CS 580K Advanced Topics in Cloud

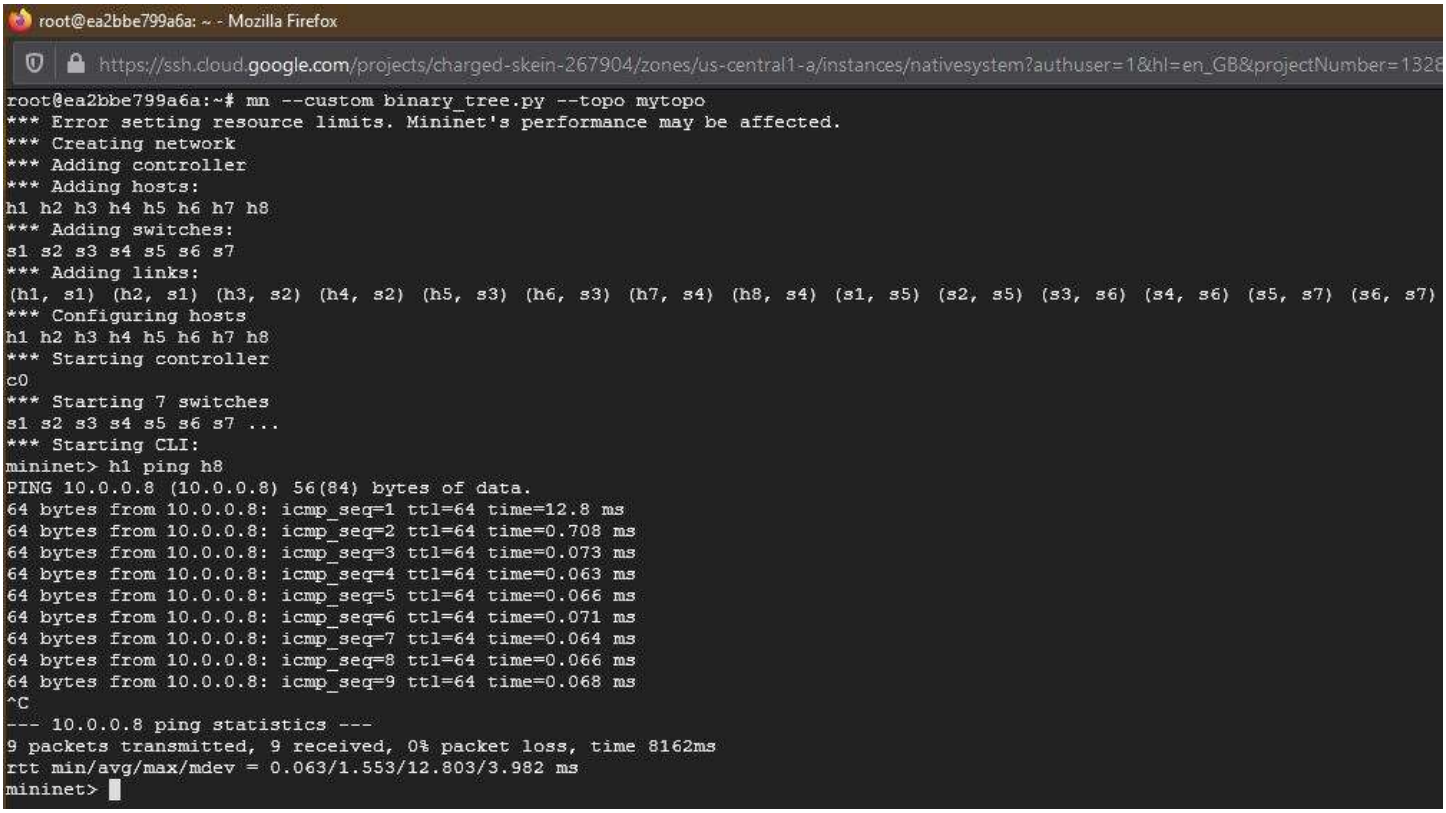
Mini Project 2

Ketan Deshpande

B00816854, kdeshpa5@binghamton.edu

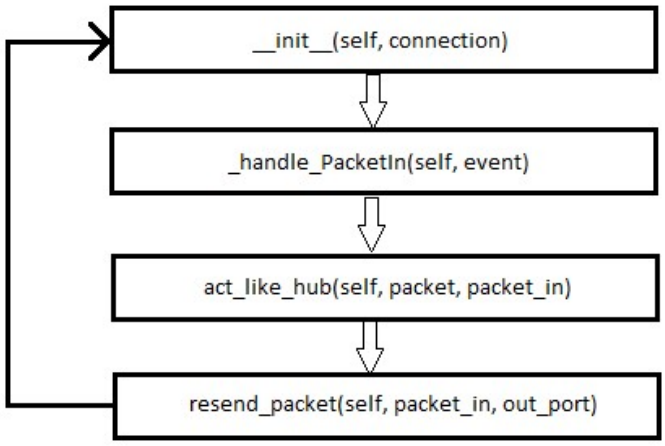
Task 1: Task I ask you to develop a Python script that can populate such a binary tree topology in the Mininet. After completing the Python script, you should run the following commands to verify it "h1" should reach "h8" with ping.

I created a python script named as binary_tree.py to create a topology in mininet. I provided that custom topology to mininet and pinged h1 with h8 by which I got response as below:



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?



The above is the flow of function calls gets called, first it start from __init__ function and next function which gets called is handle_packetin function, handle_packetin function calls act_like_hub function, act_like_hub function calls resend_packet function which floods the packets.

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?

Case 1 - h1 ping -c100 h2 Average Time = 26.196 ms

Case 2 - h1 ping -c100 h8 Average Time = 34.976 ms

From case 1 and case 2, the average time for ping h1 to h8 is more than h1 to h2.

In our topology, h1 and h2 are connected to a same switch i.e. s1 while h1 and h8 are connected to different switches.

The number of switches for the case 2 are more than the case 1 which results into increased flooding in the path from h1 to h8 hence the time to reach the target is more.

Q.3 Run “iperf h1 h2” and “iperf h1 h8”. What is “iperf” used for? What is the throughput for eachcase? What is the difference, and why?

iperf is a widely used tool for network performance measurement and tuning. It has client and server functionality and can create data streams to measure the throughput between the two ends in one or both directions. Typical iperf output contains a time-stamped report of the amount of data transferred and the throughput measured.

Case 1 - iperf h1 h2 Throughput = 5.42 Mbits/sec, 6.33 Mbits/sec

Case 2 - iperf h1 h8 Throughput = 2.89 Mbits/sec, 3.46 Mbits/sec

The throughput observed through iperf from h1 to h2 is more than the throughput from h1 top h8.

This is because of the reason that there is an increased flooding from h1 to h8 as each switch floods the request to the attached switches and nodes than h1 to h2 path, this attributes to the increased time there by reducing the throughput bandwidth.

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).

As per the current functions in ‘of_tutorial’, controller that makes switches flood every packet they receive, and thus, in practice, behave like hubs. This results into flooding and every switch in the flow/path observes the traffic. We can reduce the traffic by enabling the function act_like_switch and disabling the function act_like_hub. The function act_like_switch will implement the functionality of MAC learning controller or MAC learning controller with OpenFlow rules.

Task 3: 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).

Consider the example of h1 ping h2.

When h1 pings h2, the request from h1 goes to s1. s1 will send the packet to the controller seeking for the destination.

Controller will perform below steps -

1. Check if the port mapping is present for the source port (h1), if yes, go to step 2 else store the mapping and then go to step 2.

2. Now, it will check for the destination mapping (h2), if destination mapping is present, it resends the package to the destination. If the destination mapping is not present in mac_to_port list, then it will resend the packet to everyone except the source node.

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)?

Case 1 - h1 ping -c100 h2 Average Time = 25.987 ms

Case 2 - h1 ping -c100 h8 Average Time = 36.794 ms

There is slight improvement in the average time as compared to Hub Case.

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

Case 1 - iperf h1 h2 Throughput = 9.2 Mbits/sec, 9.7 Mbits/sec

Case 2 - iperf h1 h8 Throughput = 3.12 Mbits/sec, 3.69 Mbits/sec

There is significant increment in the throughput as compared to the Hub Case, as improvement in switches behavior.

Task 4: MAC learning controller with OpenFlow rules

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)?

Case 1 - h1 ping -c100 h2 Average Time = 0.720 ms

Case 2 - h1 ping -c100 h8 Average Time = 1.946 ms

There is huge improvement in the average time as compared to other two cases.

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

Case 1 - iperf h1 h2 Throughput = 29.0 Gbits/sec, 29.0 Gbits/sec

Case 2 - iperf h1 h8 Throughput = 24.4 Gbits/sec, 24.4 Gbits/sec

The throughput returned by iperf from a switch is much faster than other two cases.

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

The results got better compared to task 2 and task 3.

Instead of just flooding all the packets to all the hubs initially it learns the port/mac address from controller and then it stores in the flow table i.e. controller installs a flow rule on the switch to handle future packets. When we send the new packets to same destination address, it just uses the stored mapping, and controller gets only those packets for which switch doesn't have a flow entry.

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

```
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)
```

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)?

Cookie: value - A cookie can be associated with a flow using the add-flow add-flows, and mod-flows commands. Its value can be any 64-bit number and need not be unique among flows. If this field is omitted, a default cookie value as 0 is used.

Duration: secs - The time, in seconds, that the entry has been in the table. secs include as much precision as the switch provides, possibly to nanosecond resolution.

n_packets: The number of packets that have matched the entry.

n_bytes: The total number of bytes from packets that have matched the entry.

d1_dst: The destination MAC address

actions: The output port

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s1**

cookie=0x0, duration=1139.179s, table=0, n_packets=27, n_bytes=1918, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s1-eth1"

cookie=0x0, duration=1139.110s, table=0, n_packets=26, n_bytes=1820, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s1-eth2"

cookie=0x0, duration=1138.973s, table=0, n_packets=5, n_bytes=378, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s1-eth3"

cookie=0x0, duration=1138.898s, table=0, n_packets=5, n_bytes=378, dl_dst=e6:14:00:26:0b:79 actions=output:"s1-eth3"

cookie=0x0, duration=1138.753s, table=0, n_packets=5, n_bytes=378, dl_dst=32:65:e2:b0:81:7b actions=output:"s1-eth3"

cookie=0x0, duration=1138.671s, table=0, n_packets=5, n_bytes=378, dl_dst=56:82:2f:45:02:6c actions=output:"s1-eth3"

cookie=0x0, duration=1138.520s, table=0, n_packets=5, n_bytes=378, dl_dst=62:17:89:ae:56:fb actions=output:"s1-eth3"

cookie=0x0, duration=1138.443s, table=0, n_packets=5, n_bytes=378, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s1-eth3"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s2**

cookie=0x0, duration=1179.723s, table=0, n_packets=7, n_bytes=518, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s2-eth3"

cookie=0x0, duration=1179.590s, table=0, n_packets=25, n_bytes=1778, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s2-eth1"

cookie=0x0, duration=1179.515s, table=0, n_packets=24, n_bytes=1736, dl_dst=e6:14:00:26:0b:79 actions=output:"s2-eth2"

cookie=0x0, duration=1179.044s, table=0, n_packets=7, n_bytes=518, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s2-eth3"

cookie=0x0, duration=1178.657s, table=0, n_packets=5, n_bytes=378, dl_dst=32:65:e2:b0:81:7b actions=output:"s2-eth3"

cookie=0x0, duration=1178.648s, table=0, n_packets=5, n_bytes=378, dl_dst=56:82:2f:45:02:6c actions=output:"s2-eth3"

cookie=0x0, duration=1178.516s, table=0, n_packets=5, n_bytes=378, dl_dst=62:17:89:ae:56:fb actions=output:"s2-eth3"

cookie=0x0, duration=1178.503s, table=0, n_packets=5, n_bytes=378, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s2-eth3"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s3**

cookie=0x0, duration=1218.132s, table=0, n_packets=7, n_bytes=518, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s3-eth3"

cookie=0x0, duration=1217.997s, table=0, n_packets=23, n_bytes=1694, dl_dst=32:65:e2:b0:81:7b actions=output:"s3-eth1"

cookie=0x0, duration=1217.911s, table=0, n_packets=22, n_bytes=1652, dl_dst=56:82:2f:45:02:6c actions=output:"s3-eth2"

cookie=0x0, duration=1217.528s, table=0, n_packets=7, n_bytes=518, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s3-eth3"

cookie=0x0, duration=1217.354s, table=0, n_packets=7, n_bytes=518, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s3-eth3"

cookie=0x0, duration=1217.025s, table=0, n_packets=7, n_bytes=518, dl_dst=e6:14:00:26:0b:79 actions=output:"s3-eth3"

cookie=0x0, duration=1216.798s, table=0, n_packets=5, n_bytes=378, dl_dst=62:17:89:ae:56:fb actions=output:"s3-eth3"

cookie=0x0, duration=1216.776s, table=0, n_packets=5, n_bytes=378, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s3-eth3"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s4**

cookie=0x0, duration=1297.211s, table=0, n_packets=7, n_bytes=518, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s4-eth3"

cookie=0x0, duration=1297.139s, table=0, n_packets=21, n_bytes=1610, dl_dst=62:17:89:ae:56:fb actions=output:"s4-eth1"

cookie=0x0, duration=1297s, table=0, n_packets=20, n_bytes=1568, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s4-eth2"

cookie=0x0, duration=1296.760s, table=0, n_packets=7, n_bytes=518, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s4-eth3"

cookie=0x0, duration=1296.524s, table=0, n_packets=7, n_bytes=518, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s4-eth3"

cookie=0x0, duration=1296.250s, table=0, n_packets=7, n_bytes=518, dl_dst=e6:14:00:26:0b:79 actions=output:"s4-eth3"

cookie=0x0, duration=1296.149s, table=0, n_packets=7, n_bytes=518, dl_dst=32:65:e2:b0:81:7b actions=output:"s4-eth3"

cookie=0x0, duration=1296.016s, table=0, n_packets=7, n_bytes=518, dl_dst=56:82:2f:45:02:6c actions=output:"s4-eth3"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s5**

cookie=0x0, duration=1346.872s, table=0, n_packets=23, n_bytes=1638, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s5-eth1"

cookie=0x0, duration=1346.869s, table=0, n_packets=21, n_bytes=1498, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s5-eth2"

cookie=0x0, duration=1346.731s, table=0, n_packets=21, n_bytes=1498, dl_dst=e6:14:00:26:0b:79 actions=output:"s5-eth2"

cookie=0x0, duration=1346.650s, table=0, n_packets=11, n_bytes=854, dl_dst=32:65:e2:b0:81:7b actions=output:"s5-eth3"

cookie=0x0, duration=1346.504s, table=0, n_packets=11, n_bytes=854, dl_dst=56:82:2f:45:02:6c actions=output:"s5-eth3"

cookie=0x0, duration=1346.415s, table=0, n_packets=11, n_bytes=854, dl_dst=62:17:89:ae:56:fb actions=output:"s5-eth3"

cookie=0x0, duration=1346.277s, table=0, n_packets=11, n_bytes=854, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s5-eth3"

cookie=0x0, duration=1346.185s, table=0, n_packets=22, n_bytes=1540, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s5-eth1"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s6**

cookie=0x0, duration=1377.095s, table=0, n_packets=15, n_bytes=1078, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s6-eth3"

cookie=0x0, duration=1377.026s, table=0, n_packets=19, n_bytes=1414, dl_dst=32:65:e2:b0:81:7b actions=output:"s6-eth1"

cookie=0x0, duration=1376.941s, table=0, n_packets=19, n_bytes=1414, dl_dst=56:82:2f:45:02:6c actions=output:"s6-eth1"

cookie=0x0, duration=1376.854s, table=0, n_packets=17, n_bytes=1330, dl_dst=62:17:89:ae:56:fb actions=output:"s6-eth2"

cookie=0x0, duration=1376.716s, table=0, n_packets=17, n_bytes=1330, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s6-eth2"

cookie=0x0, duration=1376.491s, table=0, n_packets=15, n_bytes=1078, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s6-eth3"

cookie=0x0, duration=1376.317s, table=0, n_packets=15, n_bytes=1078, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s6-eth3"

cookie=0x0, duration=1375.988s, table=0, n_packets=15, n_bytes=1078, dl_dst=e6:14:00:26:0b:79 actions=output:"s6-eth3"

root@ea2bbe799a6a:~/pox/pox/misc# **ovs-ofctl dump-flows s7**

cookie=0x0, duration=1411.400s, table=0, n_packets=15, n_bytes=1078, dl_dst=12:b9:07:ab:7c:d9 actions=output:"s7-eth1"

cookie=0x0, duration=1411.397s, table=0, n_packets=11, n_bytes=854, dl_dst=32:65:e2:b0:81:7b actions=output:"s7-eth2"

cookie=0x0, duration=1411.249s, table=0, n_packets=11, n_bytes=854, dl_dst=56:82:2f:45:02:6c actions=output:"s7-eth2"

cookie=0x0, duration=1411.162s, table=0, n_packets=11, n_bytes=854, dl_dst=62:17:89:ae:56:fb actions=output:"s7-eth2"

cookie=0x0, duration=1411.024s, table=0, n_packets=11, n_bytes=854, dl_dst=f6:5f:1b:0e:0e:4a actions=output:"s7-eth2"

cookie=0x0, duration=1410.796s, table=0, n_packets=15, n_bytes=1078, dl_dst=c6:7c:4b:66:c2:1f actions=output:"s7-eth1"

cookie=0x0, duration=1410.621s, table=0, n_packets=15, n_bytes=1078, dl_dst=3a:96:b8:2c:ad:e8 actions=output:"s7-eth1"

cookie=0x0, duration=1410.293s, table=0, n_packets=15, n_bytes=1078, dl_dst=e6:14:00:26:0b:79 actions=output:"s7-eth1"

Task 5: Layer-3 routing

We can use the following commands to set the flow table rules for IP matching –

Consider, example of ping h1 to h8

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

nw_src and nw_dst are the identifier of ip source/destination address respectively.

But before you set the ip source/destination address, you must set the Ethernet Type(dl_type). Type values include : 0x0800: matches IPv4 source/destination address ip (eg: ip , tcp).0x0806:thearp protocol type, matches the ar_sqa or ar_tpa field. respectively, in ARP packets for IPv4 and Ethernet. 0x8035:therarp protocol type, matches the ar_spa or ar_tpa field, respectively, in RARP packets for IPv4 and Ethernet. Other than 0x0800, 0x0806, or 0x8035, the values of nw_src and nw_dst are ignored.