Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001

Assignment 8 Final-Project

1. Introduction

I have built an SDN so each switches and hosts in same subnet can send data between each other efficiently. I have implemented an algorithm to install rules for switches so that each data can go through shortest path to the destination. Also, I have implemented a load balancer which uses virtual IP so that client request can be evenly distributed to server hosts.

2. Proposed Solution & Architecture

I have implemented shortest path method using Bellman-Ford algorithm. There are many other faster algorithms such as Dijkstra's algorithm and Johnson's algorithm to find shortest paths between each pair of nodes in a graph but I have chosen Bellman-Ford for simplicity in implementation.

For load balancer, I have made the rules in two layers so that connection-specific packets and packets to the real hosts can go through the switches without controller interruption, while new connection to a virtual IP can be managed by the controller. I have implemented two layers rules by assign higher priority to packets to real hosts and connection-specific packets.

3. Algorithms

Bellman-Ford algorithm is simple but powerful algorithm which recalculates

distance between source node and all other nodes for O(V) times so that we can achieve shortest path between each node. Asymptotic running time is O(VE), which is not as efficient as Dijkstra's but Bellman-Ford algorithm can be applied to graphs with negative edge weight thus more versatile.

Below is pseudo-code for Bellman-Ford algorithm from (Intro. to Algorithms, CLRS)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

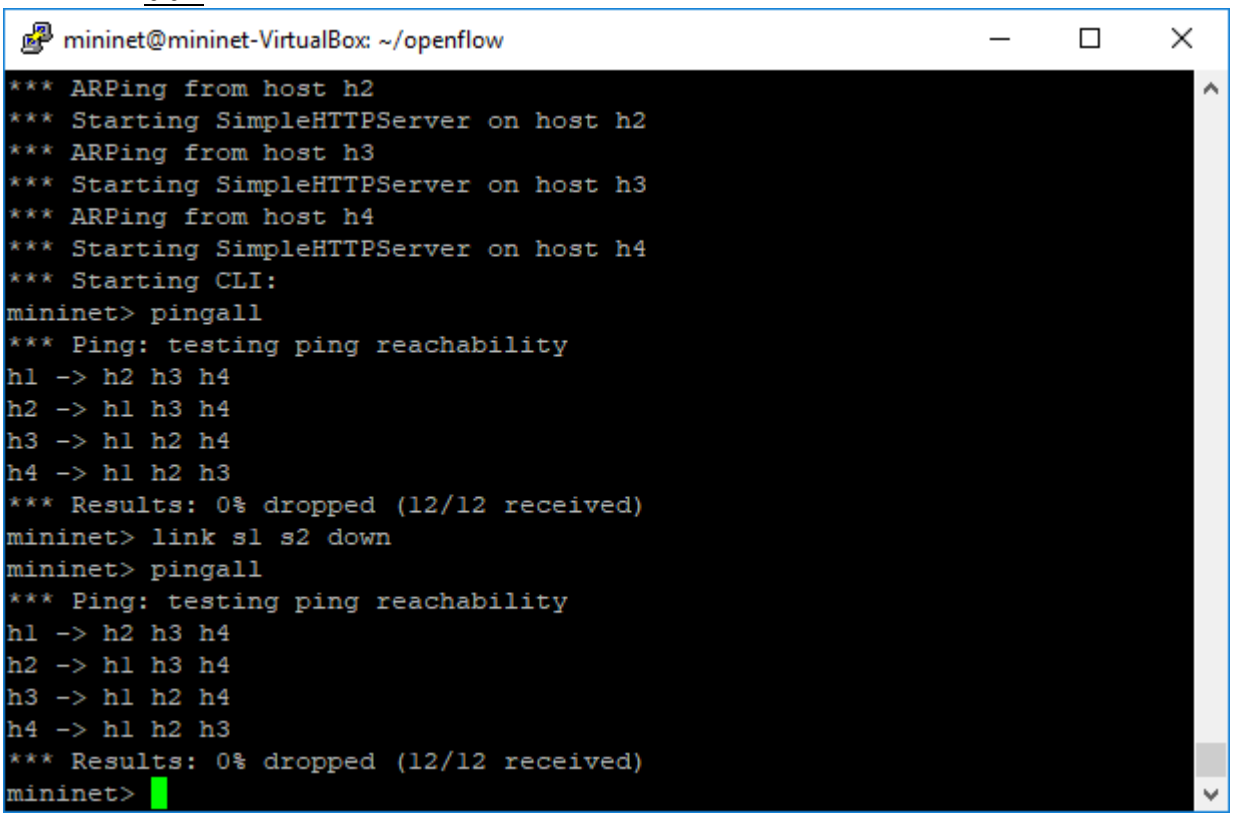
```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

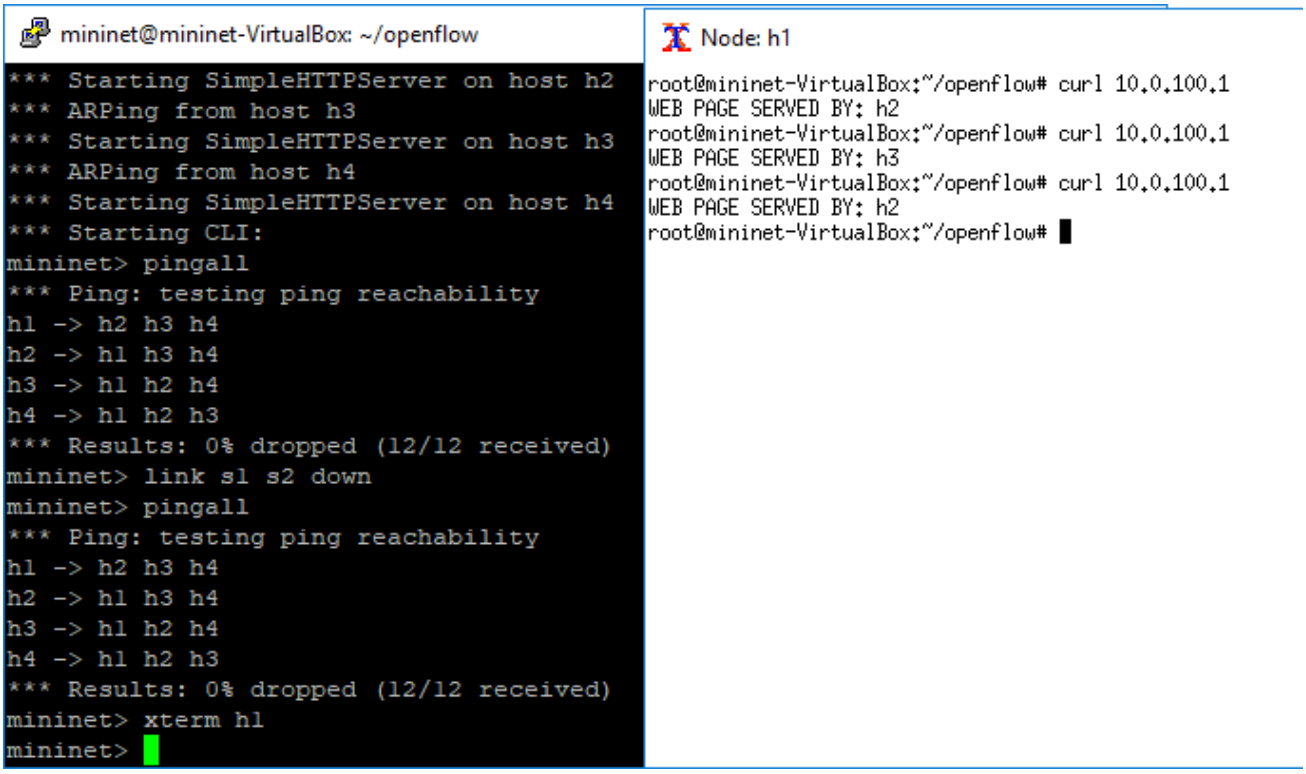
4. Simulations & Results

I have tried various simulation to all the network topologies provided including someloops, mesh, triangle by removing edge between switches and calling pingall such as below picture.

Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001



For simulating load balancer I have opened new terminal for the host and made web request such ac curl to virtual IP.



Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001

5. Result & Conclusion

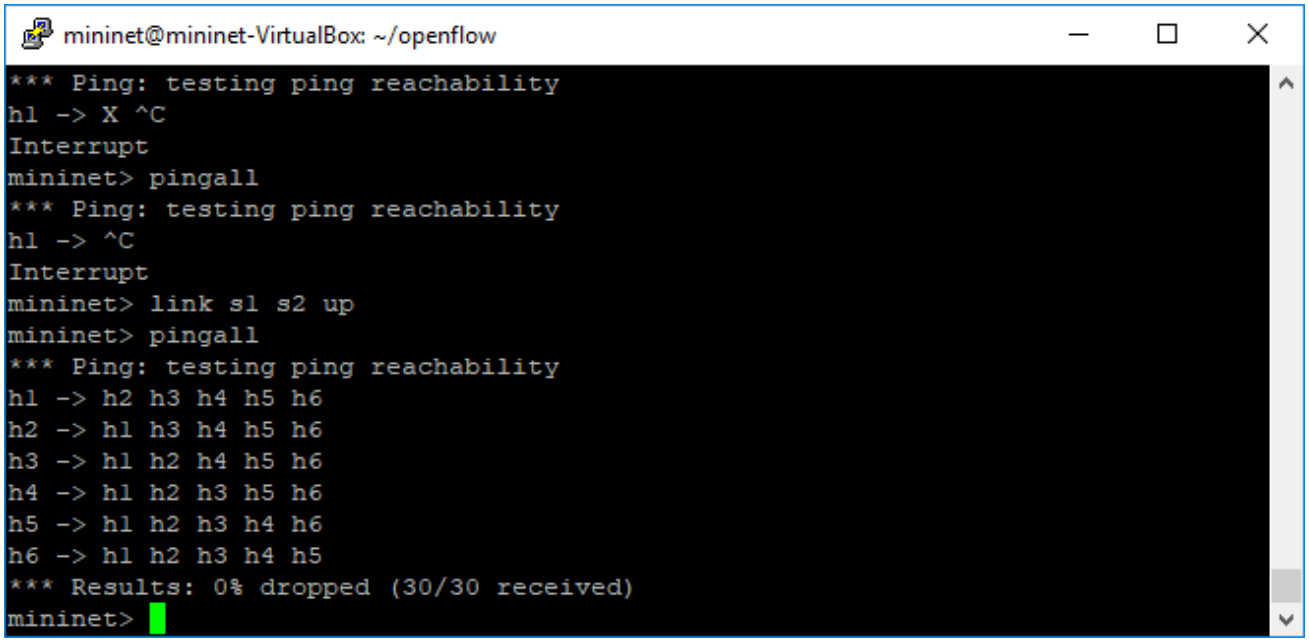
Every implementation worked correctly. It seemed simple and easy when I just read about the work, but it was quite tricky to make a real implementation to work. It was much difficult to debug because network keeps changing. However, it was really good to make a real feeling of what network is.

Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001

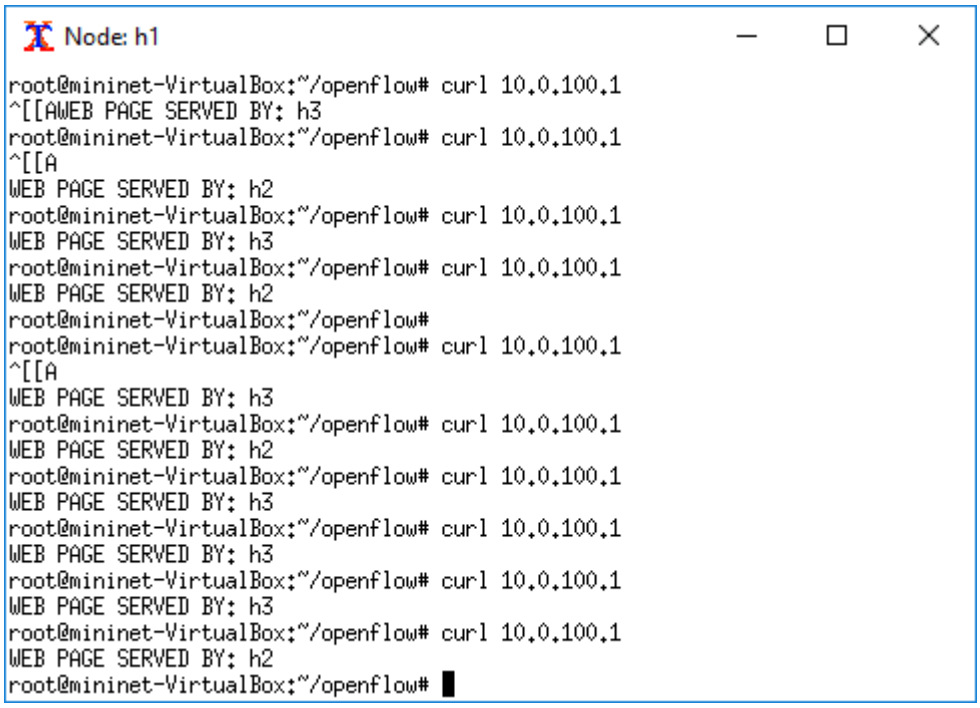
Tests

I have tested and debugged the algorithm using

1. Mininet commands such as pingall, link up& down

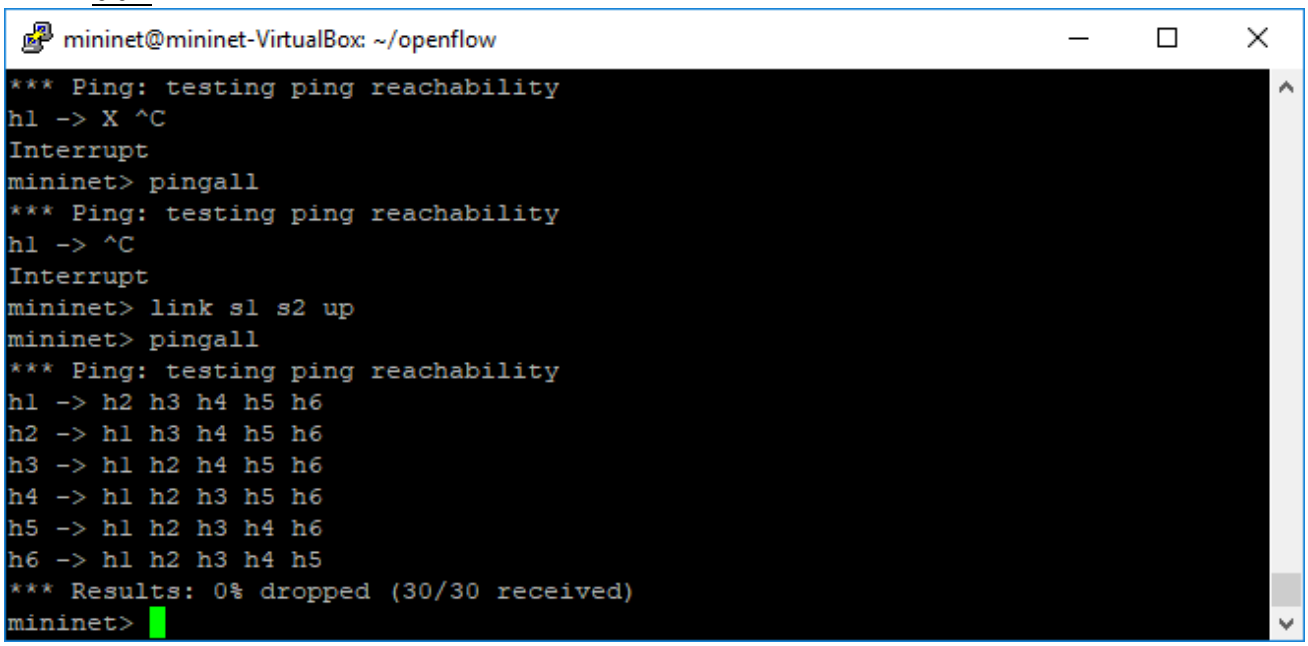


2. I used xterm to send ping and curl from one host to the other



3. Inserted logs between source code

Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001



Name: Chun, Sungwoo (sc7408) Date: 05/21/2019
Section: 001

Vulnerabilities

1. Unknown port number

   : There exist unknown port number -2, which appears on log but can't be identified in mininet

```
19:52:06.666 INFO [ShortestPathSwitching:Topology Updates] Link s3:-2 -> host updated
19:52:06.675 INFO [ShortestPathSwitching:Topology Updates] Link s4:0 -> host updated
19:52:06.707 INFO [ShortestPathSwitching:Topology Updates] Link s3:1 -> host updated
19:52:06.708 INFO [ShortestPathSwitching:Topology Updates] Link s3:4 -> host updated
19:52:06.708 INFO [ShortestPathSwitching:Topology Updates] Link s3:2 -> host updated
19:52:06.710 INFO [ShortestPathSwitching:Topology Updates] Link s3:3 -> host updated
```

   dump result

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=6199>
<Host h2: h2-eth0:10.0.0.2 pid=6200>
<Host h3: h3-eth0:10.0.0.3 pid=6201>
<Host h4: h4-eth0:10.0.0.4 pid=6203>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=6207>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None pid=6214>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None pid=6219>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None,s4-eth4:None pid=6224>
<RemoteController c0: 127.0.0.1:6633 pid=6192>
```

2. Phantom host

   : Phantom host with weird number frequently disappears. It only disappears but can't be identified when it was created. It doesn't have IP address nor any other attributes. I have inserted if clause to sort it out when calculating shortest path

```
19:51:06.037 INFO [ShortestPathSwitching:Scheduled-2] Host h51260762254292 is no longer attached to a switch
19:51:06.174 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h2 added
19:51:06.176 INFO [ShortestPathSwitching:New I/O server worker #2-2] address 167772162 rule...
19:51:06.178 INFO [ShortestPathSwitching:New I/O server worker #2-2] installing h2 rule...
19:51:06.180 INFO [ShortestPathSwitching:New I/O server worker #2-2] id 2 rule...
19:51:07.255 INFO [ShortestPathSwitching:New I/O server worker #2-3] Host h3 added
19:51:07.258 INFO [ShortestPathSwitching:New I/O server worker #2-3] address 167772163 rule...
```