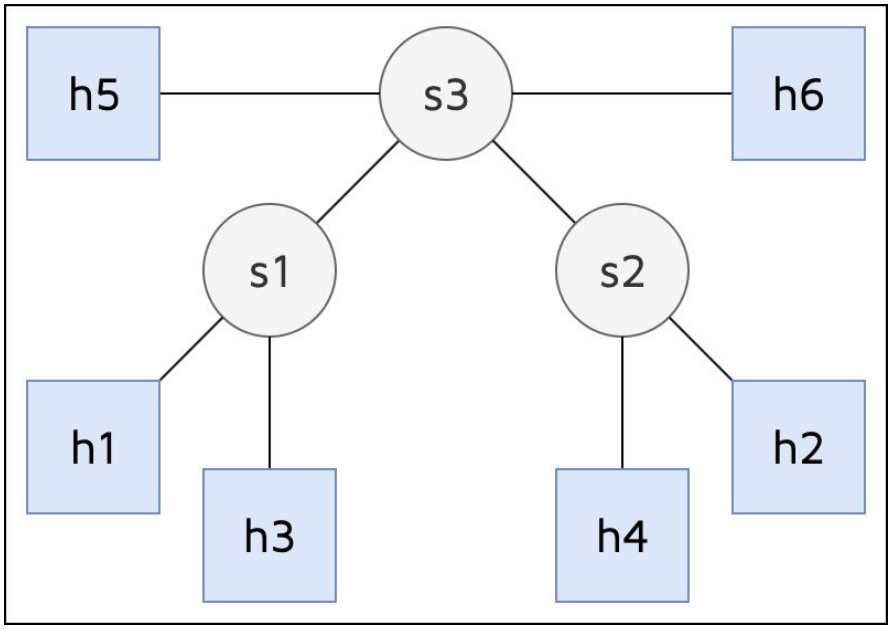
Ashwin Chidambaram

CSE 150/L - HW 1

01/19/20

Lab 1 Question

1. In Mininet change the default configuration through a python script to have 6 hosts connected through three switches as shown in the figure:



Answer:

```python
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class MyTopology(Topo):
    """
    A basic topology
    """
    def __init__(self):
        Topo.__init__(self)

        # Set Up Topology Here
        switch1 = self.addSwitch('s1')                  ## Adds Switch 1    [s1]
        switch2 = self.addSwitch('s2')                  ## Adds Switch 2    [s2]
        switch3 = self.addSwitch('s3')                  ## Adds Switch 3    [s3]

        host1 = self.addHost('h1')                      ## Adds a Host      [h1]
        self.addLink(host1, switch1)                    ## Link             [h1] to [s1]


        host2 = self.addHost('h2')                      ## Adds a Host      [h2]
        self.addLink(host2, switch2)                    ## Link             [h2] to [s2]

        host3 = self.addHost('h3')                      ## Adds a Host      [h3]
        self.addLink(host3, switch1)                    ## Link             [h3] to [s1]

        host4 = self.addHost('h4')                      ## Adds a Host      [h4]
        self.addLink(host4, switch2)                    ## Link             [h4] to [s2]

        host5 = self.addHost('h5')                      ## Adds a Host      [h5]
        self.addLink(host5, switch3)                    ## Link             [h5] to [s3]

        host6 = self.addHost('h6')                      ## Adds a Host      [h6]
        self.addLink(host6, switch3)                    ## Link             [h6] to [s3]

        self.addLink(switch1, switch3)                  ## addLink          [s1] to [s3]
        self.addLink(switch2, switch3)                  ## addLink          [s2] to [s3]

if __name__ == '__main__':
    """
    If this script is run as an executable (by chmod +x), this is
    what it will do
    """

    topo = MyTopology()            ## Creates the topology
    net = Mininet( topo=topo )        ## Loads the topology
    net.start()                     ## Starts Mininet

    # Commands here will run on the simulated topology
    CLI(net)

    net.stop()                      ## Stops Mininet
```

2. [30 pts] Save a screenshot of *dump* and *pingall* output. Explain what is being shown in the screenshot.

dump:

```
mininet@mininet-vm:~/Desktop$ sudo python example-topo.py
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=5019>
<Host h2: h2-eth0:10.0.0.2 pid=5023>
<Host h3: h3-eth0:10.0.0.3 pid=5025>
<Host h4: h4-eth0:10.0.0.4 pid=5027>
<Host h5: h5-eth0:10.0.0.5 pid=5029>
<Host h6: h6-eth0:10.0.0.6 pid=5031>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=5036>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=5039>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None pid=5042>
<Controller c0: 127.0.0.1:6633 pid=5012>
```

Explanation: the *dump* command returns information about all of the nodes within the network. From the screenshot above, we can tell what each of the connected nodes [h1, h2, h3, h4, h5, h6, s1, s2, s3] are, and their assigned ip addresses. And since this is a simple network simulator, we have each of the hosts with a simple one digit variation in ip, with the switches all sharing the same ip.

pingall:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
```

Explanation: the *pingall* command returns information on whether the nodes are all connected, and capable of sending/receiving data. From our screenshot above, we can tell that the command systematically has each host send data packets to the other 5 hosts, and checks to see if there is any loss during transmission. In the network I setup per instruction, there is 0% packet loss.

3. [10 pts] Run the *iperf* command as well, and screenshot the output, how fast is the connect?

Answer: the connection speed is 41 GB/s

4. Run wireshark, and using the display filter, filter for “of”. Note: When you run wireshark you should do so as “sudo wireshark”. When you choose an interface to capture on, you should select “any”.
    a. [20 pts] Run ping from a host to any other host using *hX ping -c 5 hY*. How many *of_packet_in* messages show up? Take a screenshot of your results.
        - When running the ping, I get **11** “*of_packet_in*” messages
    b. [20 pts] What is the source and destination IP addresses for these entries? Find another packet that matches the “of” filter with the OpenFlow typefield set to *OFPT_PACKET_OUT*. What is the source and destination IP address for this entry? Take screenshots showing your results.
        - Source ip (*OFPT_PACKET_IN)*: 10.0.0.1
        - Destination ip (*OFPT_PACKET_IN)*: 10.0.0.6
        - Source ip (*OFPT_PACKET_OUT)*: 127.0.0.1
        - Destination ip (*OFPT_PACKET_OUT)*: 127.0.0.1
    c. [20 pts] Replace the display filter for “of” to “icmp && not of”. Run *pingall* again, how many entries are generated in wireshark? What types of icmp entries show up? Take a screenshot of your results.
        - 870 entries are generated
        - The two types of icmp entries that show up are
            - (Echo (ping) request)
            - (Echo (ping) reply)

Screenshots for Question 4:

- [4a]

No.	Time	Source	Destination	Protocol	Length	Info
65	0.000049000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
66	0.000015000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
67	0.000851000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
69	0.000224000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
71	0.000175000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
74	0.000261000	10.0.0.1	10.0.0.6	OF 1.0	184	of_packet_in
75	0.000447000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
81	0.000114000	10.0.0.1	10.0.0.6	OF 1.0	184	of_packet_in
82	0.000265000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
90	0.000129000	10.0.0.6	10.0.0.1	OF 1.0	184	of_packet_in
91	0.000008000	10.0.0.1	10.0.0.6	OF 1.0	184	of_packet_in
92	0.000252000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
97	0.000237000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
100	0.000200000	10.0.0.6	10.0.0.1	OF 1.0	184	of_packet_in
101	0.000258000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
106	0.000349000	10.0.0.1	10.0.0.6	OF 1.0	184	of_packet_in
107	0.000418000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
111	0.000199000	10.0.0.1	10.0.0.6	OF 1.0	184	of_packet_in
112	0.000251000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
143	0.628208000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
144	0.000561000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
147	0.000492000	ea:7e:e2:3b:72:57	7e:36:51:a6:ee:fe	OF 1.0	128	of_packet_in
148	0.000500000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
152	0.000450000	ea:7e:e2:3b:72:57	7e:36:51:a6:ee:fe	OF 1.0	128	of_packet_in
153	0.000310000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
157	0.000206000	7e:36:51:a6:ee:fe	ea:7e:e2:3b:72:57	OF 1.0	128	of_packet_in
158	0.000187000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
161	0.000188000	7e:36:51:a6:ee:fe	ea:7e:e2:3b:72:57	OF 1.0	128	of_packet_in
162	0.000179000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
166	4.536874000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
167	0.000059000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
168	0.000015000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
169	0.000418000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
171	0.000233000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
173	0.000214000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
175	4.998877000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request

- [4b]

```
    12 0.000469000  127.0.0.1            127.0.0.1            OF 1.0       92 of_packet_out
▸ Frame 12: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 59456 (59456), Seq: 9, Ack: 125, Len: 24
▽ OpenFlow
    version: 1
    type: OFPT_PACKET_OUT (13)
    length: 24
    xid: 0
    buffer_id: 288
    in_port: 1
    actions_len: 8
  ▷ of_action list
```

- [4c]

```
    71 0.000005000  10.0.0.1             10.0.0.3             ICMP        100 Echo (ping) request  id=0x2ad9, seq=1/256, ttl=64
▷ Frame 71: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.3 (10.0.0.3)
▽ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xe698 [correct]
    Identifier (BE): 10969 (0x2ad9)
    Identifier (LE): 55594 (0xd92a)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    Timestamp from icmp data: Jan 19, 2020 19:24:00.134788000 PST
    [Timestamp from icmp data (relative): 0.001250000 seconds]
  ▷ Data (48 bytes)
```

```
    52 0.000064000  10.0.0.2             10.0.0.1             ICMP        100 Echo (ping) reply    id=0x2ad8, seq=1/256, ttl=64
▷ Frame 52: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
▷ Linux cooked capture
▷ Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
▽ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xebb1 [correct]
    Identifier (BE): 10968 (0x2ad8)
    Identifier (LE): 55338 (0xd82a)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    Timestamp from icmp data: Jan 19, 2020 19:24:00.128647000 PST
    [Timestamp from icmp data (relative): 0.003758000 seconds]
  ▷ Data (48 bytes)
```