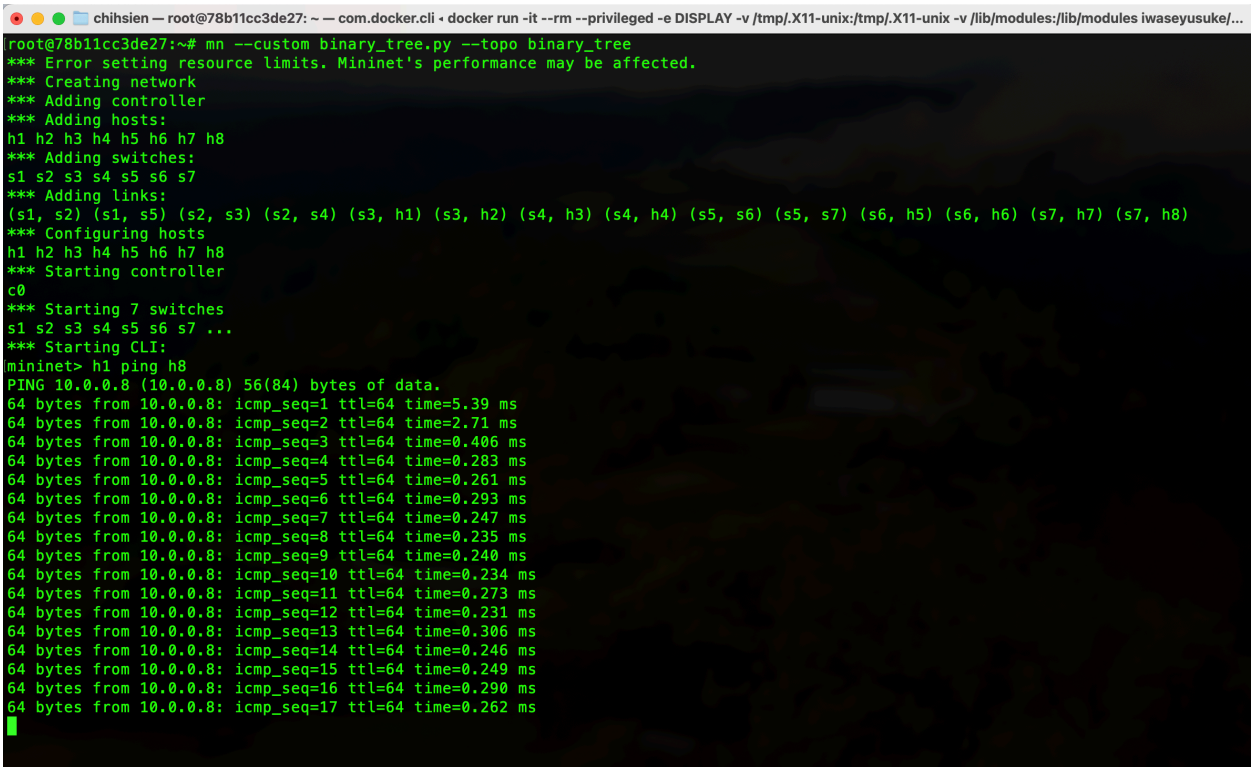
CSEN241-HW3 W1652176 Chi-Hsien Wu

GitHub URL: https://github.com/chwu891/Cloud-Computing.git
Git Commit ID: ee0a55a

Task1

$ mn --custom binary_tree.py --topo binary_tree
mininet> h1 ping h8



1. What is the output of “nodes” and “net”

```
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-eth2
h2 h2-eth0:s3-eth3
h3 h3-eth0:s4-eth2
h4 h4-eth0:s4-eth3
h5 h5-eth0:s6-eth2
h6 h6-eth0:s6-eth3
h7 h7-eth0:s7-eth2
h8 h8-eth0:s7-eth3
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s5-eth1
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:s4-eth1
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h1-eth0 s3-eth3:h2-eth0
s4 lo:  s4-eth1:s2-eth3 s4-eth2:h3-eth0 s4-eth3:h4-eth0
s5 lo:  s5-eth1:s1-eth2 s5-eth2:s6-eth1 s5-eth3:s7-eth1
s6 lo:  s6-eth1:s5-eth2 s6-eth2:h5-eth0 s6-eth3:h6-eth0
s7 lo:  s7-eth1:s5-eth3 s7-eth2:h7-eth0 s7-eth3:h8-eth0
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::f083:66ff:fed4:6a1f  prefixlen 64  scopeid 0x20<link>
        ether f2:83:66:d4:6a:1f  txqueuelen 1000  (Ethernet)
        RX packets 49  bytes 3834 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 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
```

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?

packet comes -> _handle_PacketIn() -> act_like_hub() -> resend_packet() ->
send the packet to a specific port

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

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99320ms
rtt min/avg/max/mdev = 1.193/5.457/6.918/0.846 ms
mininet>
```

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99328ms
rtt min/avg/max/mdev = 4.623/14.941/59.402/5.066 ms
mininet>
```

- How long does it take (on average) to ping for each case?

  h1 ping -c100 h2: 5.457 ms
  h1 ping -c100 h8: 14.941 ms

- What is the minimum and maximum ping you have observed?

  minimum:
  h1 ping -c100 h2: 1.193 ms
  h1 ping -c100 h8: 4.623 ms
  maximum:
  h1 ping -c100 h2: 6.918 ms
  h1 ping -c100 h8: 59.402 ms

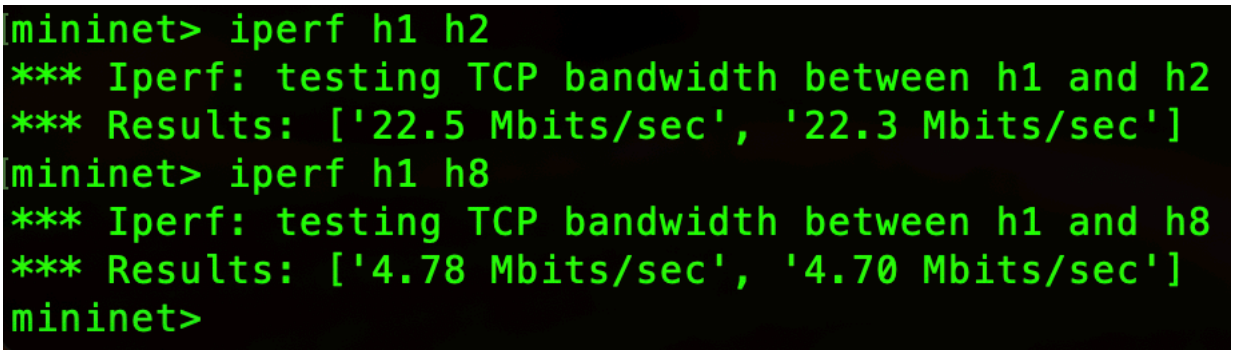
- What is the difference, and why?

When h1 pings h2, the communication is more efficient compared to when h1 pings h8. This is because h1 has a direct connection to h2 through only one switch, resulting in a shorter path for the data to traverse. On the other hand, when h1 pings h8, the data must navigate through a more complex network, involving five switches in total. This increased number of intermediary switches introduces additional latency and potentially slows down the communication process between h1 and h8.

3. Run “iperf h1 h2” and “iperf h1 h8”
    - What is “iperf” used for?

“iperf” is an open-source tool designed to measure network performance by assessing bandwidth, latency, and other metrics between two systems in a client-server architecture. It is commonly used for evaluating the speed and reliability of network connections in various operating environments.

- What is the throughput for each case?



- What is the difference, and explain the reasons for the difference.

The disparity in throughput between h1 and h2, compared to h1 and h8, stems from the differing network configurations. The direct link between h1 and h2 via a single switch results in higher throughput due to reduced network congestion. In contrast, the connection between h1 and h8 involves traversing multiple switches, leading to increased latency and decreased data transfer speed. The additional hops introduce complexities that contribute to a lower overall throughput, highlighting the impact of network topology on data transmission efficiency.

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).

Invoking the _handle_PacketIn() method in the of_tutorial.py file occurs whenever a packet is received. Therefore, incorporating the specified line within this method facilitates traffic inspection, revealing that all switches are actively monitoring the network's traffic.

log.info("Switch Traffic: %s" % (self.connection))

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).

The inclusion of the provided code in the act_like_switch() method facilitates the identification of the locations of MAC addresses. When the controller recognizes that a MAC address corresponds to the intended recipient of a message, it can efficiently map the address to a specific port. This streamlined process enhances the controller's speed in transmitting packets to known addresses, as they are seamlessly directed to their designated well-known port. In cases where the destination address is unfamiliar, the controller resorts to broadcasting the packet to all possible destinations, a technique known as flooding. The MAC Learning Controller contributes to improved ping times and throughput by minimizing the need for flooding and optimizing the handling of packet transmissions.

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).

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99341ms
rtt min/avg/max/mdev = 2.528/5.669/9.071/1.016 ms
```

```
--- 10.0.0.8 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99315ms
rtt min/avg/max/mdev = 4.547/13.731/18.888/2.329 ms
```

- How long did it take (on average) to ping for each case?

  h1 ping -c100 h2: 5.669 ms
  h1 ping -c100 h8: 13.731 ms

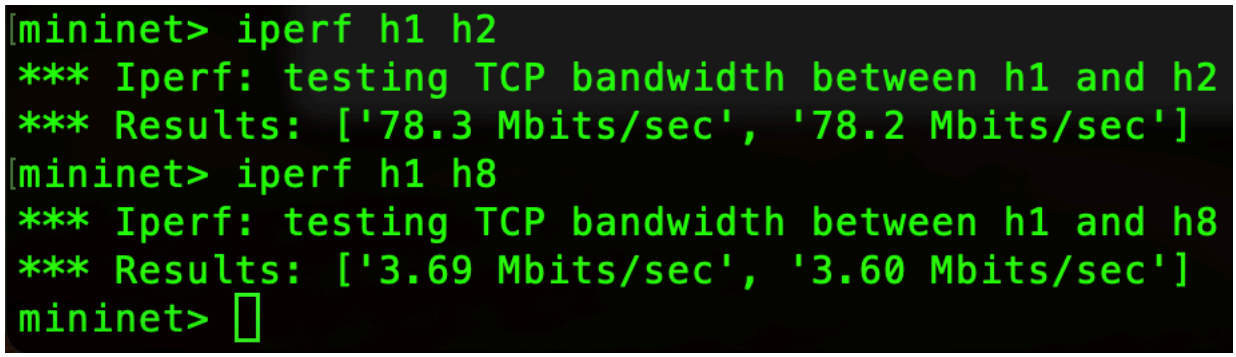
- What is the minimum and maximum ping you have observed?

  minimum:
  h1 ping -c100 h2: 2.528 ms
  h1 ping -c100 h8: 9.071 ms
  maximum:
  h1 ping -c100 h2: 4.547 ms
  h1 ping -c100 h8: 18.888 ms

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

When examining task 2, the duration for h1 pinging h2 and h1 pinging h8 doesn't exhibit a substantial time disparity. However, the latter case demonstrates quicker response times because the switch avoids unnecessary flooding to every node. Instead, it intelligently learns the location of the hosts and directs packets straight to their destinations. The "mac_to_port" list variable stores the destination MAC addresses, mitigating network congestion by facilitating direct and efficient routing based on acquired knowledge.

3. Run “iperf h1 h2” and “iperf h1 h8”.
    - What is the throughput for each case?



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

The throughput observed in task 3 surpasses that of task 2. This improvement is attributed to the "MAC_to_port" mapping having learned all the ports, leading to a reduction in packet flooding and consequently minimizing network congestion. Notably, when comparing task 2 and task 3, a substantial enhancement in throughput is observed for h1 and h2. This can be attributed to pre-computed and trained routes, reflecting changes in the controller's configuration. Conversely, in the case of h1 and h8, there is no significant difference in throughput between task 2 and task 3. This lack of distinction is influenced by the increased number of hops and the consideration of packet dropping in the routing process.