COEN 241: Cloud Computing
Mininet and OpenFlow

Janan Gandhi (W1607782)

Task 1

Q1. What is the output of “nodes” and “net”

```
mininet> nodes
available nodes are:
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
```

Q2. 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::249f:d0ff:fe0c:bbb8  prefixlen 64  scopeid 0x20<link>
        ether 26:9f:d0:0c:bb:b8  txqueuelen 1000  (Ethernet)
        RX packets 155  bytes 11890 (11.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 936 (936.0 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

Q1. 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:

The functions are called in the following order:

launch()
start_switch (event)
Tutorial class constructor __init__
_handle_PacketIn (self, event)
act_like_hub (self, packet, packet_in)
resend_packet (self, packet_in, out_port)

Q2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

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

h1 ping -c100 h2 : 5.282 ms

h1 ping -c100 h8 : 9.406 ms

b. What is the minimum and maximum ping you have observed?

- h1 ping -c100 h2 :
  Max - 11.742 ms
  Min - 1.533 ms

- h1 ping -c100 h8 :
  Max - 16.432 ms
  Min - 5.824 ms

c. What is the difference, and why?

H1 ping H8 almost always takes longer than H1 ping H2. This is because the packer has to go through more number of switches in the second case. The path taken in both cases is as follows:

H1 ping h2 - h1->s3->h2
H1 ping h8 - h1->s3->s2->s1->s5->s7->h8

Hence, since the number of jumps is more comparatively, h1 pinging h8 takes longer than h1 pinging h2.

Q3. Run “iperf h1 h2” and “iperf h1 h8”

a) What is “iperf” used for?

It is to test the bandwidth between two hosts.

b) What is the throughput for each case?

H1 -> H2 : Bandwidth is 24.4 Mbits/sec
H1 -> H8 : Bandwidth is 6.38 Mbits/sec

c) What is the difference, and explain the reasons for the difference?

The bandwidth for h1 -> h2 is greater compared to h1 -> h8 because of the following reasons:

1. Latency - It is the time taken by a packet to travel from sender to receiver. Since the number of hops is more in h1 -> h8, the latency is more leading to lower bandwidth.
2. Network congestion - This occurs because there is too much traffic passing through the switches.

Q4. 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).

The switches act as a hub by sending traffic to all ports except the incoming port. In order to verify this, I added the below snippet in the handle_packetIn function declaration :

```
def _handle_PacketIn (self, event):
  """
  Handles packet in messages from the switch.
  """
  log.debug("Switch observing traffic: %s" % (self.connection))

  packet = event.parsed # This is the parsed packet data.
  if not packet.parsed:
    log.warning("Ignoring incomplete packet")
    return

  packet_in = event.ofp # The actual ofp_packet_in message.

  # Comment out the following line and uncomment the one after
  # when starting the exercise.
  # self.act_like_hub(packet, packet_in)
  self.act_like_switch(packet, packet_in)
```

The handle_packetIn function is invoked every time the switch receives some traffic. Hence whenever a packet will be passed through a switch this print statement will display the switch number the traffic is passing through.

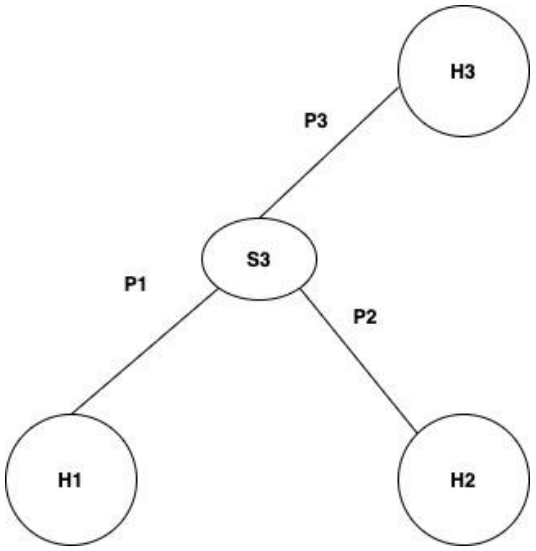
Task 3

Q1. 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:

“MAC to port” is a mapping table that each switch stores where they keep entries of the MAC addresses and their corresponding port. This table is initially empty and the switch keeps adding entries based on a process of trial and error. How this works is that when a switch recieves a new packet, the act_like_switch function first checks if the map already contains the source MAC address. If the map doesn’t, then the switch maps the source MAC address to the corresponding port.

After this, the next step is to check the destination MAC address entry. If an entry is not found, the switch floods the packet to all ports. When it receives an acknowledgement from a port, it adds the entry back to the mapping table for next time. For example, consider the below with the below structure, H1 is about to ping H2 10 times:



The steps would be as follows:

First ping:

1. H1 sends a packet over P1 and is received by S3
2. S3 checks the MAC to port table to map the packet to H1 via the port P1.
3. S3 checks the MAC to port table for the destination address but doesn't find an entry.
4. S3 floods al other ports with the packet i.e. P2 and P3.
5. The packet sent over P3 doesn't reach destination and S3 doesn't get an acknowledgement.
6. The packet sent through P2 reaches H2 which is connected to the destination MAC address.
7. Once H2 gets the packet, it returns an acknowledgement back to the source i.e S3
8. S3 on receiving the acknowledgement adds an entry to the mapping table: h2 -> p2
9. S3 then forwards the packet through.

Second ping onwards:

1. H1 sends packet to s3 which it receives through port P1.
2. S3 already has the source address. S3 and the destination address in the dictionary. So it just forwards the packet to H2 through the P2 port.

Q2. Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).

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

h1 ping -c100 h2 : 3.916 ms
h1 ping -c100 h8 : 8.836 ms

b. What is the minimum and maximum ping you have observed?

- h1 ping -c100 h2 :
  Max - 10.521 ms
  Min - 1.473 ms

- h1 ping -c100 h8 :
  Max - 14.934 ms
  Min - 5.231 ms

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

Comparing values of h1 ping h2 in task 3 as compared to task 2, we see that task 3 is faster but the difference is not big. The reason for the small difference is that there is only one switch between the nodes hence, the network latency is not that long and the switch also doesnt see any major network congestion.

However, when we see the difference between h1 ping h8, the difference is more clearly visible. The time in task 3 is much smaller. This is because by using the mapping table, only the initial few requests are congested but once the switch has the right mappings, the remaining requests are processed faster leading to an overall lower average. This way switches will avoid network congestion.

Q3. Run “iperf h1 h2” and “iperf h1 h8”.

a. What is the throughput for each case?

H1 -> H2 : Bandwidth is 48.43 Mbits/sec
H1 -> H8 : Bandwidth is 8.51 Mbits/sec

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

The throughput for task 3 is greater. As before, the increased throughput is due to lesser network congestion and reduced latency. Because of the mapping that takes place at each switch, they won't be burdened with huge number of incoming packets as there will not be flooding of packets.