COEN 241(Cloud Computing)
Poonam Kanani(1620239)

Homework 3

```
root@e44c90101425:~# mn --custom binary_tree.py --topo binary_tree
*** 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, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s2, s1) (s3, s2) (s4, s2) (s5, s1) (s6, s5) (s7, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

Task 1

1. What is the output of “nodes” and “net”
   Output can be seen in screenshot below.
   “nodes” prints all the nodes available in network
   “net” prints all the available connections

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
```

2. What is the output of “h7 ifconfig”

```
mininet> h7 ifconfig
h7-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.7  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::8047:43ff:fe00:2639  prefixlen 64  scopeid 0x20<link>
        ether 82:47:43:00:26:39  txqueuelen 1000  (Ethernet)
        RX packets 61  bytes 4766 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Task 2

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?

   Function call of controller can be described as below:
   - Controller starts with launch() function
   - It starts listener which calls start_switch() when connection is made
   - For each switch object of class Tutorial is created and constructor is called for the same
   - _handle_PacketIn() handles packets in msg from switch
   - It calls act_like_hub() or act_like_switch(), and they in turn calls resend_packet().

2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).
   mininet> h1 ping -c100 h2

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99150ms
rtt min/avg/max/mdev = 1.631/1.787/5.978/0.435 ms
mininet>
```

   mininet> h1 ping -c100 h8

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99134ms
rtt min/avg/max/mdev = 8.199/11.682/26.847/3.936 ms
mininet>
```

   a. How long does it take (on average) to ping for each case?
      Avg time for each case is as below:
      h1 -> h2: 1.787 ms
      h1 -> h8: 11.682 ms

   b. What is the minimum and maximum ping you have observed?
      Minimum time for each case is as below:
      h1 -> h2: 1.631 ms
      h1 -> h8: 8.199 ms

      Maximum time for each case is as below:
      h1 -> h2: 5.978 ms
      h1 -> h8: 26.847 ms

   c. What is the difference, and why?
      Timings for pinging h8 are significantly higher than that for h2. This is because h8 is farther from h1 when compared to h2. While pinging h2, path will be as: h1 -> s3 -> h2. However while pinging h8 ,path will be as: h1 -> s3 -> s2 -> s1 -> s5 ->

s7 -> s8 -> h8. From this we can see that since route to h8 contains more hops than h2 it takes longer time.

3. Run “iperf h1 h2” and “iperf h1 h8”

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['6.41 Mbits/sec', '7.52 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.01 Mbits/sec', '2.46 Mbits/sec']
mininet>
```

   a. What is “iperf” used for?
   Iperf checks bandwidth between two hosts

   b. What is the throughput for each case?
   In case of h1 h2, throughput of **h1 is 6.41 Mbits/sec** and throughput of **h2 is 7.52 Mbits/sec**
   In case of h1 h8, throughput of **h1 is 2.01 Mbits/sec** and throughput of **h8 is 2.46 Mbits/sec**

   c. What is the difference, and explain the reasons for the difference.
   Again in this case throughput while sending to h2 is higher than in case of h8. Again for the same reason because it takes longer time for each packet to reach h8 when compared to h2. Also each packet needs acknowledgement which takes additional time causing congestion at the top level switch. So s1 creates a bottleneck which causes delay for packages traveling between h1 and h8.

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).

   Each switch observes traffic because act_like_hub() sends packets to all the switches. We can observe traffic by adding print statement in _handle_PacketIn() function. We can get id of the switch using self.connection.dpid and print it to know when traffic is observed by particular switch.

Task 3

1. 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).

"MAC to Port" map keeps track of all the MAC to Port mapping. So while sending the packet, if the destination MAC already exists in the map it can be directly sent to the destination and in turn reducing unnecessary traffic. Let's see h1 ping h2 example to see the process in details.

- Initially map is empty
- When h1 pings h2, we check if mapping of h1 to port exists. It doesn't exist so it gets inserted in the map.
- Next destination MAC address to port mapping is checked.
- In this case mapping does not exist for h2 yet, so we sends data to all the ports.
- Now when h2 receives the packet it sends out acknowledgement.
- While sending we check if mapping exists for h2. It doesn't exist so it gets inserted in the map.
- Now the port of the destination i.e., h1 is checked.
- Port of h1 is already available in map. So packet is directly sent to h1.

  If we ping second time, both source and destination ports would be already known so packet will be directly send from h1 to h2 only and then acknowledge will be sent from h2 to h1 only.

2. (Comment out all prints before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).
   mininet> h1 ping -c100 h2

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99145ms
rtt min/avg/max/mdev = 1.645/1.740/2.659/0.110 ms
```

   mininet> h1 ping -c100 h8

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 6.708/7.207/7.801/0.248 ms
```

   a. How long did it take (on average) to ping for each case?
      Avg time for each case is as below:
      h1 -> h2: 1.740 ms
      h1 -> h8: 7.207 ms

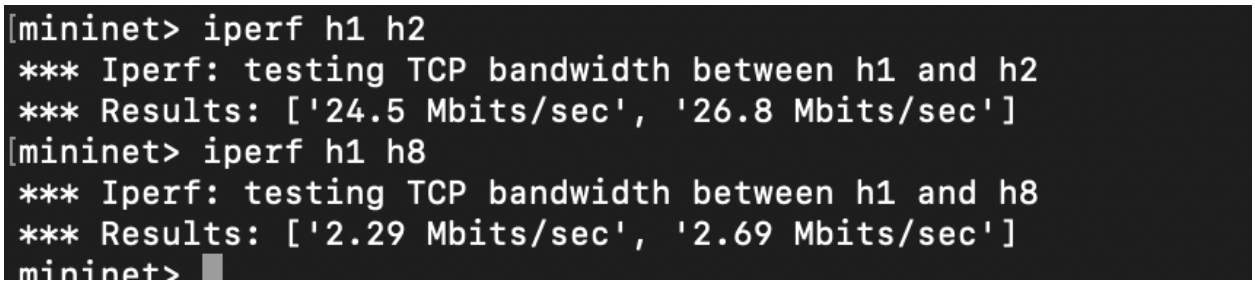
   b. What is the minimum and maximum ping you have observed?
      Minimum time for each case is as below:
      h1 -> h2: 1.645 ms
      h1 -> h8: 6.708 ms

      Maximum time for each case is as below:
      h1 -> h2: 2.659 ms

h1 -> h8: 7.801 ms

c. Any difference from Task 2 and why do you think there is a change if there is?
It is faster than in task 2. Because in this case once the port for a given MAC is known, a packet is sent directly to it. In task 2 it was being sent to every port. This causes network flooding causing delay in case when a packet needs to pass through a busy node like switch s1. When a network makes smart choices and sends data to multiple nodes only when it is necessary, it decreases traffic and hence makes it faster. Also we can see that there isn't a significant difference in case of h1 -> h2. This might be because they have only one switch between them, which isn't a high demand switch even when a packet is being sent to every port.

3. Run “iperf h1 h2” and “iperf h1 h8”.



a. What is the throughput for each case?
In case of h1 h2, throughput of **h1 is 24.5 Mbits/sec** and throughput of **h2 is 26.8 Mbits/sec**
In case of h1 h8, throughput of **h1 is 2.29 Mbits/sec** and throughput of **h8 is 2.69 Mbits/sec**

b. What is the difference from Task 2 and why do you think there is a change if there is?
Throughput also increases from Task 2 to Task 3. This is because the network is not flooded with unnecessary traffic. Once the destination port is known, it is stored in a map and used for all the future transmission. This way it decreases network traffic. Hence reducing congestion and increasing throughput.