Amanda Boss, Annie Lin, Billie Wei
CS143: Computer Networks
Programming Assignment
October 20, 2014

**Code on Github:** https://github.com/theannielin/drkung143

**Question 1**

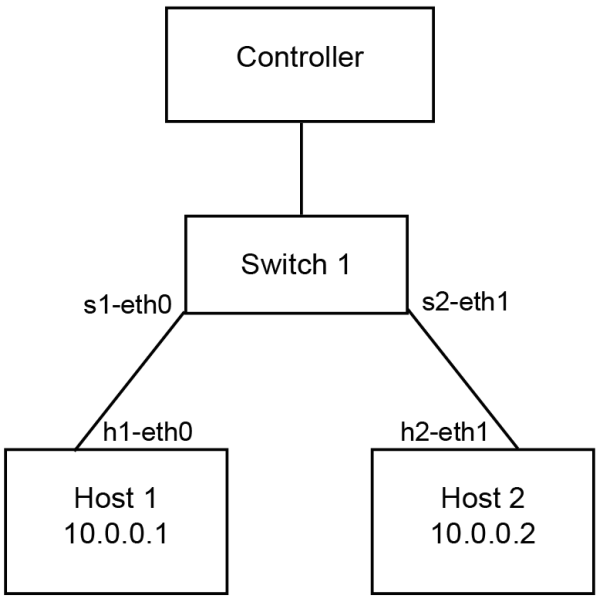
**What did you see? How many ICMP packets are (ping requests) sent out and replied successfully by a Harvard web server?**

We can see that we pinged the Harvard web server. We can see the number of bytes we sent, the time it took, the time to live, and also IP address. We can also see average routing times. 3 packets were sent out and replied to successfully by the Harvard web server.

```
mininet@mininet-vm:~$ ping -c3 www.harvard.edu
PING d1mldapedjoasw.cloudfront.net (54.230.53.72) 56(84) bytes of data.
64 bytes from server-54-230-53-72.jfk6.r.cloudfront.net (54.230.53.72): icmp_req
=1 ttl=63 time=15.4 ms
64 bytes from server-54-230-53-72.jfk6.r.cloudfront.net (54.230.53.72): icmp_req
=2 ttl=63 time=7.65 ms
64 bytes from server-54-230-53-72.jfk6.r.cloudfront.net (54.230.53.72): icmp_req
=3 ttl=63 time=8.29 ms

--- d1mldapedjoasw.cloudfront.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 7.651/10.457/15.426/3.524 ms
mininet@mininet-vm:~$
```

**Question 2**

**Draw a precise diagram of the default network topology (i.e., label nodes and their network interfaces correctly with IP addresses as in the figure from http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Start_Network).**



```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=4.40 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.224 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.053 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.053/1.561/4.407/2.013 ms
```

```
mininet> h2 ping -c3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_req=1 ttl=64 time=2.13 ms
64 bytes from 10.0.0.1: icmp_req=2 ttl=64 time=0.246 ms
64 bytes from 10.0.0.1: icmp_req=3 ttl=64 time=0.055 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.055/0.810/2.130/0.936 ms
```

**Question 3a**

**Does pinging between hosts work? If not, why did it work in Question 2 but not for this one? Explain in 20 words or less.**

Pinging between hosts did not work, with open flow you have a central controller, no way to ping that if the hosts are not directly connected. Basically, the switch flow table is empty.

**Question 3b**

**Explain how you can fix the ping issue in 3a**.

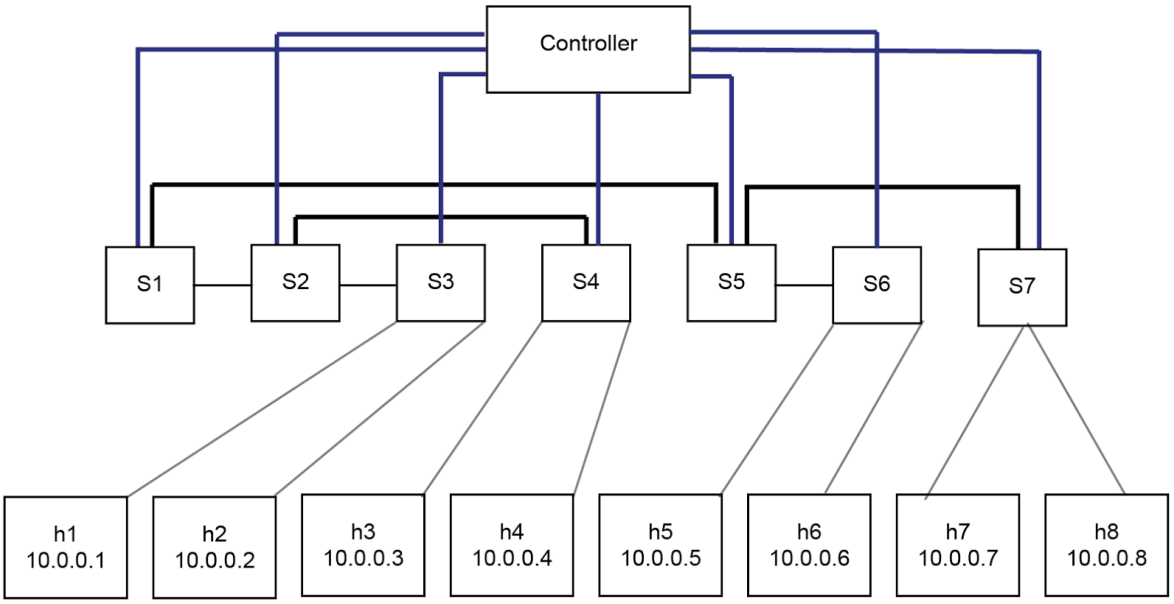
You can use the dpctl command to manually install necessary flows into the switch flow table.

```
mininet@mininet-vm:~$ sudo mn --topo linear,7 --mac --switch ovsk --controller r
emote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (h6, s6) (h7, s7) (s1, s2) (s2, s3)
 (s3, s4) (s4, s5) (s5, s6) (s6, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
```

```
mininet> h1 ping -c3 h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.7 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2008ms
pipe 3
```

**Question 4**

**Draw a precise description of the network topology with IP addresses.**




```
mininet@mininet-vm:~$ sudo mn --topo tree,depth=3,fanout=2 --mac --switch ovsk
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s1, s2)
 (s1, s5) (s2, s3) (s2, s4) (s5, s6) (s5, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
```

**Question 5a**

**You should see two types of packets in the tcpdump xterms of h2 and h3. What are the two types? Draw a path on the diagram of ping packets (both request and response packets).**

The two types are ARP and ICMP packets. The packet transmission is represented by the arrow below

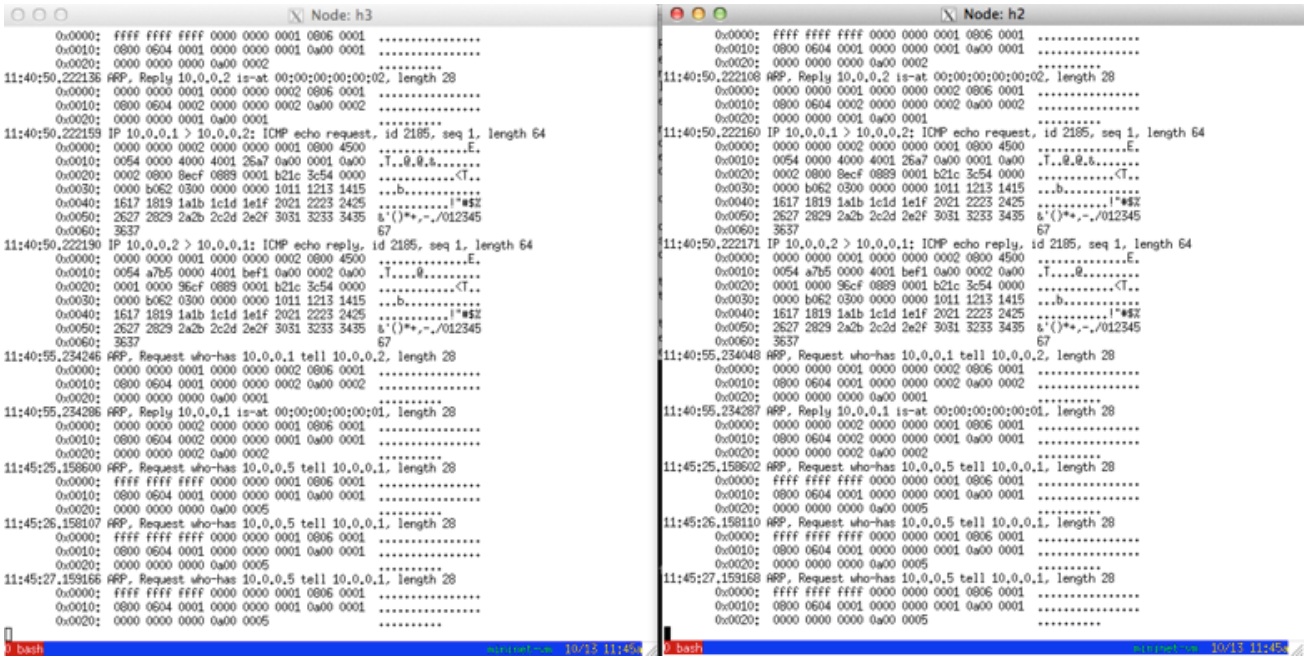
# h1 → h2

```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.354 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.354/0.354/0.354/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.163 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.163/0.163/0.163/0.000 ms
```

**Question 5b**

**Now, see what happens when a non-existent host doesn't reply (e.g. ping -c1 10.0.0.5) How many packets and of what type do you see in the tcpdump xterms of h2 and h3?**

5 ARP requests sent out to h2 and h3. h1 console says 1 packet transmitted, zero received.




```
root@mininet-vm:~# ping -c1 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

**Question 5c**

**What sort of speed are you seeing? The bandwidth of the virtual links in Mininet are very fast, so that is not a reason for slow speeds. Can you give a couple reasons based on what you now know about OpenFlow and this hub why the speeds are slow?**

We're seeing speeds of '237 Mbits/sec', '237 Mbits/sec'. I think the speeds might be so slow because OpenFlow can be slowing down - using a software defined paradigm. There is a tradeoff between getting control using SDN - give a little speed with that

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
waiting for iperf to start up...*** Results: ['237 Mbits/sec', '237 Mbits/sec']
mininet>
```

**Question 6**

**What did you see? Examine the code of l2_learning controller available at /pox/pox/forwarding/l2_learning.py. Explain the behavior of this controller.**

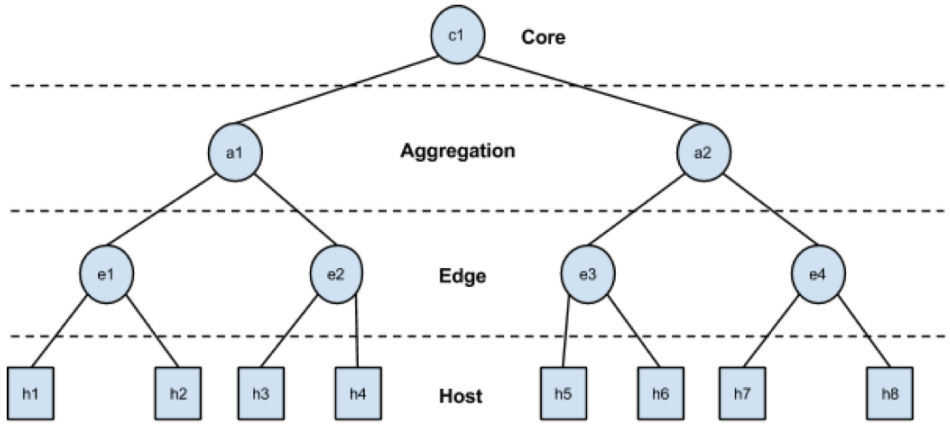
We saw less arp requests (no flooding) because of the behavior of the l2_learning controller . The l2_learning controller is able to remember the location of hosts after initial broadcasts, therefore reducing the need to flood the topology with packets during successive pings because the controller remembers which port is connected to which host.

```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=8.33 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 8.334/8.334/8.334/0.000 ms
root@mininet-vm:~#
```

**Question 7**

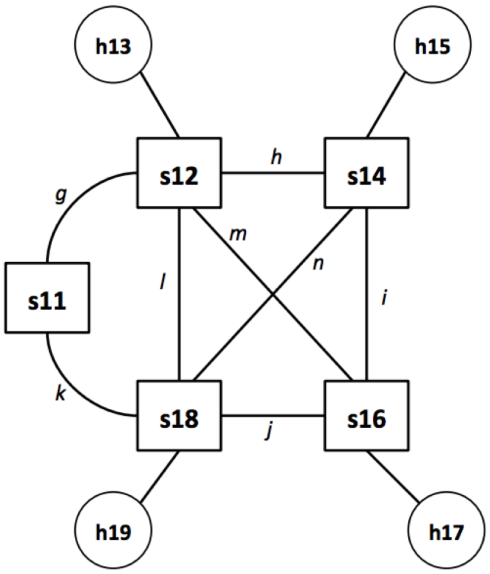
**Create the tree-based datacenter topology described above.**

We created our custom topology in CustomTopo.py. We built it starting from the core and fanning out twice from each subsequent layer of switches using three for-loops.




**Question 8**

**Write a function that reads the policy file and update the _handle_ConnectionUp function. The function should install rules in the OpenFlow switch that drop packets whenever a matching src/dst MAC address (for any of the listed MAC pairs) enters the switch. (Note: make sure that you handle the conflicts carefully.)**



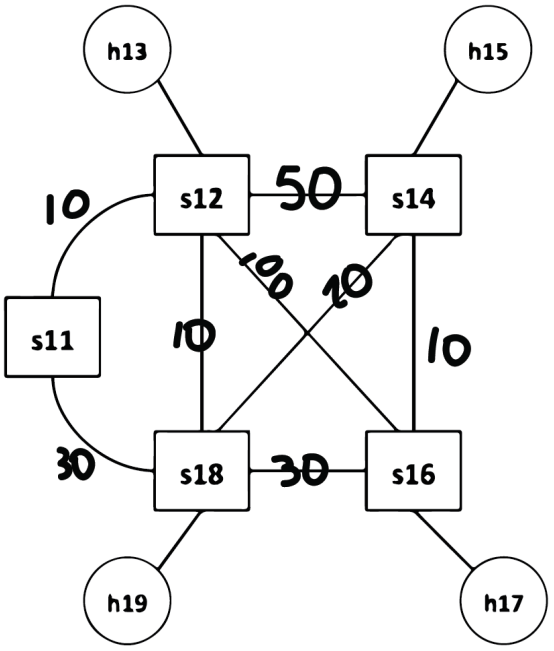

| Link | Delay (msec) |
|---|---|
| $g$ | 10 |
| $h$ | 50 |
| $i$ | 10 |
| $j$ | 30 |
| $k$ | 30 |
| $l$ | 10 |
| $m$ | 100 |
| $n$ | 20 |

**Question 9a**
**Create a custom topology depicted in the figure above with the corresponding delays.**

**Question 9b**
**Manually work out shortest paths for each host with Dijkstra's algorithm based on the network topology and delays in Figure 1.**
To manually work out the shortest paths for each host with Dijkstra's algorithm, we first labeled each link in the topology with our given link delays.

For example, let's work out the shortest path from h19 from h13. First, h13 connects to s12, with delay of 0 ms. Then at s12, we have four possible paths each with delay of 10, 10, 100, and 50 ms. The algorithm should make note of the possible paths and then select one with the lowest delay.

In this case, we have two paths with delay 10 ms - so imagine that we first test the path from s12 → s11. If we repeat path testing, we'll discover that the path from s12 → s11 → s18 has a total delay of 40 ms, which is less than the 100 and 50 ms paths from s12 → s14 and s12 → s16 respectively, but more than the delay in the path from s12 → s18. Therefore, we should test the path from s12 → s18, and we'll find that the shortest path from h19 → h13 is from s12 → s18. For further iterations of the algorithm, the flow table should have recorded the paths we tested initially and delay values of those paths such that upon each further iteration, the necessity of testing paths diminishes.




**Shortest Path**

| | h13 | h15 | h19 | h17 |
|---|---|---|---|---|
| **h13** | --- | S12 - S18 - S14 | S12 - S18 | S12 - S18 - S16 |
| **h15** | S14 - S18 - S12 | --- | S14 - S18 | S14 - S16 |
| **h19** | S18 - S12 | S18 - S14 | --- | S18 - S16 |
| **h17** | S16 - S18 - S12 | S16 - S14 | S16 - S18 | --- |

**Total Delay of Shortest Path (msec)**

| | h13 | h15 | h19 | h17 |
|---|---|---|---|---|
| **h13** | --- | 30 | 10 | 40 |
| **h15** | 30 | --- | 20 | 10 |
| **h19** | 10 | 20 | --- | 30 |
| **h17** | 40 | 10 | 30 | --- |

**Question 9c**

**Program your controller to use Dijkstra's algorithm to find the shortest path between any pairs of nodes given a topology and delay table. To do this, your controller should read in a delay table given in delay.csv file, and then program the controller to use this information in order to run Dijkstra's algorithm and then install the resulting flows in the flow table. Your code should be general in the sense that it can take any delay values so that we may test your implementation**

**using new delay values. Be sure to check your result with what you have found in part b.**