CSCI-GA2262 Data Communications & Networks
Final Project: Software-Defined Networking

Yulin Shen
ys2542@nyu.edu

1 Introduction

Software-defined networking (SDN) technology is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring making it more like cloud computing than traditional network management.[5] Our SDN design is associated with the OpenFlow protocol, which could give access to the forwarding plane of a network switch or router over the network.[4] Then, we use Floodlight Java-based SDN controller to implement our specific network design. We can use Mininet network emulator to build the specific architectures in the SDN network design. Also, it could emulate the package sent and received from hosts to hosts.

In the project, we have 2 main tasks. The first one is to install rules in the layer-3 routing application to forward packets from hosts to hosts with the shortest path. The second one is to use a distributed load balance application to distribute TCP connections from client requests among a collection of hosts with virtual IP addresses.

2 Related Work

2.1 Layer–3 routing application

In the seven-layer OSI model of computer networking, the network layer is layer-3.[1] It is responsible for packet forwarding including routing through intermediate routers.[1] The network layer provides the means of transferring variable-length network packets from a source to a destination host via one or more networks.[1] Within the service layering semantics of the OSI network architecture, the network layer responds to service requests from the transport layer and issues service requests to the data link layer.[1]

2.2 Bellman-Ford Algorithm

The Bellman-Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted graph.[2] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.[2]

2.3 Load Balancing

In computing, load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.[3] Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.[3]

3 Methodology

3.1 Modify Graph Topology

3.1.1 Connect a Host with a Switch

When we need to connect a host with a switch, the program will call function deviceAdded. In the function, we need to remove all old tables and then add all new tables, because the new connections with hosts and switches could change the shortest paths from each node dramatically.

3.1.2 Disconnect a Host with a Switch

When we need to disconnect a host with a switch, the program will call function deviceRemoved. In the function, we need to remove all old tables of the disconnected host for all switches, but we do not need to update other tables of other hosts.

3.1.3 Move Connection of a Host and a Switch

When we need to adjust a host to connect with another switch, the program will call function deviceMoved. Actually, the logic in the case is like we remove the host and the switch at first, then, we reconnect the host with another switch. So, we should update all tables including removing and adding.

3.1.4 Add a Switch

When we need to add a switch, the program will call function switchAdded. In the function, we need to update all tables including removing and adding.

3.1.5 Remove a Switch

When we need to remove a switch, the program will call function switchRemoved. In the function, we need to update all tables including removing and adding.

3.1.6 Connect Switch with Switch

After modifying switches, we could connect switches or disconnect them. The function linkDiscoveryUpdate could help us to finish it. In the function, we still need to remove old tables, and add new tables.

3.2 Bellman-Ford Algorithm

I use Bellman-Ford algorithm to find the shortest path from hosts to hosts. The algorithm pseudocode is descripted below.[2] Because our graph topology will not have a negative-weight cycle, I neglect the part 3 in the original version.

Algorithm 1 Bellman-Ford algorithm

```
1:  procedure BellmanFord(list vertices, list edges, vertex source)
2:      for each vertex v in vertices:                          ▷ Initialize graph
3:          distance[v] = inf                  ▷ Initialize the distance to all vertices to infinity
4:          predecessor[v] = null                              ▷ And having a null predecessor
5:      distance[source] = 0          ▷ The distance from the source to itself is, of course, zero
6:
7:      for i from 1 to size(vertices) - 1:
8:          for each edge (u, v) with weight w in edges:   ▷ Relax edges repeatedly for all vertices
9:              if distance[u] + w <distance[v]:                          ▷ Find shorter path
10:                 distance[v] = distance[u] + w                        ▷ Update distance
11:                 predecessor[v] = u                                   ▷ Update predecessor
12:
13:     return distance[], predecessor[]
```

4 Implementation and Simulation

4.1 Install a rule for a host and a switch

At first, we need to check whether this host is connected to the switch. If it is not, it is not necessary to do anything. But if so, we need to use SwitchCommands to install a new rule for this switch.

4.2 Install Rules for All Hosts and All Switches

I use a hash table to store the shortest path searched from the Bellman-Ford algorithm. Because the node is in a graph, so it is still connected with a series of nodes. Then, we can use getHosts and getSwitches to get a series of hosts and switches. We can use for loops to install rules for them pair by pair. And I still check the connection of them at first. Then, for each host, we can install rules for them one by one by the hash map. Finally, all rules for all switches are updated.

4.3 Remove a Rule for a Host and a Switch

The implementation of the function is very similar with install a rule for a host and a switch. We just need to use SwitchCommands to remove rules of the switch instead of installing.

4.4 Remove Rules for a Host and All Switches

After we can remove a rule for a host and a switch, then we can use getSwitches and a for loop to remove rules of all switches with this host.

4.5 Remove Rules for All Hosts and All Switches

After we can remove rules for a host and all switches, then we can use getHosts and a for loop to remove rules of all switches with all hosts.

4.6 Search Shortest Path for Each Switch

As I said above, I use Bellman-Ford algorithm to find the best routes from one node to all other nodes. Because our graph is undirected with equal weight, so I assume all weights are 1. As the pseudocode, I use a hash table for distances, and another hash table for predecessors. Then, we need to initialize them with a very large integer or 0 and a null for object link. Then, we can relax edges repeatly to update distances and predecessors if we find a shorter path then previous. Finally, we can run the algorithms for all nodes in the graph to get a new table.

5 Testing and Debugging

At first we need to run the Floodlight controller program with our new configuration, and Mininet emulator to build a network graph topology. The Mininet emulator provide us a lot of different kinds of topology like triangle, single, tree, someloops, and so on. After building a topology of network, we can debug and test our functionality by the ways below.

The command link in the Mininet emulator window could connect and disconnect the switch and the host. So, when we link up two switches or link down two switches by the commands link [switch name] [switch name] up or link [switch name] [switch name] down, we could test the functionality of linkDiscoveryUpdate. Link a switch and a host up or down by the commands link [switch name] [host name] up or link [switch name] [host name] down could help us to test the functionality of deviceAdded and deviceRemoved. The command sudo ovs-vsctl del-br [switch name] in the general terminal window could call the function SwitchRemoved to help us to remove a switch, but it is hard to add it back. The process of building topology in the Mininet emulator could help us to test the functionality of SwitchAdded. After doing the actions above, we can use the command sudo ovs-ofctl -O OpenFlowl3 dump-flows [switch name] in the general terminal window to see the table for the switch. Then, we can check whether the package is sent in a shortest path from the original host to the target host by all tables. For load balancing, we can use [host name] curl [host name] to issue web request. Then, we can use tcpdump -v -n -i h[host name] -eth0 in a general terminal window to check which package the host is sending and receiving.

6 Results and Discussion

When I use Mininet emulator to build a someloops topology, we can see from the figure 1 below that all switches with hosts are successfully built and linked. So, we can think SwitchAdded function works right now. When I use the command to remove a switch s1 in the figure 2, we can see if I try to send the packet from h1 to h4, it shows unreachable now, because h1 is connected to nothing now. The figure 3 shows us when h1 sends the packets to h4, the path is h1-s1-s6-h4. We can check my our eyes in the someloops topology, it is exactly the shortest path.

When I use the command link s1 s6 down before h1 ping h4, we can see from figure 4 that the shortest path is changed to h1-s1-s2-s3-s6-h4. We also can check that by our eyes to see it is a updated shortest path. Then I use the command link s1 s6 up, two switches are connected together again, so the shortest path return to the old version in the figure 5.

When I use the command link s1 h1 down before h1 ping h4, we can see the Mininet shows unreachable now in the figure 6, because h1 is connected to nothing now. Then, I use the command link s1 h1 up, they are connected together again. Now, we can see h1 can send packet to h4 in the figure 7.

Figure 1: add switches

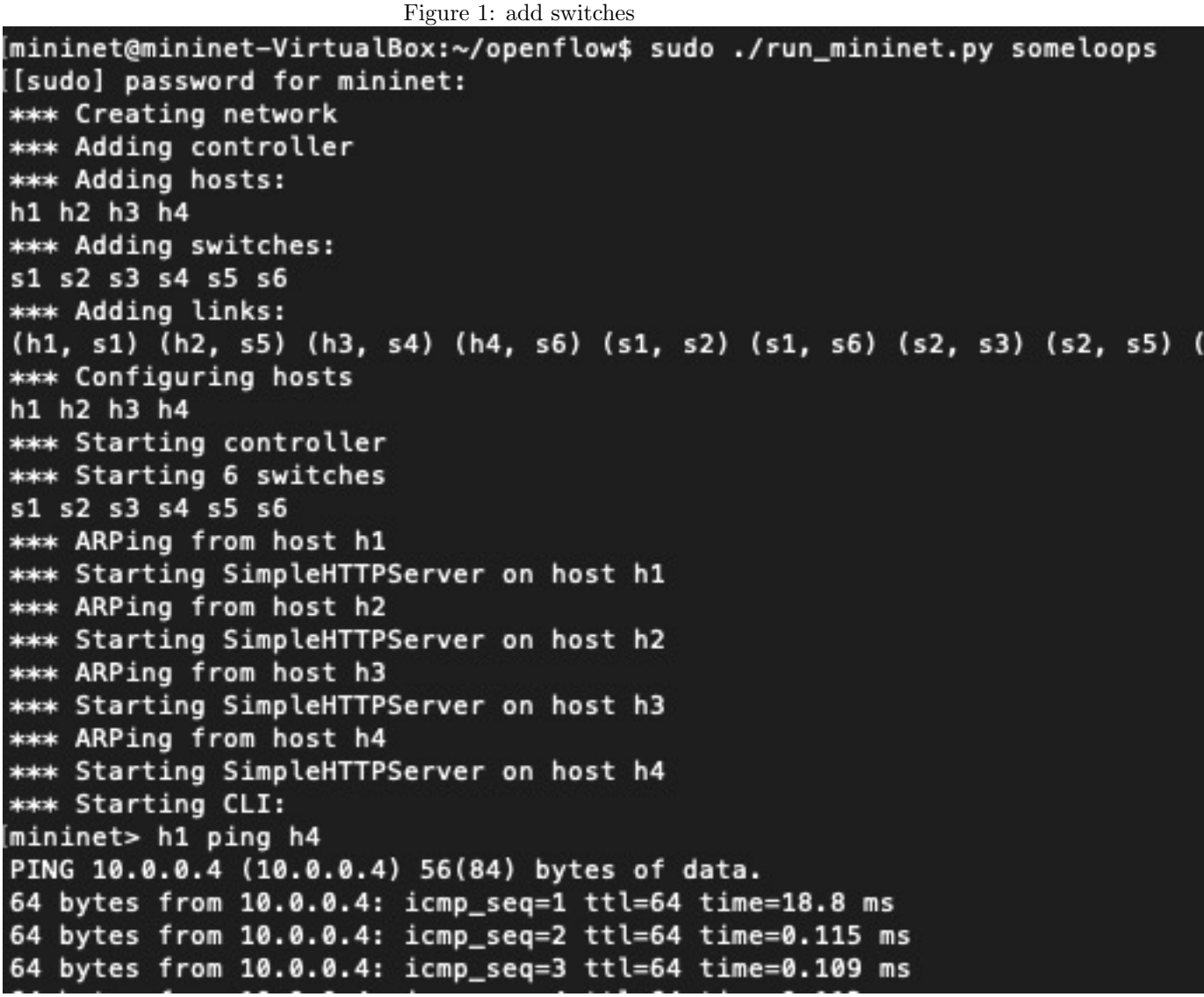


Figure 2: remove a switch

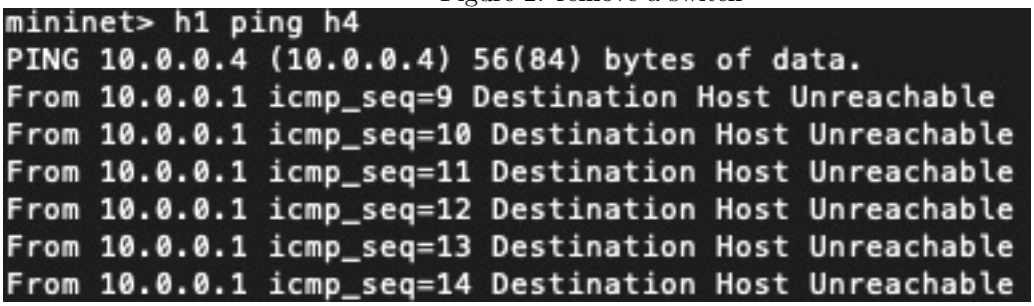


Figure 3: shortest path table

```
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for mininet:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=15.946s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=15.841s, table=0, n_packets=15, n_bytes=1470, prior
 cookie=0x0, duration=15.917s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=15.843s, table=0, n_packets=15, n_bytes=1470, prior
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s2
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=17.778s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=17.73s, table=0, n_packets=0, n_bytes=0, priority=1
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s3
OFPST_FLOW reply (OF1.3) (xid=0x2):
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s4
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=26.404s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=26.316s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=26.361s, table=0, n_packets=0, n_bytes=0, priority=
 cookie=0x0, duration=26.319s, table=0, n_packets=0, n_bytes=0, priority=
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s5
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=0.596s, table=0, n_packets=0, n_bytes=0, priority=1
 cookie=0x0, duration=0.589s, table=0, n_packets=0, n_bytes=0, priority=1
 cookie=0x0, duration=0.641s, table=0, n_packets=0, n_bytes=0, priority=1
 cookie=0x0, duration=0.687s, table=0, n_packets=0, n_bytes=0, priority=1
mininet@mininet-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s6
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=2.379s, table=0, n_packets=0, n_bytes=0, priority=1
 cookie=0x0, duration=2.272s, table=0, n_packets=2, n_bytes=196, priority
 cookie=0x0, duration=2.273s, table=0, n_packets=2, n_bytes=196, priority
 cookie=0x0, duration=2.32s, table=0, n_packets=0, n_bytes=0, priority=1,
```

Figure 4: link s1 s6 down

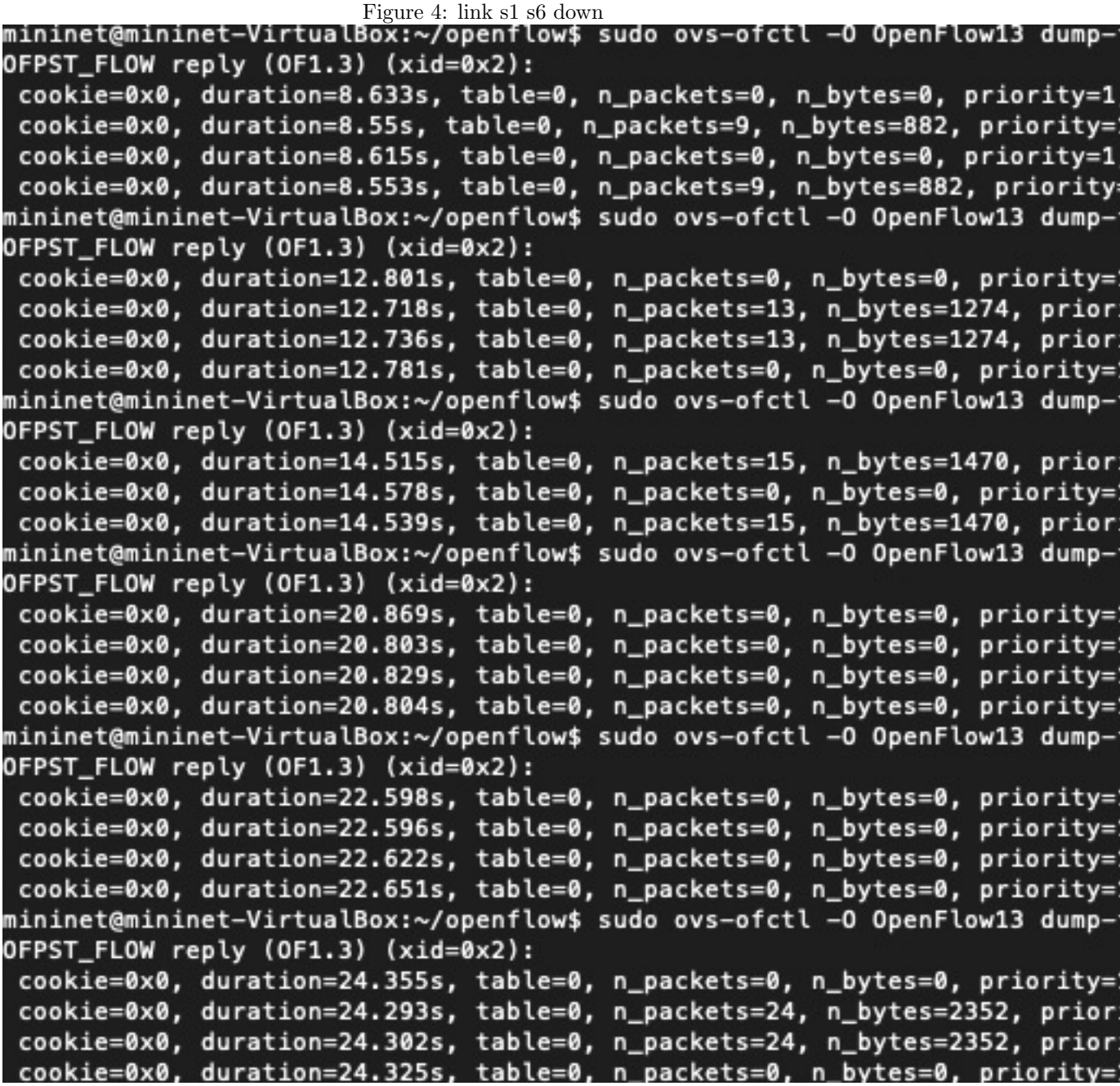


Figure 5: link s1 s6 up

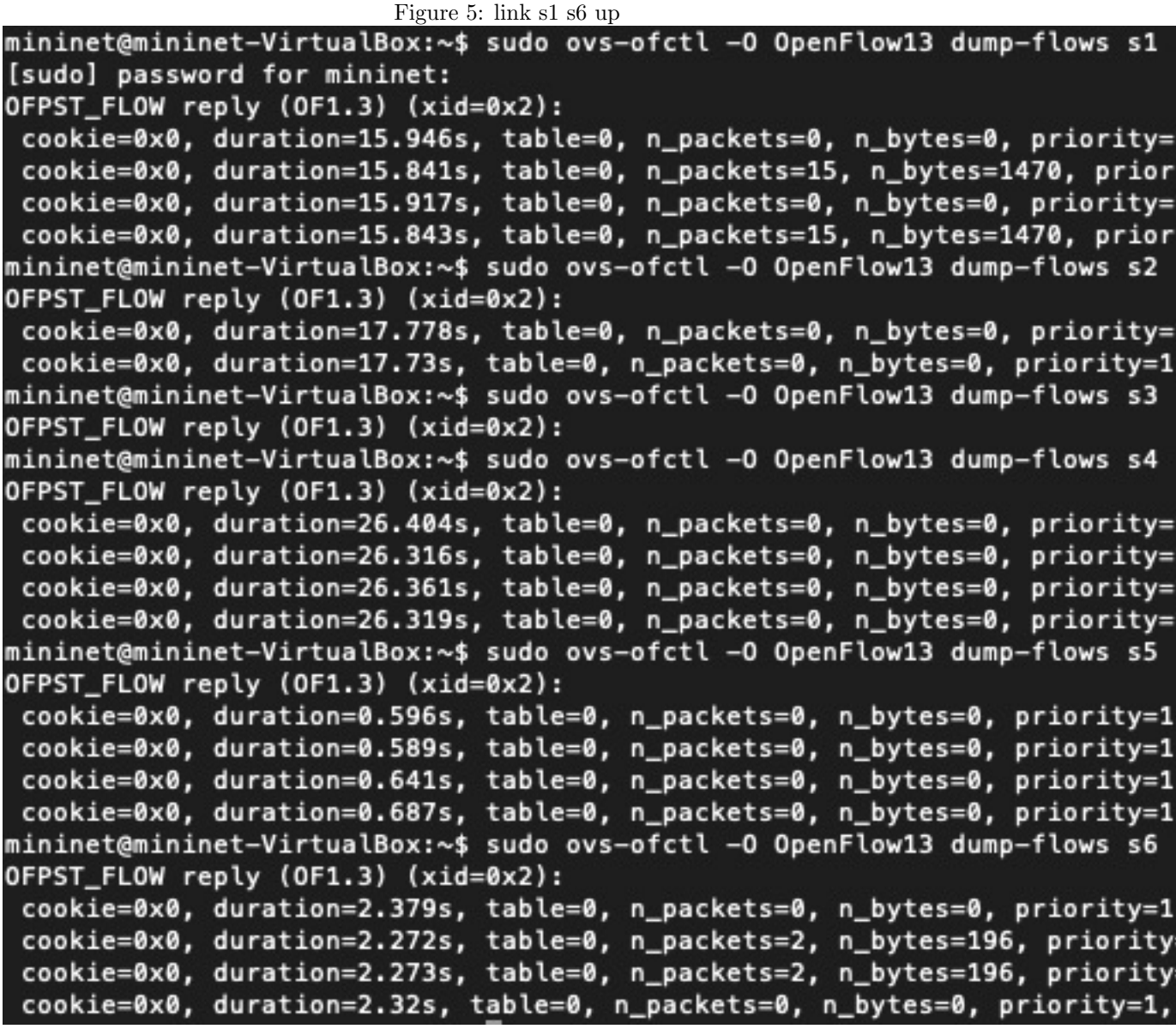


Figure 6: link s1 h1 down

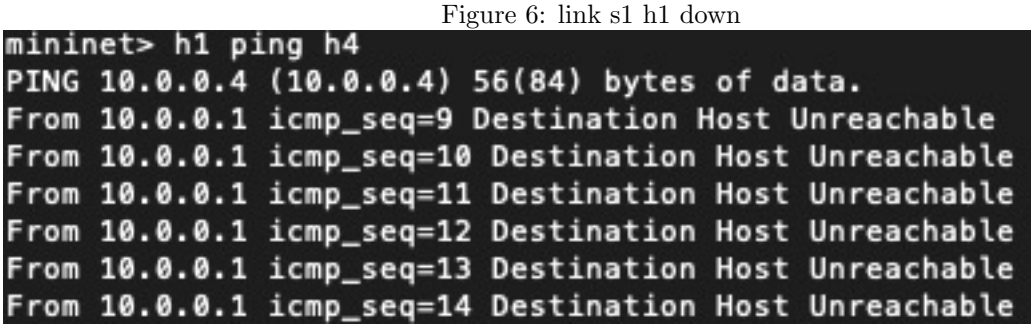
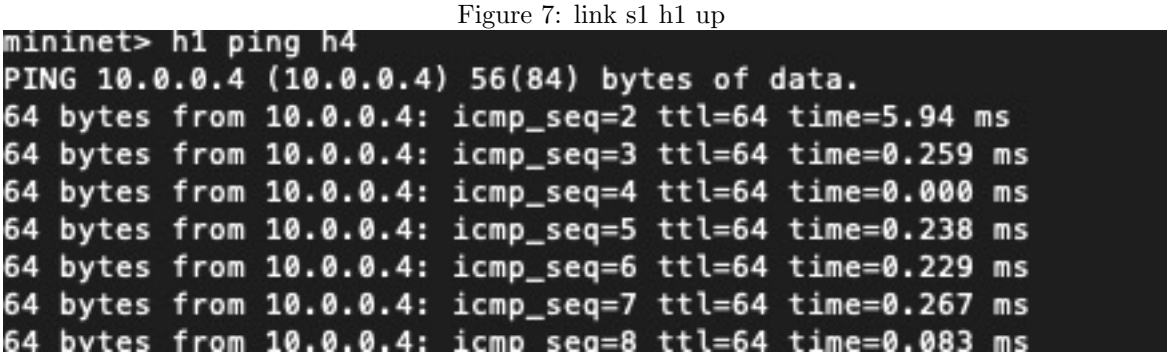


Figure 7: link s1 h1 up



The results show me the program works right, however, it still has some vulnerabilities. The weights in reality are impossibly all equal. Also, I can only send packet from a host to another host one by one, however, in reality, we can send packet from hosts to hosts at same time. The tables may be updated to some problems. Also, the simple SDN program does not concern about the safety issue.

7 Conclusion

SDN could help us to design our networks. We can build the real network based on the result in the program. In the project, we fulfill network layer with modifying switches, hosts, and sending packets in shortest path. Also, the program gives us the idea to distributes TCP connections. In the future, we can come up with a lot of innovative ideas. But we cannot directly build a network of the design, SDN is a good idea to help us to check the idea, and do some tests. Then, we can prove the idea is good.

References

[1] Jeff Tyson. How lan switches work. ttps://computer.howstuffworks.com/lan-switch15.htm, May 2019.

[2] Wikipedia. Bellman–ford algorithm. https://en.wikipedia.org/wiki/Bellman\OT1\textendashFord_algorithm, May 2019.

[3] Wikipedia. Load balancing. https://en.wikipedia.org/wiki/Load_balancing_(computing), May 2019.

[4] Wikipedia. Openflow. https://en.wikipedia.org/wiki/OpenFlow, May 2019.

[5] Wikipedia. Software-defined networking. https://en.wikipedia.org/wiki/Software-defined_networking, May 2019.