CS580K: Adv. Topics In Cloud Computing

Mini Project-3

Name : Prathamesh N Lonkar

Bnumber: B00811727

Task I: Define a Custom Topology

In this task we are asked to create a binary tree topology which resembles the following tree sample:

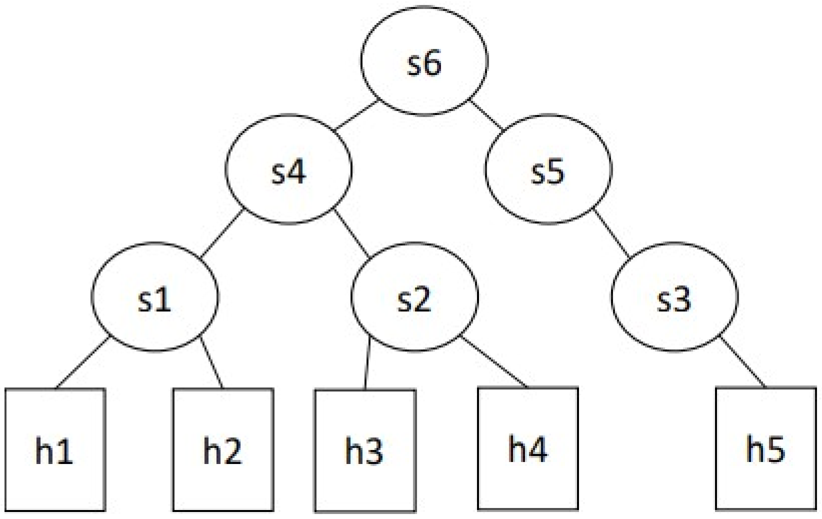
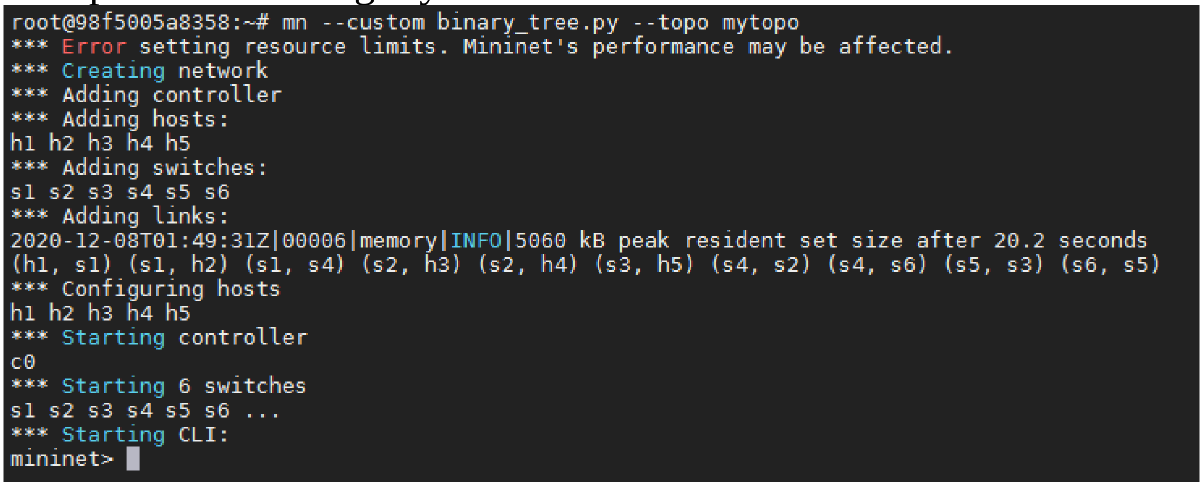


Figure 1: A binary tree network of depth 3.

The code to create this topology was referred from the following link:
http://mininet.org/walkthrough/
which was provided by professor. I have started building the tree from the host1 i.e. 'h1' and build my way round the tree till host5 i.e 'h5'.
I have done this by using the inbuilt functions provided by the Topo library which are self.addLink(node1,node2).
You can check out the code for the above provided in the file with the name 'binary_tree.py'.

The output after running my code was as follows:



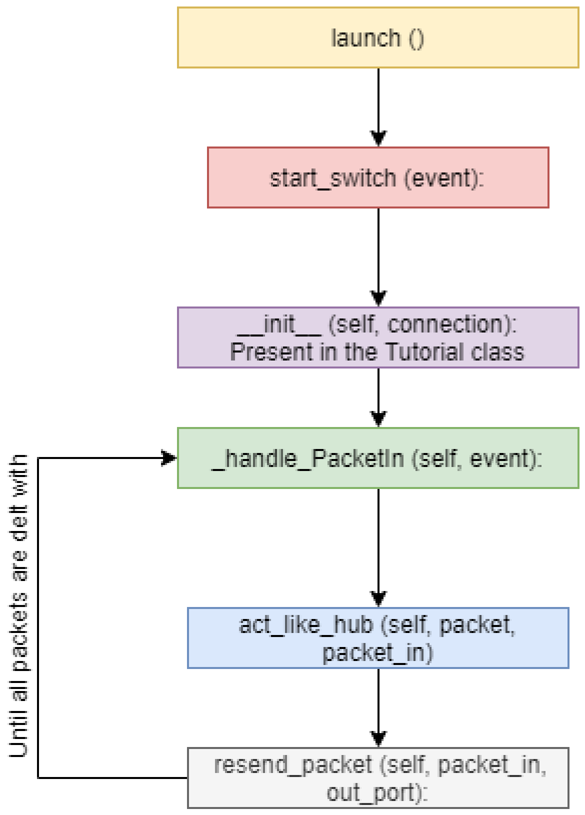
```
root@98f5005a8358:~# mn --custom binary_tree.py --topo mytopo
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
2020-12-08T01:49:31Z|00006|memory|INFO|5060 kB peak resident set size after 20.2 seconds
(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>
```

Task II: 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?

Answer:

The data flows as follows:


launch ()
start_switch (event):
__init__ (self, connection):
Present in the Tutorial class
_handle_PacketIn (self, event):
act_like_hub (self, packet, packet_in)
resend_packet (self, packet_in, out_port):
Until all packets are delt with

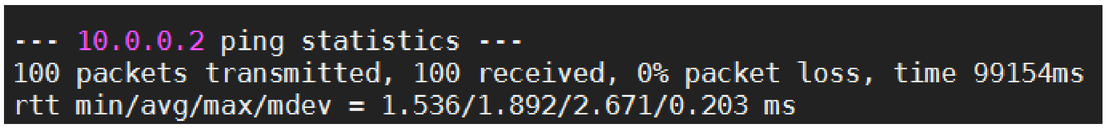
- The above graph shows that first the packets will enter the launch function.
- Then they will enter the start_switch function.
- Then the init function will be called where the connection will be established which is present in the Turorial class.
- The the handle_packetsin function is called where it is checked if the packet is empty or not.
- Then the control goes to act_like_hub function which will behave like a hub and send packets to all ports beside the input port.
- At last the resend_packet is called where it will instructs the switch to resend a packet that it had sent.

- The control will be transfered to the handle-packetsIn until all the packets are delt with

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?

Answer:

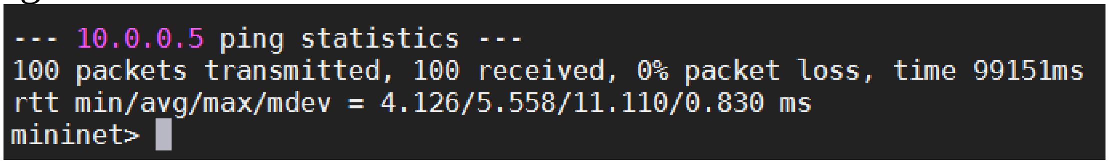
The output after the ping is as follows:
h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99154ms
rtt min/avg/max/mdev = 1.536/1.892/2.671/0.203 ms
```

Here as we can see the average time is 1.892ms and the total time taken is 99154ms.

h1 ping -c100 h5:



```
--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 4.126/5.558/11.110/0.830 ms
mininet> 
```

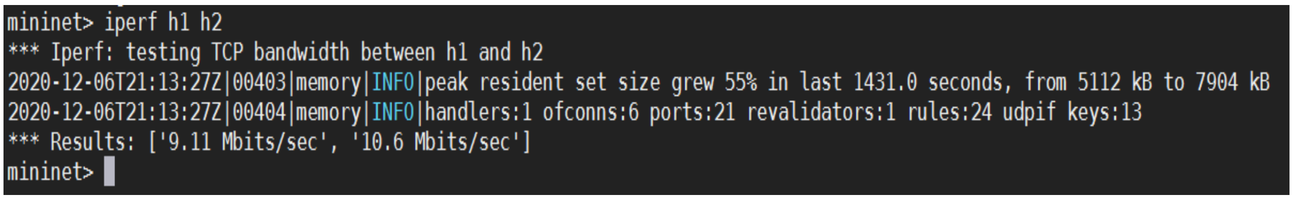
Here we can see that the average time taken is 5.558ms and the total time is 99151.

We can see that there is a substantial difference between the average time taken, this is because the nodes h1 and h2 are closer to each other that the nodes h1 and h5 as seen in the above graph, hence it takes more time for the packet to be transferred from h1 to h5 tha it takes to transfer from h1 to h2.

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?

Answer:
The output after performing iperf is as follows:

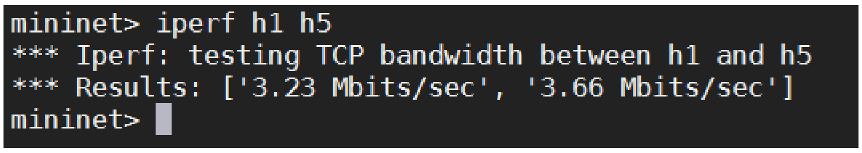
iperf h1 h2:



```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
2020-12-06T21:13:27Z|00403|memory|INFO|peak resident set size grew 55% in last 1431.0 seconds, from 5112 kB to 7904 kB
2020-12-06T21:13:27Z|00404|memory|INFO|handlers:1 ofconns:6 ports:21 revalidators:1 rules:24 udpif keys:13
*** Results: ['9.11 Mbits/sec', '10.6 Mbits/sec']
mininet> 
```

Here we can see that after the test the the results show that the throughput between h1 and h2 is 9.11Mbits/sec.

Iperf h1 h5:



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['3.23 Mbits/sec', '3.66 Mbits/sec']
mininet> 
```

After performing the iperf on h1 to h5 node we can see that there is a decrease in the throughput which in this case is 3.23 Mbits/sec.

There is a decrease in the throughput as the nodes h1 and h2 are close by and only have to switch between single link and switch which is switch 1, but for the data packets to reach h5 there are many switch the packets has to be passed to which decreases the throughput.

• 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).

Answer:

Basically if we pass the packet from h1 to h5 then all the switches will observe traffic, and even if this is not the case still all the switches will observe traffic as the resend_packet function in of_tutorial will resend the packets to all the switches.

I tried to observe the traffic by inserting some print statements in the of_tutorial code but after running the mininet there were some garbage values printed on the console, and it seemed to be running in an infinite loop.
Hence I did’nt do this part.

Task III: MAC learning controller

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).

After adding the code provided in the documentation we will have to comment the act_like_hub as it send the packets to all the ports which we do not want. And we will have to call our added function instead which is done in the _handle_PacketIn function.

After the packet is passed to the function act_like_switch, in the function we check if the packet src is present in the mac_to_port. As at first our mac_to_port is initialized as empty we will enter the if condition and set the port in the mac_to_port object.

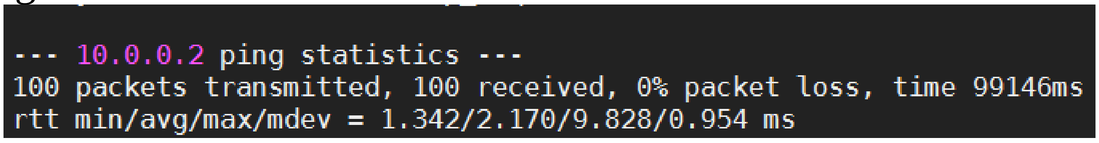
When the next packet arrives at the act_like_switch function now we have the port in the mac_to_port hence we will only send the packet to that port and no one else.

If the above two conditions are not satisfied then we will send the packets to all the ports which is done in the lase else condition in act_like_hub.

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)?

Answer:

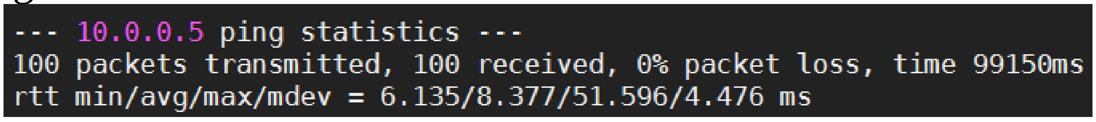
The output after performin the ping is as follows:
h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99146ms
rtt min/avg/max/mdev = 1.342/2.170/9.828/0.954 ms
```

Here we can see that the average time is 2.170ms when we perform a ping from h1 to h2.

h1 ping -c100 h5:



```
--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99150ms
rtt min/avg/max/mdev = 6.135/8.377/51.596/4.476 ms
```

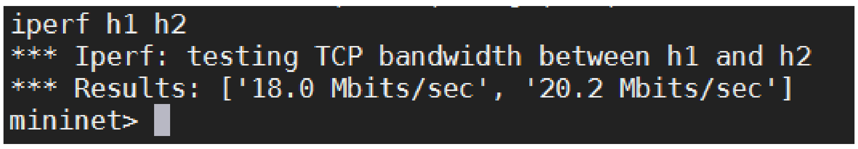
Here we can see that the average time is 8.377ms after we perform ping from h1 to h5.

We can see that there is a small increase in the average time of packet transfer than that of Task II.

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

Answer:
The output after performing the iperf is as follows:

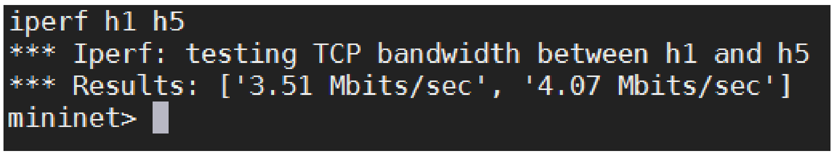
iperf h1 h2:



```
iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['18.0 Mbits/sec', '20.2 Mbits/sec']
mininet>
```

Here we can see that the throughput is 18.0 Mbits/sec after performing the iperf from h1 to h2.

iperf h1 h5;



```
iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['3.51 Mbits/sec', '4.07 Mbits/sec']
mininet>
```

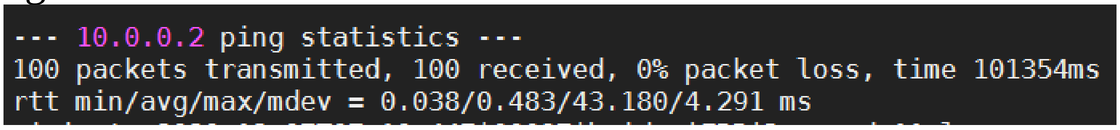
Here we can see that the throughput is 3.51 Mbits/sec after performing the iperf from h1 to h5.

We can see that there is a substantial increase in the throughput from h1 to h2 which is almost double of that in task II and there is also increase in the throughput from h1 to h5.

Task IV: 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)?
Answer:

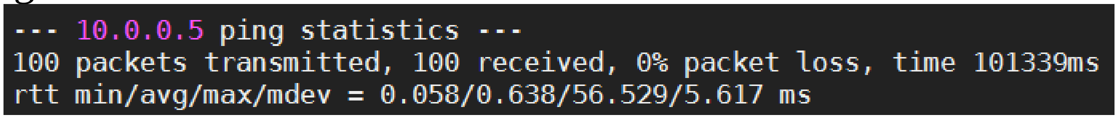
The output after performing the ping is as follows:
h1 ping -c100 h2:



```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101354ms
rtt min/avg/max/mdev = 0.038/0.483/43.180/4.291 ms
```

Here we can see that the average time to transfer packets is 0.483ms after performing the ping.

h1ping -c100 h5:



```
--- 10.0.0.5 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101339ms
rtt min/avg/max/mdev = 0.058/0.638/56.529/5.617 ms
```

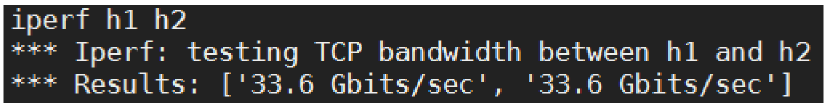
Here we can see that the average time to transfer packets is 0.638ms after performing the ping.

We can see that there is a substantial decrease in the rate of tranfer of packets that we observed in the Task III.

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

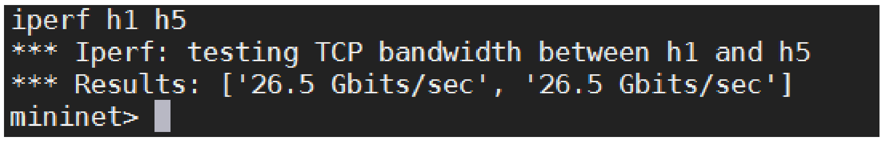
Answer:
The output after performing the iperf is as follows:

iperf h1 h2



```
iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['33.6 Gbits/sec', '33.6 Gbits/sec']
```

Here we can see that the throughput is 33.6 Gbits/sec between h1 and h2 iperf h1 h5;



```
iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['26.5 Gbits/sec', '26.5 Gbits/sec']
mininet>
```

Here we can see that the throughput is 26.5 Gbits/sec between h1 and h5

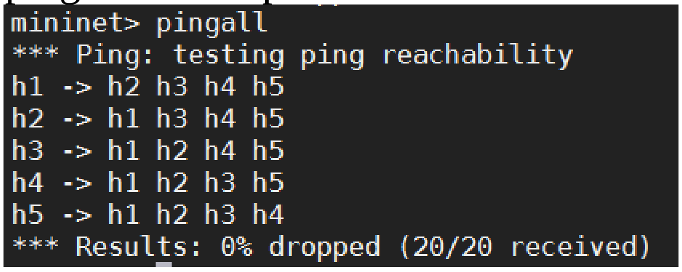
We can observe that there is a very large increase in the bandwidth than the one we observed in Task III. The increase is double than that observed in Task III.

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

The above difference is observed because we have added the code to our of_tutorial which will add the flow rules to our switches which in tun will make our switches a little smarter and they will be able to handle future packets. After adding the code, when a packet will arrive at the act_like_switch function the switches will now know what they have to do with the packets and will directly send those packets to their destination. Hence the packets are transfered faster.

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

After running the ping all the output is as follows:

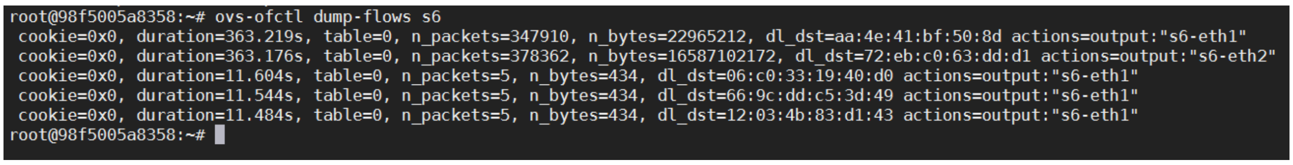


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

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)?

Answer:

The output after dumping the flow rules for switch6:



```
root@98f5005a8358:~# ovs-ofctl dump-flows s6
 cookie=0x0, duration=363.219s, table=0, n_packets=347910, n_bytes=22965212, dl_dst=aa:4e:41:bf:50:8d actions=output:"s6-eth1"
 cookie=0x0, duration=363.176s, table=0, n_packets=378362, n_bytes=16587102172, dl_dst=72:eb:c0:63:dd:d1 actions=output:"s6-eth2"
 cookie=0x0, duration=11.604s, table=0, n_packets=5, n_bytes=434, dl_dst=06:c0:33:19:40:d0 actions=output:"s6-eth1"
 cookie=0x0, duration=11.544s, table=0, n_packets=5, n_bytes=434, dl_dst=66:9c:dd:c5:3d:49 actions=output:"s6-eth1"
 cookie=0x0, duration=11.484s, table=0, n_packets=5, n_bytes=434, dl_dst=12:03:4b:83:d1:43 actions=output:"s6-eth1"
root@98f5005a8358:~#
```

The above output defines 5 flow rules which refer to every host available and the flow rules to reach them.

I have displayed the flow rules for the switch 6 which tells us that if a packe is arrived then it is transfered by the output port s6-eth1 or s6-eth2.

Task V: Layer-3 routing

Here I could see after performing ifconfig on every host that the ip ranges from 10.0.0.1 to 10.0.0.5 for each host.

But when I tried to install the ip-matching rules on switch s6, i was not able to do so. Hence I have not finished this task.

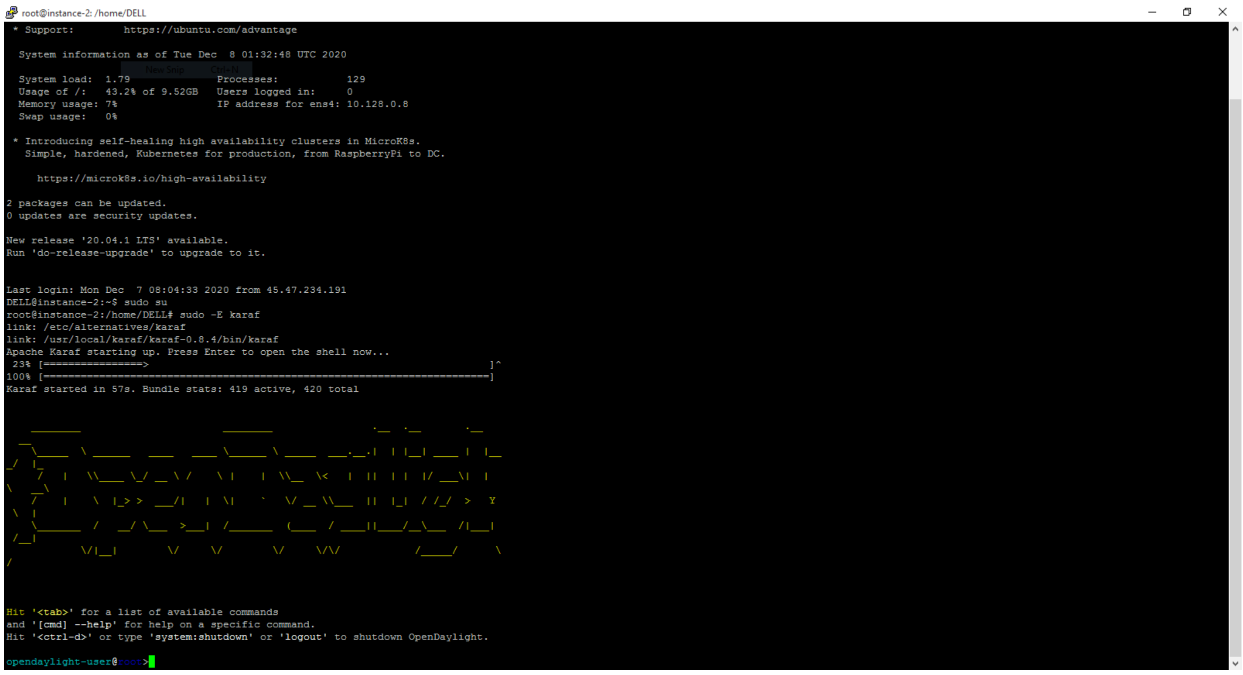
But I have Implemented the bonus task, which is shown below.

Bonus Task: OpenDaylight

To perform this task there are several steps, which are mentioned below:

Step 1: Run OpenDaylight On a Vm

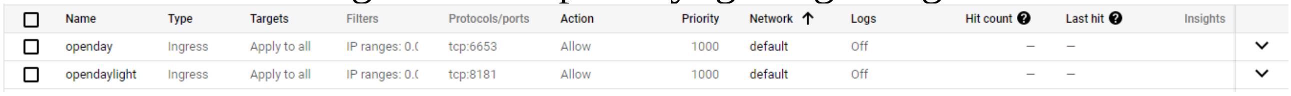
- Firstly we have to install the OpenDaylight as a service on any vm of your choice.
- I tried doing it on our college VM, but was unable to connect mininet to it.
- Hence I have Installed the OpenDaylight on the Google Cloud Instance as shown below:



```
root@instance-2: /home/DELL
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Dec  8 01:32:48 UTC 2020

  System load:  1.79              Processes:           129
  Usage of /:   43.2% of 9.52GB   Users logged in:     0
  Memory usage: 7%                IP address for ens4: 10.128.0.8
  Swap usage:   0%

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

     https://microk8s.io/high-availability

2 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Mon Dec  7 08:04:33 2020 from 45.47.234.191
DELL@instance-2:~$ sudo su
root@instance-2:/home/DELL# 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...
 23% [================>                                                      ]^
100% [========================================================================]
Karaf started in 57s. Bundle stats: 419 active, 420 total



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>
```

Step 2: Add Firewall Rules to give access to mininet

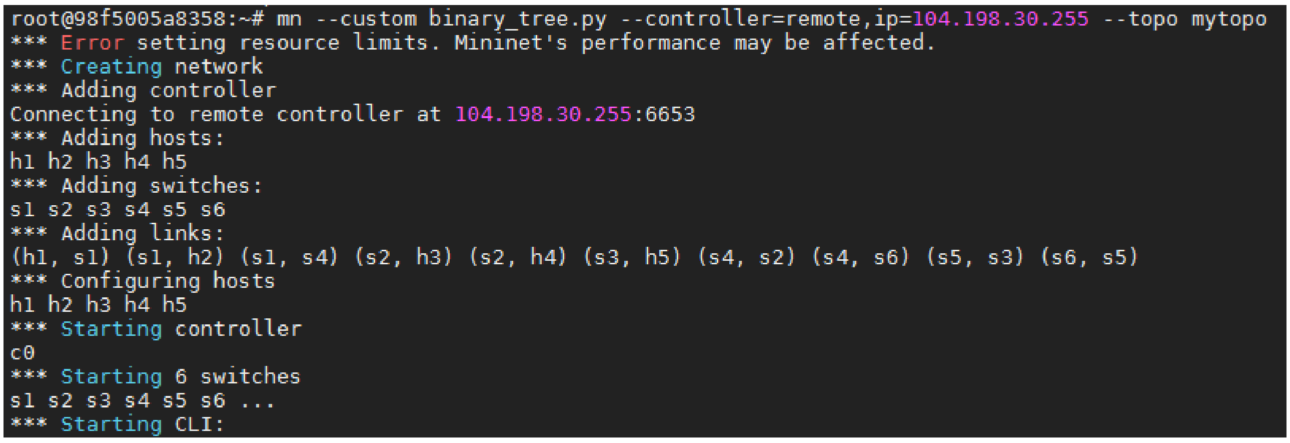
- I have added the following two firewall rules to my Vm instance
- First is for the mininet to connect to VM by using External Ip address
- Second is to log into our OpenDaylight login Page



☐	Name	Type	Targets	Filters	Protocols/ports	Action	Priority	Network ↑	Logs	Hit count	Last hit	Insights
☐	openday	Ingress	Apply to all	IP ranges: 0.(	tcp:6653	Allow	1000	default	Off	–	–	
☐	opendaylight	Ingress	Apply to all	IP ranges: 0.(	tcp:8181	Allow	1000	default	Off	–	–	

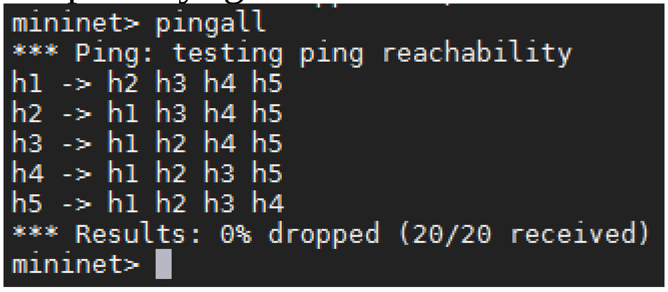
Step 3: Start the Mininet

- I have started the mininet by giving it the external ip address of my GCP Vm.
- The Output below shows how to start the Mininet



```
root@98f5005a8358:~# mn --custom binary_tree.py --controller=remote,ip=104.198.30.255 --topo mytopo
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
Connecting to remote controller at 104.198.30.255:6653
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(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:
```

- Here my VM's External Ip is 104.198.30.255
- After this we also perform a pingall operation in mininet as shown below, for our OpenDaylight to construct a flow diagram



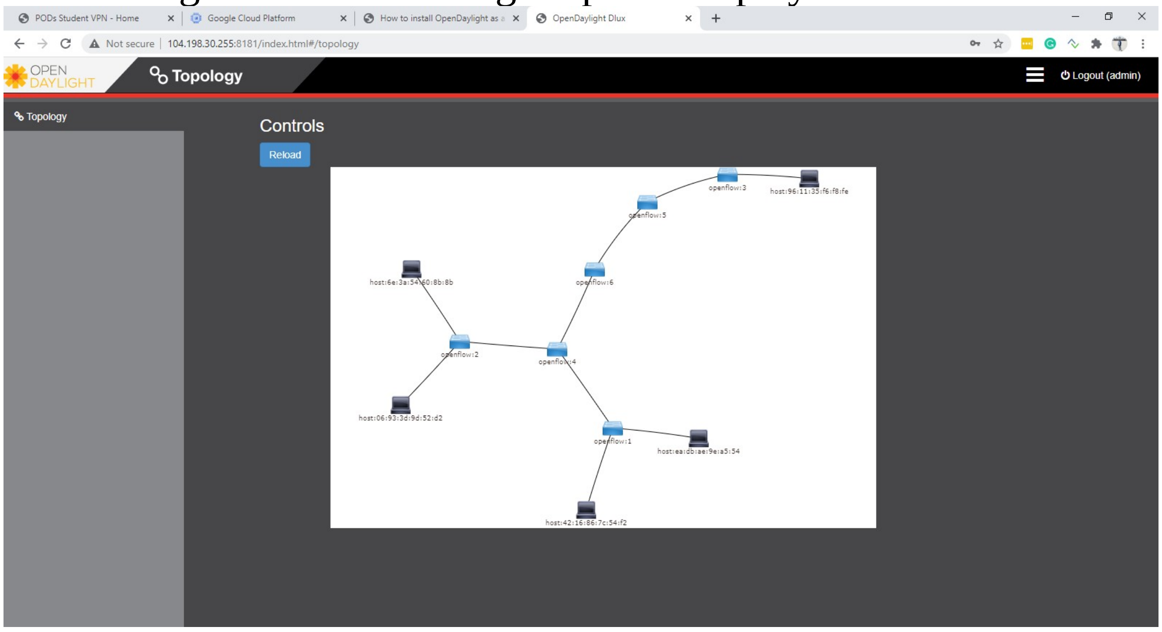
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> 
```

Step 4: Login into OpenDaylight

- This is done using the following link:

http://104.198.30.255:8181/index.html#/login

- This ip wont be working now as i have closed my vm.
- The 2nd firewall rule wass added so that the above link can work on port 8181.
- After login in the following output is displayed:



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

Documents attached in folder:

- miniproject3 document
- binary_tree.py (code to buid topology)
- of_tutorial_3.py (code for task 3)
- of_tutorial_4.py(code for task 4)