50.012 Networks – Project

Table of Contents

[1. Introduction 1](#_Toc529997956)

[1.1 SDN in Data Centre Networks 1](#_Toc529997957)

[1.2 Literature review 2](#_Toc529997958)

[2. Design 2](#_Toc529997959)

[2.1 Topology: Fan Network 2](#_Toc529997960)

[2.2 Topology: Clos Network 3](#_Toc529997961)

[3. Implementation 3](#_Toc529997962)

[3.1 Controller: POX 4](#_Toc529997963)

[3.2 POX components 4](#_Toc529997964)

[3.3 Virtual network & Topology editor: Mininet and MiniEdit 4](#_Toc529997965)

[4. Steps-to-reproduce 5](#_Toc529997966)

[4.1 Pre-requisites 5](#_Toc529997967)

[4.2 Running the Data Centre topology in MiniEdit 5](#_Toc529997968)

[4.3 Running the POX controller 6](#_Toc529997969)

[4.4 Simulating load balancing 7](#_Toc529997970)

[4.5 Simulating failure recovery 8](#_Toc529997971)

[4.6 Possible issues 10](#_Toc529997972)

[5. Evaluation 10](#_Toc529997973)

[5.1 Load balancing 10](#_Toc529997974)

[5.2 Failure Recovery 11](#_Toc529997975)

[6. Conclusion 11](#_Toc529997976)

[7. Reference 12](#_Toc529997977)

## Introduction

The objective of this project is to simulate a data centre network in *Mininet* and explore the use cases of SDN in these networks

### 1.1 SDN in Data Centre Networks

Software Defined Networking (SDN), “separates the network control plane from the data forwarding plane, and provides user applications with a centralized view of the distributed network states”, thereby assigning the control of the hardware to a software based controller and achieving virtualization. The controller, “globally regulates the network states via network policies in either a centralized or distributed manner.” This allows the Data Center to manage policy and security much more efficiently and effectively. Data forwarding can be done with the employment of programmable Openflow Switches, through the Openflow Controller. Data Centers would be able manage traffic to many IP Addresses as well as virtual machines, since SDN has dynamic routing, which optimizes the routes for traffic flow, to minimize delay and packet loss at the links. The SDN Controller “efficiently updates the network with consistency in real-time and safety without packet drops, and with low synchronization overhead.” (Akyildiz, 2014)

### 1.2 Literature review

**“*A roadmap for traffic engineering in SDN-OpenFlow networks*”** is a literature review paper that summarizes the current uses of SDN-OpenFlow for traffic engineering. In essence, there are 4 advantages of using SDN in traffic engineering: 1) centralised visibility, 2) programmability without having to handle individual switches, 3) agnostic to vendors 4) flexible configuration using multiple flow table pipelines.

**“*Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing*”** is a research paper that proves “fat-trees”, a tree topology with thicker branches near the top of the tree, are efficient for communication in parallel supercomputers. This topology inspired the field of telecommunications which resulting in the *Clos network*; a multistage circuit-switching network used in this project.

**“*PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric*”** is a research paper that proposes usage of pseudo-MAC addresses to solve scalability issues (performance, cost, routing, energy, cabling) due to a limited Ethernet bandwidth. This is most notable between Core switches and End of Row switches as it becomes a bottleneck in branches near the top of the tree. This approach is only applicable to hierarchical topologies (rather than flat topology).

PortLand can be applied to SDN. PortLand’s fabric manager can act as an OpenFlow controller that will perform two tasks

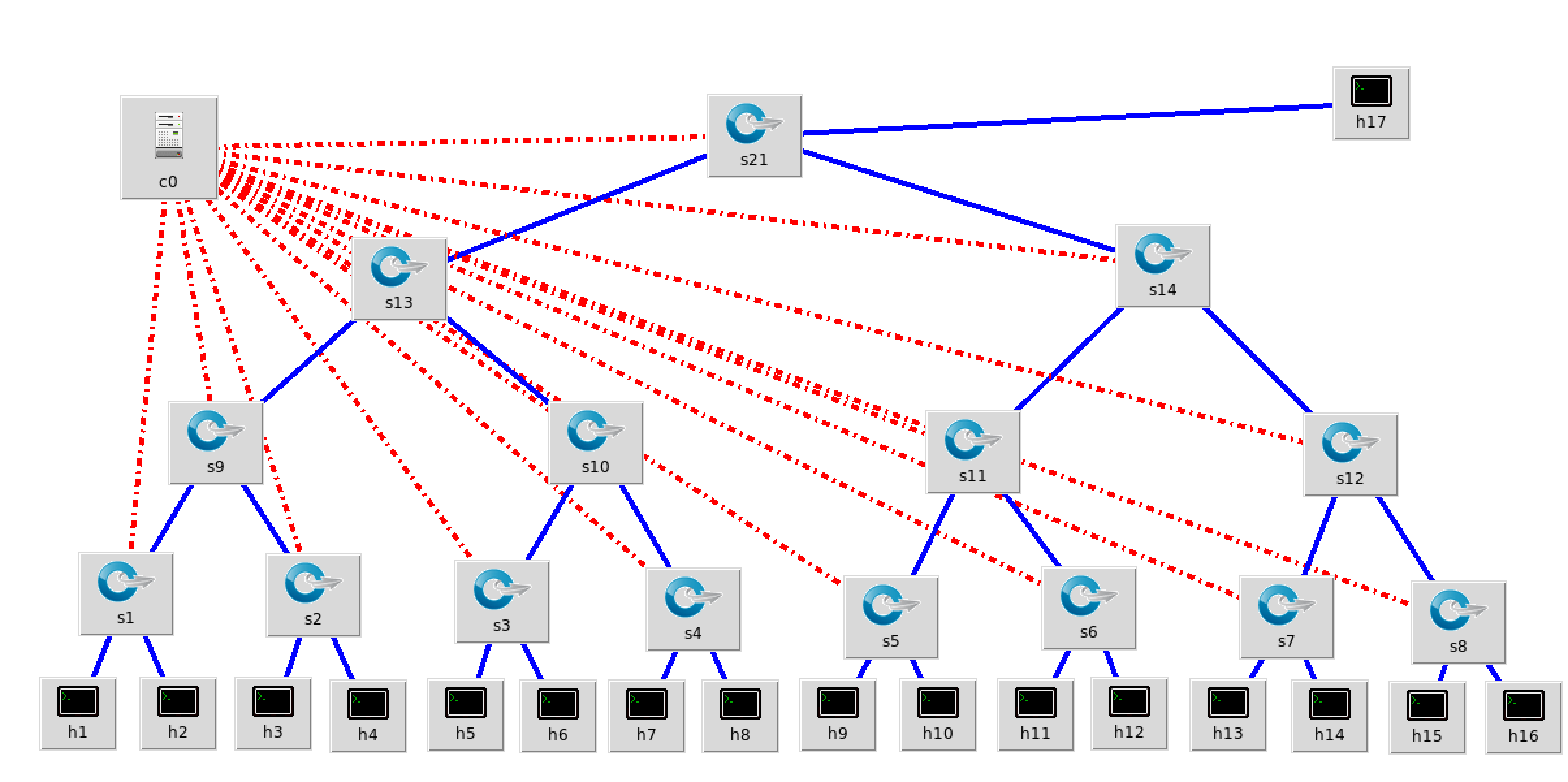
1. MAC-learning: Top of Rack switches re-assign server’s MAC addresses (usually random and flat) into a hierarchical pseudo-MAC addresses.
2. Proxy ARP: Due to these new pseudo-MAC addresses, the other switches must learn these addresses. An OpenFlow controller then act as an ARP that translates destination addresses to these pseudo-MAC addresses.

This project has considered PortLand for future improvements.

## Design

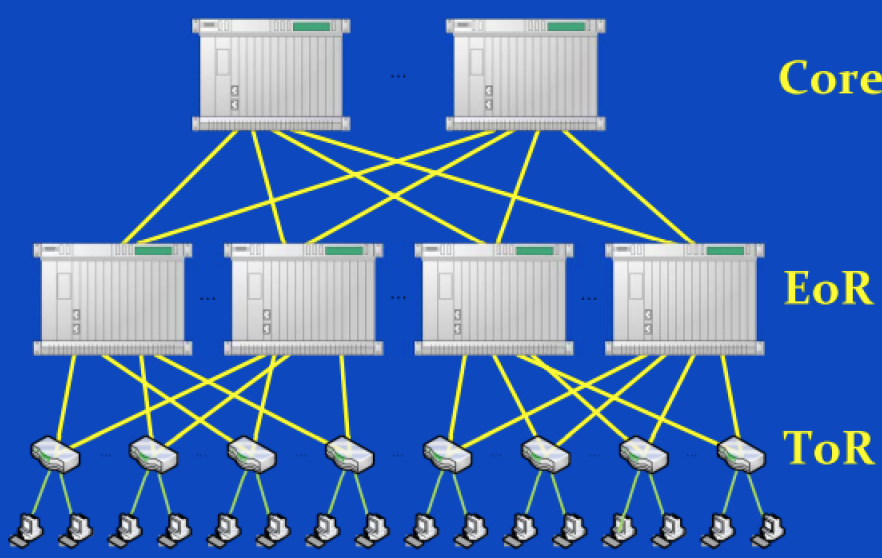
### 2.1 Topology: Fan Network

A traditional fan data centre topology is shown below:



The problem for this is that if a link/switch fails, the leaf switches/hosts will be effected.

### 2.2 Topology: Clos Network



Clos Network is a hierarchical tree topology with loops. Loops enable the assurance that if a link/switch fails, other alternative links/switches exists to reroute the traffic. The hierarchy is as follows:

1. Core switches:
2. End of Row (EoR) switches: Aggregates various “pods” (per company no. of servers).
3. Top of Rack (ToR) switches: Connects all server blades. Provide system control (temperature, OS updates…).

## Implementation

This section describes how we decide which controller to use, how we create the topology and parts of the code that we customize for the project.

### 3.1 Controller: POX

List of available SDN controllers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | NOX | POX | Ryu | Floodlight |
| Language | C++ | Python | Python | Java |
| Performance | Fast | Slow | Slow | Fast |
| Learning curve | Moderate | Easy | Moderate | Steep |

We have chosen to use the POX controller as it has the most comprehensive documentation and ease of use (Python). A slower performance is acceptable as the project is for academic (non-commercial) purposes.

POX comes with several additional features called *components* that can be modified to suit the needs of the project. The important ones are described below. (The rest are for debugging purposes)

### 3.2 POX components

* forwarding.l2\_learning

This enables OpenFlow switches to act as a type of Ethernet learning switch (l2 stands for layer 2)

* openflow.discovery

This enables the switches to discover the network topology by sending LLDP messages (Link Layer Discovery Protocol – a protocol that allows layer 2 switches to store information about other switches).

* openflow.spanning\_tree --no-flood --hold-down

This enables the openflow.discovery component to work in topologies with loops. The spanning tree protocol also works when a link is broken but an alternate link exists. It does this by re-creating the spanning tree. The options, --no-flood and –hold-down makes the spanning tree to take the “safer” approach and not flood the topology before creating the view of the tree.

* misc.ip\_loadbalancer --ip=<XX:XX:XX:XX> --servers=  
  This enables a switch to act as a type of border router (has NAT-like functionality whereby other clients use this address to contact the data centre servers). The load balancer will first find other hosts (through the given servers addresses) and randomly selects from these pool of servers.

3.3 Virtual network & Topology editor: Mininet and MiniEdit  
  
Mininet is a virtual network. MiniEdit is the pre-installed topology editor.

## Steps-to-reproduce

### 4.1 Pre-requisites

Required software:

1. *Mininet*
2. wget or curl

Code:

Attached with project submission. Alternatively, download the code from GitHub.

$ git clone https://github.com/PandaRider/50.012\_Networks\_Project.git

If you are using the LEET lab computers, please copy the latest repositories from GitHub:

$ git clone https://github.com/mininet/mininet.git

$ git clone https://github.com/noxrepo/pox.git

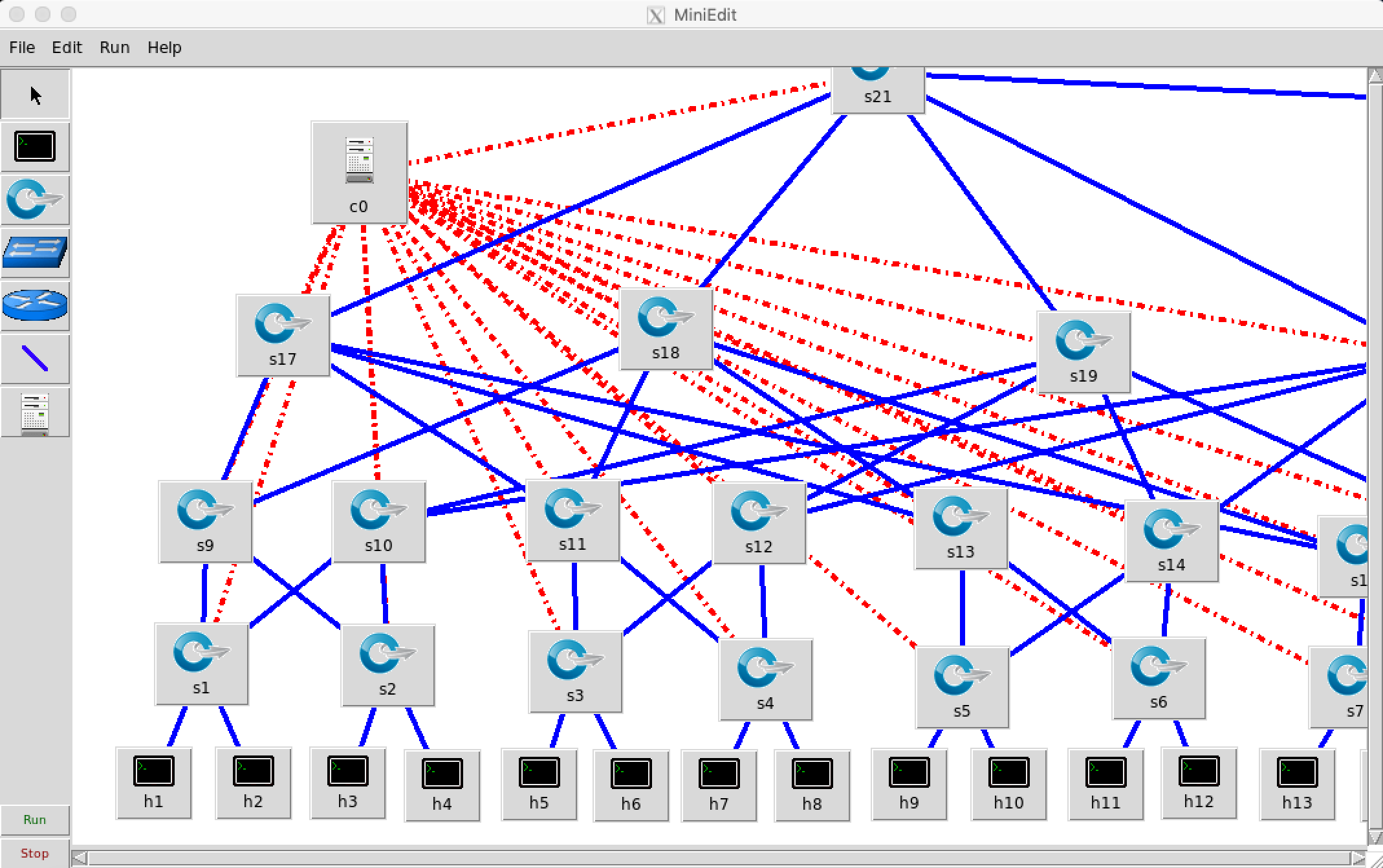
Copy selective\_switch.py to /pox/ext .

### 4.2 Running the Data Centre topology in MiniEdit

Start MiniEdit

$ sudo ~/mininet/examples/miniedit.py

In *MiniEdit*, File > Open > 50.012\_Networks\_Project/ v4\_topo.mn.



Click Run at the bottom left.

In *Mininet* console

mininet> xterm h1 h2

In h1 console

h1> python –m SimpleHTTPServer 80

In h2 console,

h2> python –m SimpleHTTPServer 80

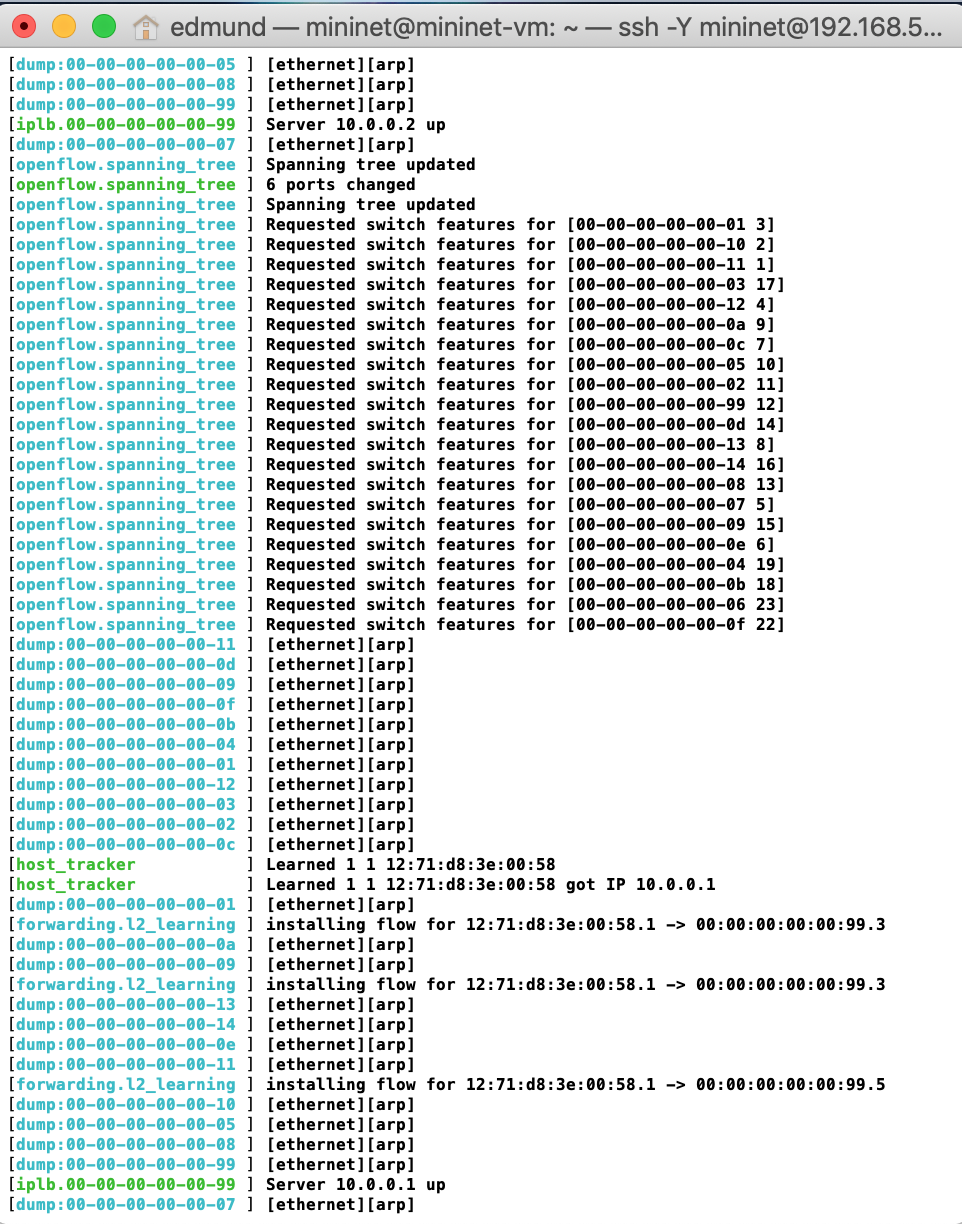
### 4.3 Running the POX controller

Open a separate terminal

$ sudo ~/pox/pox.py selective\_switch --ignore\_dpid=99 misc.ip\_loadbalancer --ip=10.8.8.8 --servers=10.0.0.1,10.0.0.2 --dpid=99 openflow.spanning\_tree --no-flood --hold-down log.level --DEBUG samples.pretty\_log openflow.discovery host\_tracker info.packet\_dump

(Note: Do note that the --servers option allows multiple servers. To provision more servers, add the respective ip addresses. For example, --servers=10.0.0.4,10.0.0.5,10.0.0.6 will provision 3 servers from h4, h5, h6.

The spanning tree protocol will update for few seconds. Wait for Server 10.0.0.2 up and Server 10.0.0.1 up as shown below:



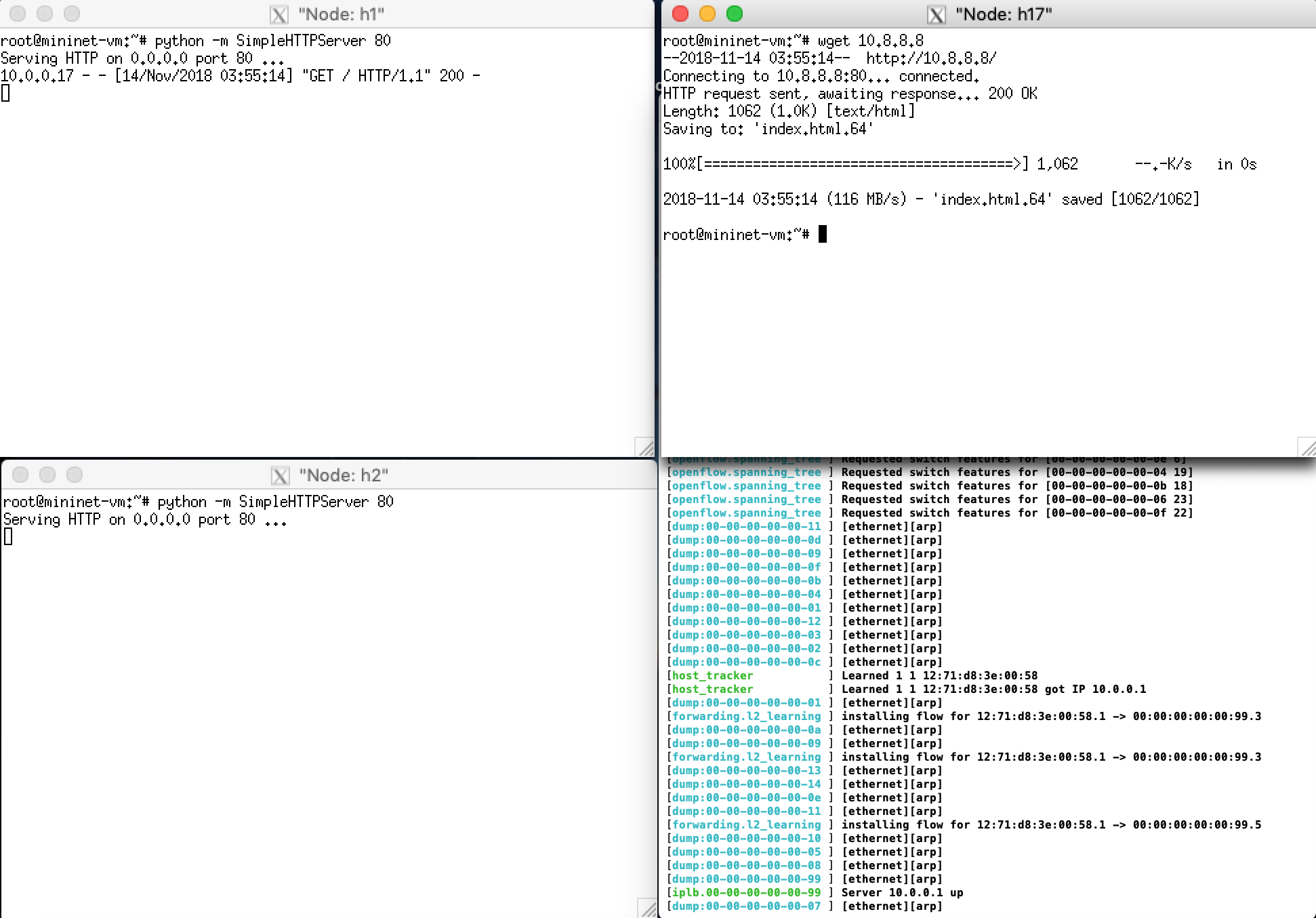
### 4.4 Simulating load balancing

In *Mininet* console

mininet> xterm h1 h2

In h17 console,

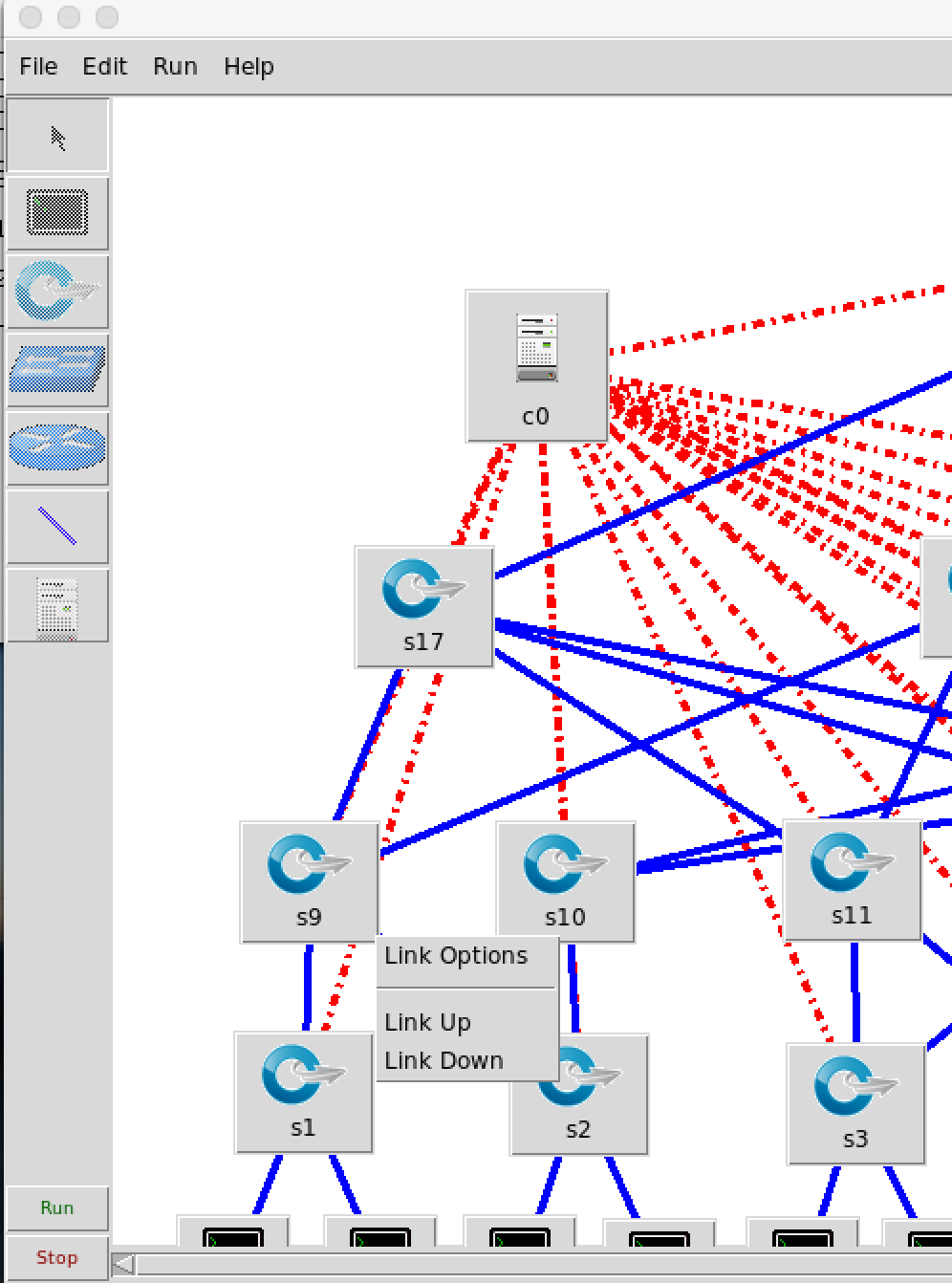
h17> wget 10.8.8.8



Observe that ***either*** h1 or h2 receives the HTTP GET command.

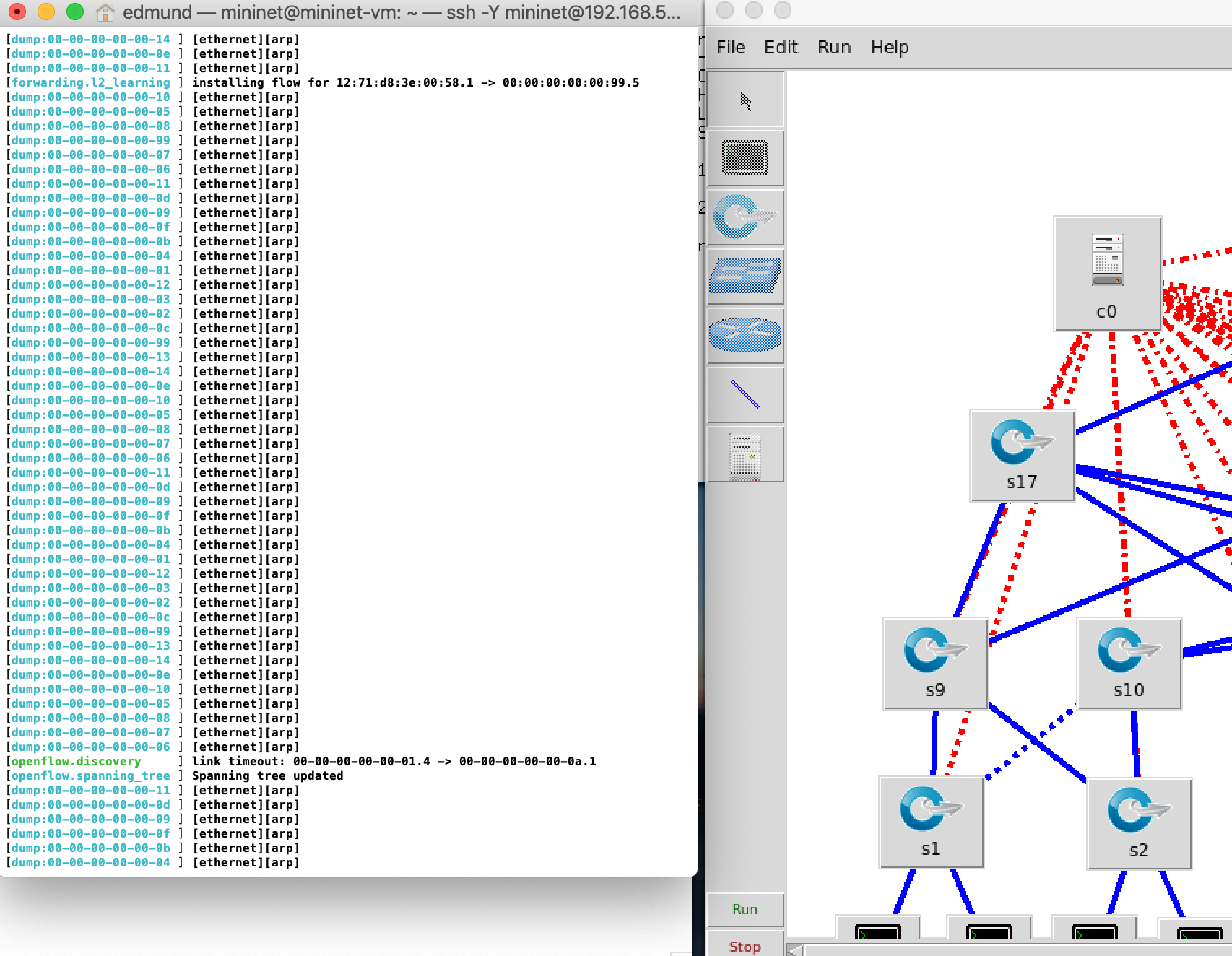
### 4.5 Simulating failure recovery

In MiniEdit, right click on the link between s1 and s10. Click on Link Down.



Observe that openflow.discovery will timeout and openflow.spanning\_tree will update by itself. Test that the client h17 is still able to retrieve from the data centre by entering:

h17> wget 10.8.8.8



### 4.6 Possible issues

* Ensure POX is updated to latest repo.

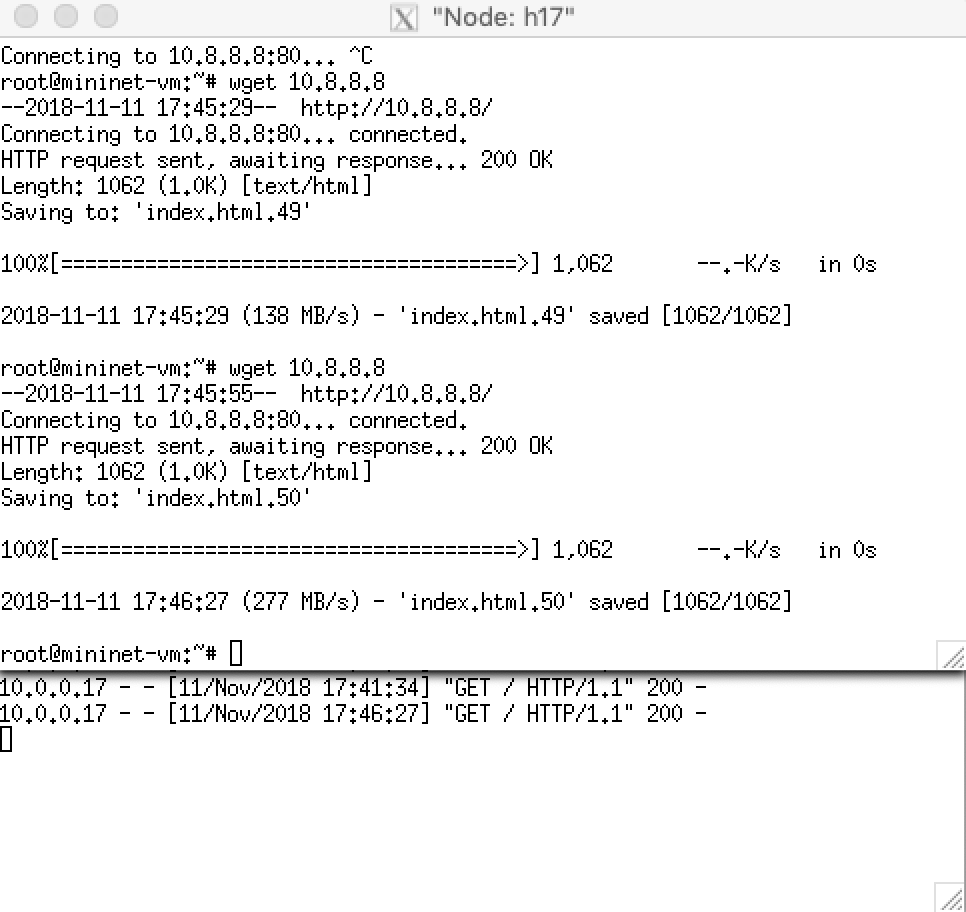
## Evaluation

### 5.1 Load balancing

*iperf* tests with 4 clients demonstrate a higher bandwidth as more servers are provisioned.

### 5.2 Failure Recovery

While the spanning tree takes about >2 seconds to load, there is an edge scenario where a HTTP GET packet is sent ***while*** the spanning tree loads. This might cause the packet to be lost as the spanning packet might be dropped at the switch with the broken link. As can be seen below, the packet eventually (after ~58 seconds) completes. This suggests that the switch does store pending packets in the buffer and wait for the latest OpenFlow flow tables to be applied



## Conclusion

With the increase use of virtualization for Data Centers, it has since automated as well as streamlined server provisioning. However, virtualization only concerns itself with the computer/server workload and it does not virtualize networks or storage. SDN routing also outperforms in terms of throughput, and this is one of the major advantage of using SDN in Data Center as this allows it to deal with big data.

In this project, we have demonstrated how load balancing and failure recovery can be simulated

## Reference

Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks,71*, 1-30. doi:10.1016/j.comnet.2014.06.002

Leiserson, C. E. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers,C-34*(10), 892-901. doi:10.1109/tc.1985.6312192

Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., . . . Vahdat, A. (2009). PortLand. *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication - SIGCOMM 09*. doi:10.1145/1592568.1592575