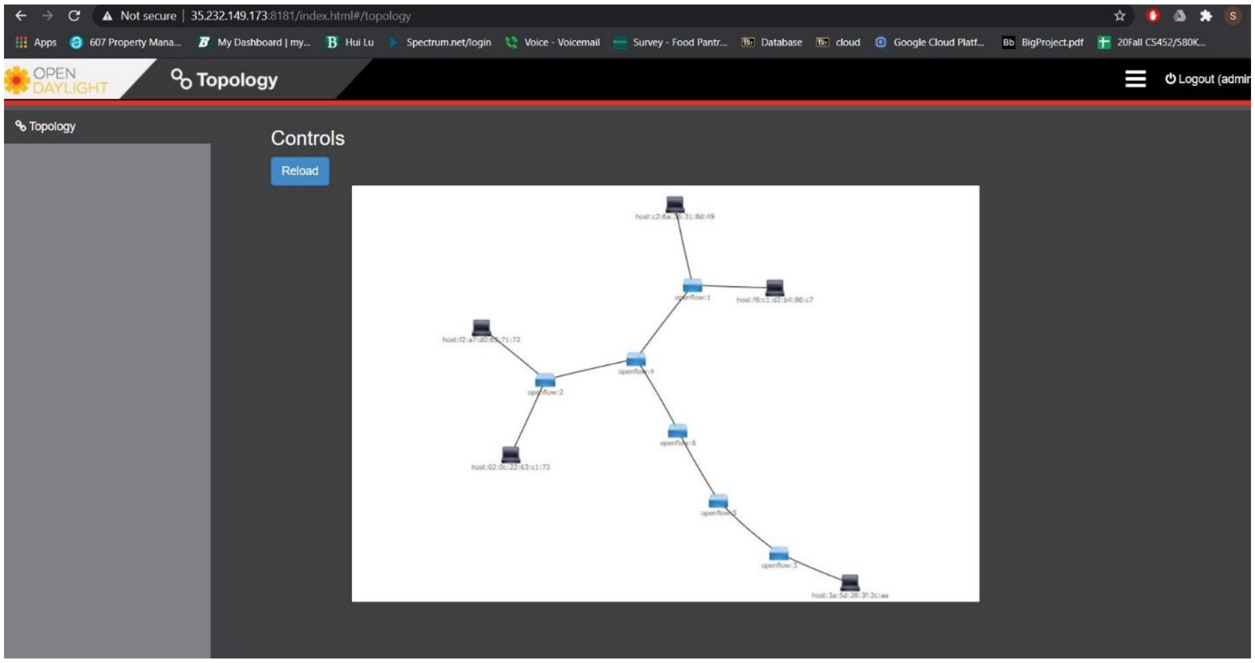
Reference: https://john.soban.ski/how-to-install-opendaylight-as-a-service-on-ubuntu.html

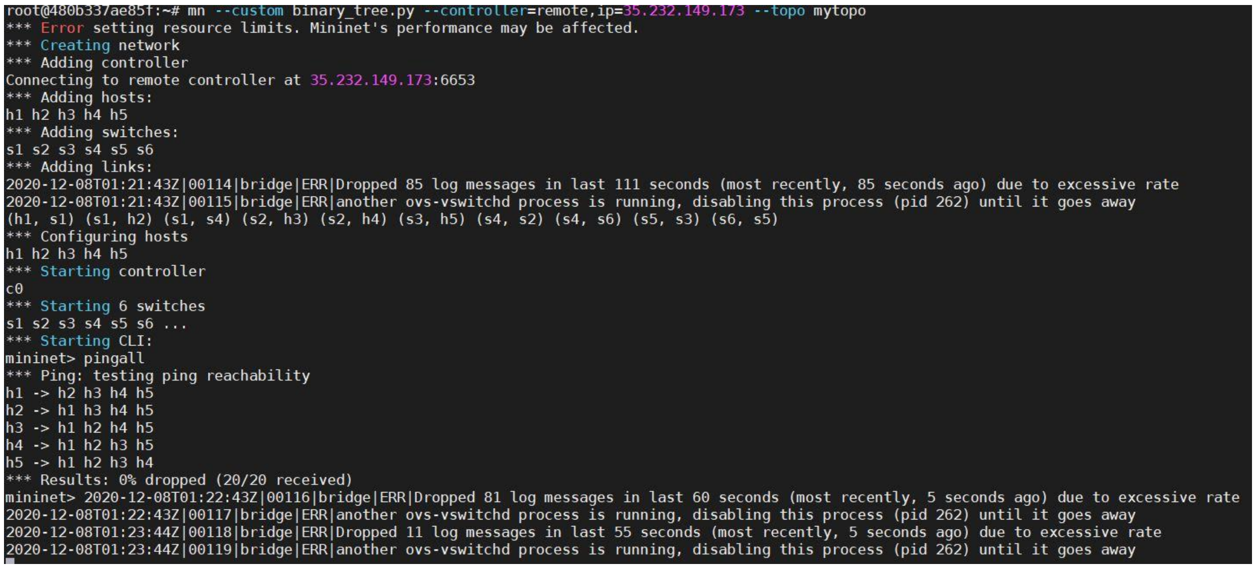
Used GCP for this task. Network Firewall Rules were easy to configure.

Here are the attached proofs:



```
root@docker:/home/shubham3001# sudo -E karaf
link: /etc/alternatives/karaf
link: /usr/local/karaf/karaf-0.8.4/bin/karaf
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]

Karaf started in 0s. Bundle stats: 13 active, 13 total

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

opendaylight-user@root>
```





```
root@480b337ae85f:~# mn --custom binary_tree.py --controller=remote,ip=35.232.149.173 --topo mytopo
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
Connecting to remote controller at 35.232.149.173:6653
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
2020-12-08T01:21:43Z|00114|bridge|ERR|Dropped 85 log messages in last 111 seconds (most recently, 85 seconds ago) due to excessive rate
2020-12-08T01:21:43Z|00115|bridge|ERR|another ovs-vswitchd process is running, disabling this process (pid 262) until it goes away
(h1, s1) (s1, h2) (s1, s4) (s2, h3) (s2, h4) (s3, h5) (s4, s2) (s4, s6) (s5, s3) (s6, s5)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
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> 2020-12-08T01:22:43Z|00116|bridge|ERR|Dropped 81 log messages in last 60 seconds (most recently, 5 seconds ago) due to excessive rate
2020-12-08T01:22:43Z|00117|bridge|ERR|another ovs-vswitchd process is running, disabling this process (pid 262) until it goes away
2020-12-08T01:23:44Z|00118|bridge|ERR|Dropped 11 log messages in last 55 seconds (most recently, 5 seconds ago) due to excessive rate
2020-12-08T01:23:44Z|00119|bridge|ERR|another ovs-vswitchd process is running, disabling this process (pid 262) until it goes away
```