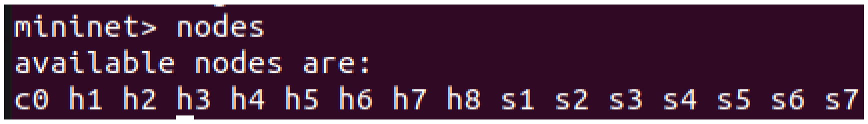
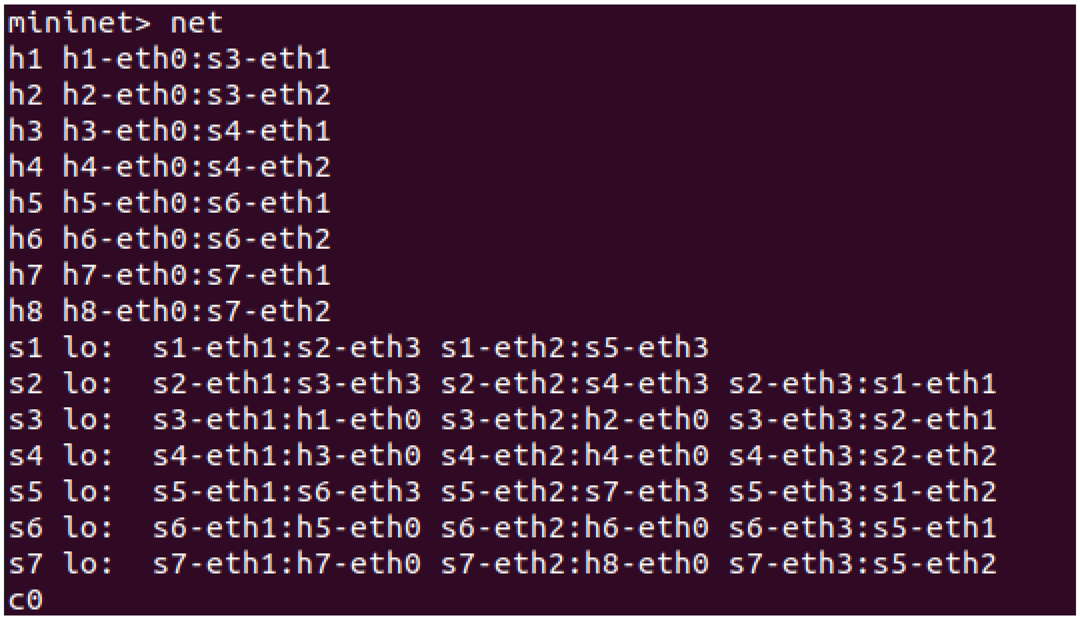
**Step 1:**
**Q1: Output of nodes and ent**

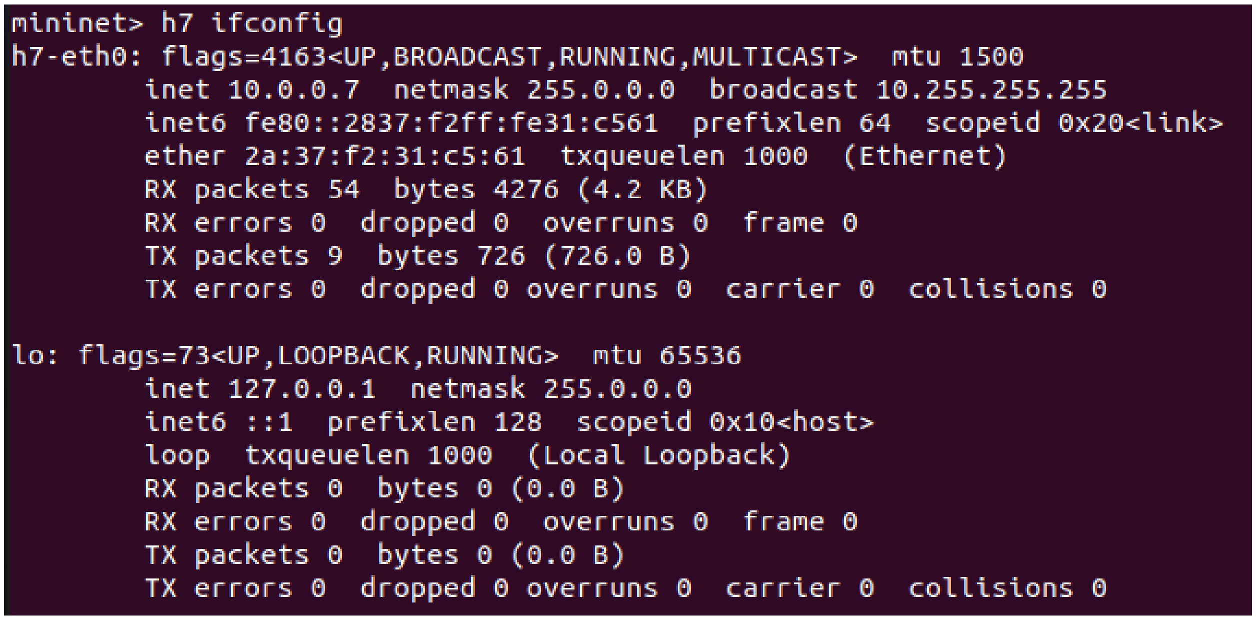


```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1 s2 s3 s4 s5 s6 s7
```



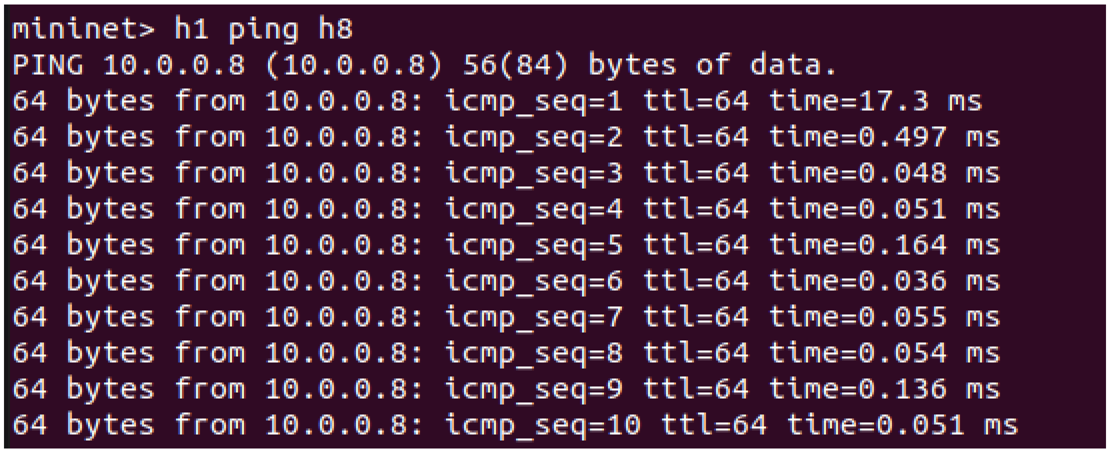
```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0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

**Q2: 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::2837:f2ff:fe31:c561  prefixlen 64  scopeid 0x20<link>
        ether 2a:37:f2:31:c5:61  txqueuelen 1000  (Ethernet)
        RX packets 54  bytes 4276 (4.2 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
```



```
mininet> h1 ping h8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=17.3 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.497 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.051 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.164 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.036 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.055 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.054 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.136 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.051 ms
```

**Step 2:**

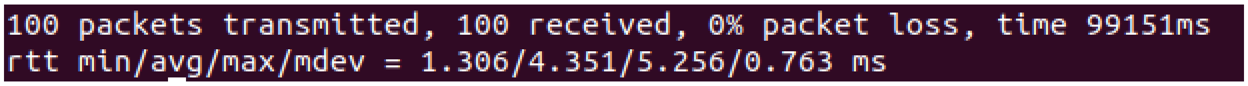
**Q1: The flow is like this:**

**launch -> init -> start_switch -> *waits***
***packet comes in* -> _handle_PacketIn -> act_like_switch (or act_like_hub) -> resend_packet**

**Q2:**

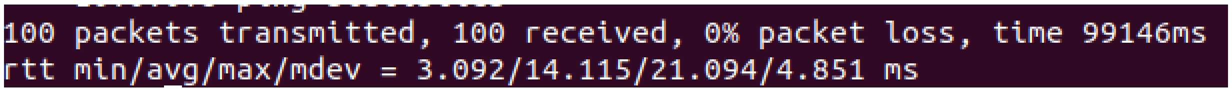
**A/B:**

h1 ping -c100 h2



```
100 packets transmitted, 100 received, 0% packet loss, time 99151ms
rtt min/avg/max/mdev = 1.306/4.351/5.256/0.763 ms
```

h1 ping -c100 h8

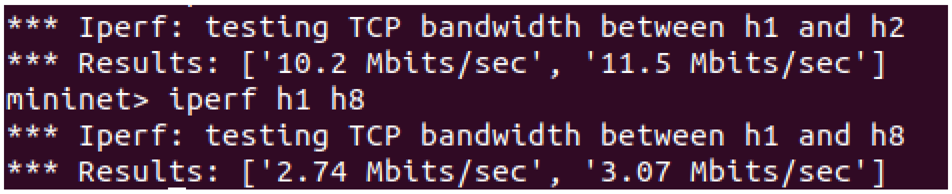


```
100 packets transmitted, 100 received, 0% packet loss, time 99146ms
rtt min/avg/max/mdev = 3.092/14.115/21.094/4.851 ms
```

| Command | Min (ms) | Avg (ms) | Max (ms) |
|---|---|---|---|
| h1 ping -c100 h2 | 1.306 | 4.352 | 5.256 |
| h1 ping -c100 h8 | 3.092 | 14.115 | 21.094 |

**C.** The difference is because h1->h2 only has to go through 1 switch, while h1->h8 has to go through 5 switches. This means that in the latter case, each message spends a lot more time in transit on the network.

**Q3:**



```
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.2 Mbits/sec', '11.5 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.74 Mbits/sec', '3.07 Mbits/sec']
```

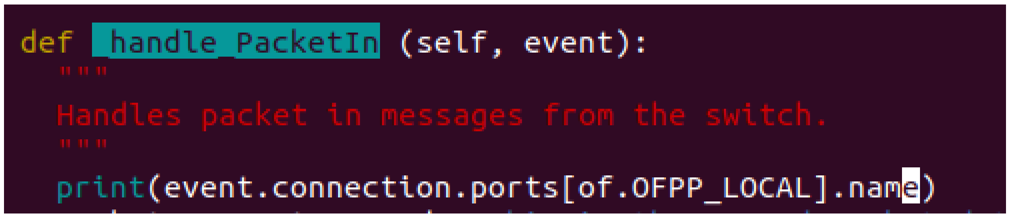
**A:** Iperf is used for testing the bandwidth of different connections

**B:**

| Command | h1 | h2/h8 |
|---|---|---|
| Iperf h1 h2 | 10.2 Mbits/sec | 11.5 Mbits/sec |
| Iperf h1 h8 | 2.74 Mbits/sec | 3.07 Mbits/sec |

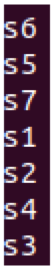
**C:** The difference is due to the fact that there is only 1 switch separating h1 and h2, whereas there are 5 separating h1 and h8. In addition, since we are flooding all ports with all messages there is a lot of extraneous traffic. This means that the more hops a message has to take on the network, the more time it is going to be held up waiting for other traffic to pass.

**Q4:** All of the switches observe traffic since they are dumb and will just forward all messages to all ports. I know this since I modified the _handle_PacketIn function in the of_tutorial.py file to print out its own name when it received a packet.



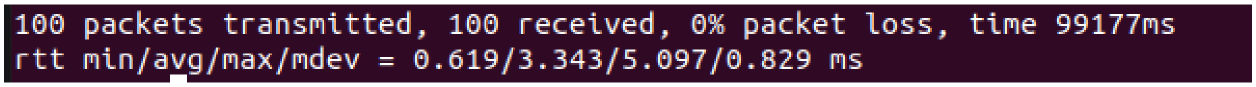
```
def _handle_PacketIn (self, event):
    """
    Handles packet in messages from the switch.
    """
    print(event.connection.ports[of.OFPP_LOCAL].name)
```

Excerpt from log running h1 ping h8



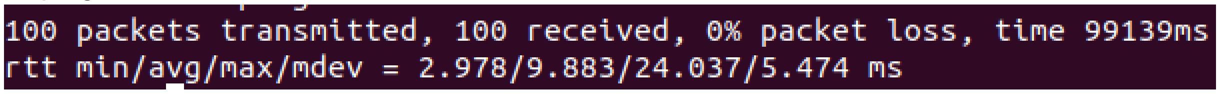
```
s6
s5
s7
s1
s2
s4
s3
```

**Step 3:**
**Q1:** The code works by using the mac_to_port table to help translate mac addresses to port numbers. When a packet comes in, it first looks to see if the src address is in mac_to_port already. If it is not then it adds it to its list of mac addresses since it knows that the direction to that mac address must be through the incoming port. Then it looks to see if the destination address matches a known entry in the mac_to_port table. If the mac address is in the mac_to_port table then it only sends the packet on that port. Otherwise it forwards the packet to all ports except the input one. This means that the switch only learns about where mac addresses actually are when that mac address sends a packet.
**Q2:**
**A/B:**
h1 ping -c100 h2



```
100 packets transmitted, 100 received, 0% packet loss, time 99177ms
rtt min/avg/max/mdev = 0.619/3.343/5.097/0.829 ms
```

h1 ping -c100 h8

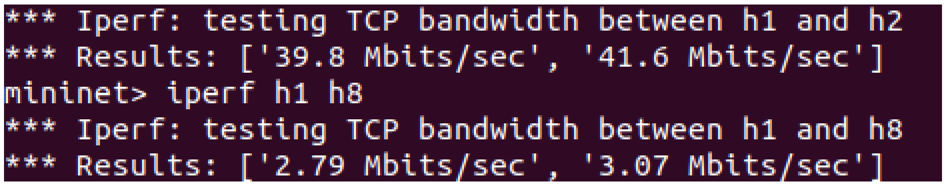


```
100 packets transmitted, 100 received, 0% packet loss, time 99139ms
rtt min/avg/max/mdev = 2.978/9.883/24.037/5.474 ms
```

| Command | Min (ms) | Avg (ms) | Max (ms) |
|---|---|---|---|
| h1 ping -c100 h2 | 0.619 | 3.343 | 5.097 |
| h1 ping -c100 h8 | 2.978 | 9.883 | 24.037 |

**C:** There was a substantial drop in the average round trip time in the second set of tests, especially for the ping from 1 to 8. This is due to the fact that the switches are only sending packets to the necessary places instead of everywhere.

**Q3:**
**A:**



```
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['39.8 Mbits/sec', '41.6 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['2.79 Mbits/sec', '3.07 Mbits/sec']
```

| Command | h1 | h2/h8 |
|---|---|---|
| Iperf h1 h2 | 39.8Mbits/sec | 41.6 Mbits/sec |
| Iperf h1 h8 | 2.79 Mbits/sec | 3.07 Mbits/sec |

**B.** There was a performance difference between task 2 and 3 in the first case, but not the second. This is most likely because the second case was more held back by the number of hops that had to be performed, whereas the first case was more held back by the extra sending that needed to be performed by S3 along its link to S2.