# CSEN 241 HW 3
# Mininet & OpenFlow

## https://github.com/soy-crypto/COEN241_Cloud-Computing

**Task1:**

1. **Binary tree**

```
from mininet.topo import Topo

class BinaryTreeTopo( Topo ):
   "Binary Tree Topology Class."

   def __init__( self ):
      "Create the binary tree topology."

      # Initialize topology
      Topo.__init__( self )

      # Add hosts
      Host1 = self.addHost( 'h1' )
      Host2 = self.addHost( 'h2' )
      Host3 = self.addHost( 'h3' )
      Host4 = self.addHost( 'h4' )
      Host5 = self.addHost( 'h5' )
      Host6 = self.addHost( 'h6' )
      Host7 = self.addHost( 'h7' )
      Host8 = self.addHost( 'h8' )

      # Add switches
      Switch1 = self.addSwitch('s1')
      Switch2 = self.addSwitch('s2')
      Switch3 = self.addSwitch('s3')
      Switch4 = self.addSwitch('s4')
      Switch5 = self.addSwitch('s5')
      Switch6 = self.addSwitch('s6')
      Switch7 = self.addSwitch('s7')

      # Add links
      self.addLink( Host1, Switch3 )
      self.addLink( Host2, Switch3 )
      self.addLink( Host3, Switch4 )
      self.addLink( Host4, Switch4 )
```

```
        self.addLink( Host5, Switch6 )
        self.addLink( Host6, Switch6 )
        self.addLink( Host7, Switch7 )
        self.addLink( Host8, Switch7 )


        self.addLink( Switch2, Switch3 )
        self.addLink( Switch2, Switch4 )
        self.addLink( Switch5, Switch6 )
        self.addLink( Switch5, Switch7 )
        self.addLink( Switch1, Switch2 )
        self.addLink( Switch1, Switch5 )

topos = { 'binary_tree': ( lambda: BinaryTreeTopo() ) }
```

**Table. 1. Binary tree.py**

2. **Question:**
   - **What is the output of “node” and “net”?**
     For “node”,
     C0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7

     For “net”,

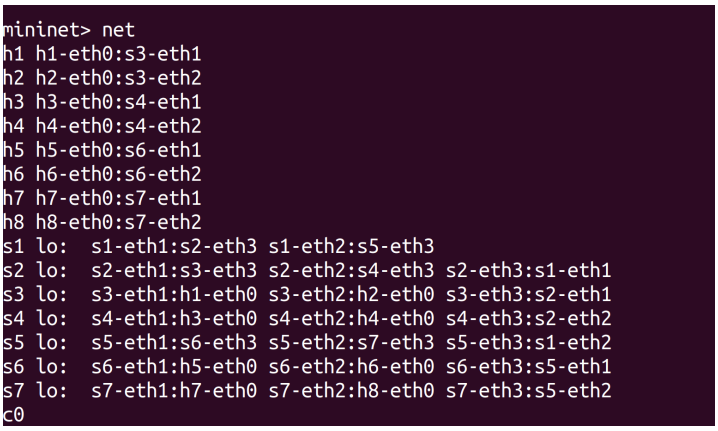


Fig.2. net

   - **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::147e:51ff:fe9d:7360  prefixlen 64  scopeid 0x20<link>
        ether 16:7e:51:9d:73:60  txqueuelen 1000  (Ethernet)
        RX packets 39715  bytes 39205058 (39.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1076 (1.0 KB)
        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
```

Fig.3. H7 ifconfig

**Task2:**

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?**
   **Solution:**
   Here is the flowchart of the controller.

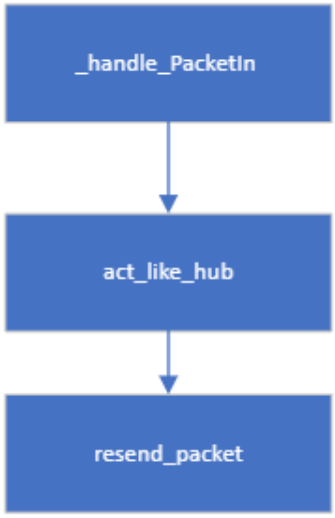


Fig.3. the flowchart

2. **Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).**
   1) **How long does it take (on average) to ping for each case?**
      For h1 to h2,

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99206ms
rtt min/avg/max/mdev = 0.785/2.637/5.816/1.090 ms
```

      The average time = 2.637 ms

      For h1 to h8,

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99172ms
rtt min/avg/max/mdev = 2.648/6.920/12.736/2.247 ms
```

The average time = 6.920

2) **What is the minimum and maximum ping you have observed?**

| COMMAND | minimum(ms) | maximum(ms) |
|---|---|---|
| H1 -> H2 | 0.785 | 5.816 |
| H1-> H8 | 2.648 | 12.736 |

Table.2. Min and max

3) **What is the difference, and why?**
The reason for this is that there is only one switch between h1 and h2, while there are 5 Switches between h1 and h8.

3. **Run "iperf h1 h2" and "iperf h1 h8"**
   1) **What is "iperf" used for?**

      The command "iperf" is used for monitoring the bandwidth between two nodes.

   2) **What is the throughput for each case?**
      **For h1 to h2**

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['25.6 Mbits/sec', '28.6 Mbits/sec']
```

**For h1 to h8**

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.76 Mbits/sec', '3.24 Mbits/sec']
```

   3) **What is the difference, and explain the reasons for the difference.**

      The reason for this is that there is only one switch between h1 and h2, while there are 5

Switches 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).**
   For h1 -> h2, s3 observe the traffic. However, for h1 -> h8, the switches s1, s2, s3, s5, and s7 are responsible for the communication between h1 and h8.

**Task3:**

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).**
   **Solution:**
   In of_tutorial.py, now the act_like_switch function is in charge of routing all the MAC addresses to a particular port number with the help of a dictionary mac_to_port. If the destination is not known, it will simply flood all the packets to all destinations and once found the right destination it will be added to the dictionary for future use. Next time when the same request is received it just looks up the dictionary map_to_port and sends accordingly. This is what we have described as MAC learning and this will ensure lesser network congestion and thus higher throughput.

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).**
   1) **How long did it take (on average) to ping for each case?**
      For h1 to h2:

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99268ms
rtt min/avg/max/mdev = 0.739/2.593/6.454/1.232 ms
```

      The average time = 2.593 ms

      For h1 to h8:

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99159ms
rtt min/avg/max/mdev = 2.233/6.769/48.233/4.746 ms
```

      The average time = 6.769

   2) **What is the minimum and maximum ping you have observed?**

| COMMAND | minimum(ms) | maximum(ms) |
|---|---|---|
| H1 -> H2 | 0.739 | 6.454 |
| H1-> H8 | 2.233 | 48.233 |

| | | |
|---|---|---|
| | | |

**Table.3. Min and max**

3) **Any difference from Task 2 and why do you think there is a change if there is?**

The average RTTs of Task3 are slightly faster than that of task2 because task2 uses flood to forward packets.However in task3, the controller is able to get MAC addresses and send them to the appropriate destination port using the mapping in mac_to_port (MAC learning).

3. **Run "iperf h1 h2" and "iperf h1 h8".**
   1) **What is the throughput for each case?**

**For iperf h1 h2,**

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['27.7 Mbits/sec', '30.9 Mbits/sec']
```

**For "iperf h1 h8",**

```
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['6.75 Mbits/sec', '7.33 Mbits/sec']
```

   2) **What is the difference from Task 2 and why do you think there is a change if there is?**

The throughputs in Task3 are greater than that of Task 2. The reason for this is that the implementation of the MAC learning enables appropriate mapping of MAC addresses to the destination port, leading to reduced congestion and increasing throughput.