CS 6301.502
Software Defined Networking
Fall 2018

Project Report
Software Defined Load Balancer with POX Controller

Submitted by:

Priyank Shah
Manan Lakhani
Karan Motani
Ankita Patil
Kripanshu Bhargava

Instructor:
Timothy Culver

Table of Contents

Abstract

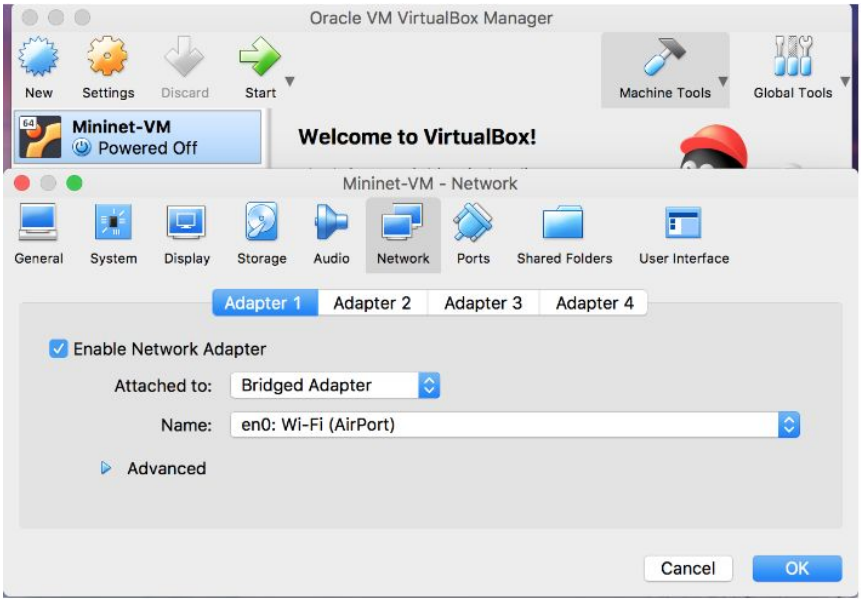
The project aims to develop a software-defined load balancer with POX controller which makes flow balancing decisions based on the statistics such as bandwidth and latency, gathered from interfaces of switches and nodes in the network. POX is an open source development platform for Python-based software-defined networking (SDN) control applications which helps in checking the configuration of a customized networking application. A network is formed with 3 switches and 3 hosts in mininet and flow modification is performed supported the statistics gathered from the network. The project is enforced with the POX controller that has sensible support and documentation. Mininet is employed with the Oracle VM for making the network.

Initial Setup

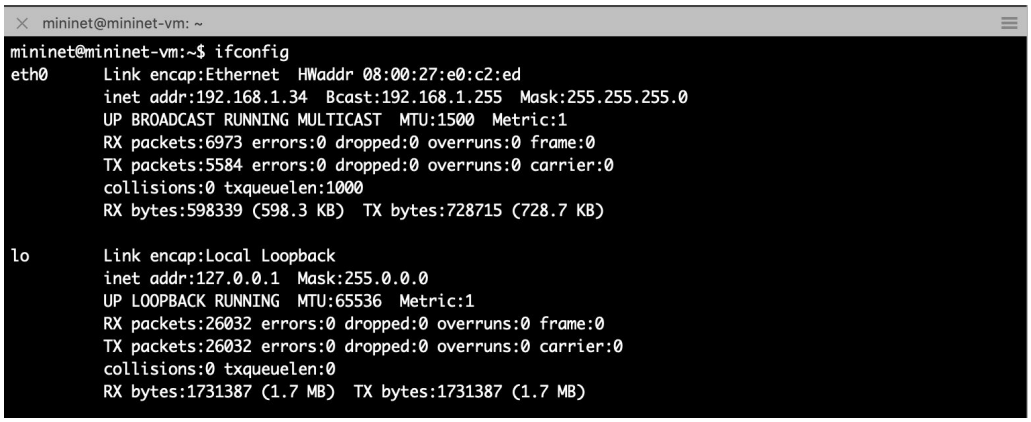
1) Install Oracle VM on the machine

2) Import the Mininet 2.2.1 OVF into the VM.

3) Then, in the network settings set up Adapter1 as Bridged Adapter.



4) Now, start the Mininet and do ifconfig which gives us the host address.



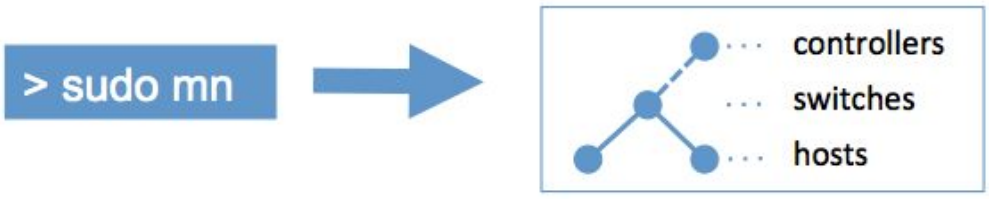
5) We can use this host address to connect to Putty and WinSCP.

Technology

1. **Mininet**

   Mininet is a software emulator for prototyping a large network on a single machine.

   Mininet can be used to quickly create a realistic virtual network running actual kernel, switch and software application code on a personal computer. Mininet allows the user to quickly create, interact with, customize and share a software-defined network (SDN) prototype to simulate a network topology that uses Openflow switches.



2. **POX Controller**

   POX is an open source development platform for Python-based software defined networking (SDN) control applications which helps in checking the configuration of a customized networking application.

   POX is in-built in mininet and has good support and documentation.

3. **OpenDayLight**

OpneDayLight (ODL) is an open source software defined networking (SDN) project aims at enhancing SDN project which is used to visually represent and simulate topologies.

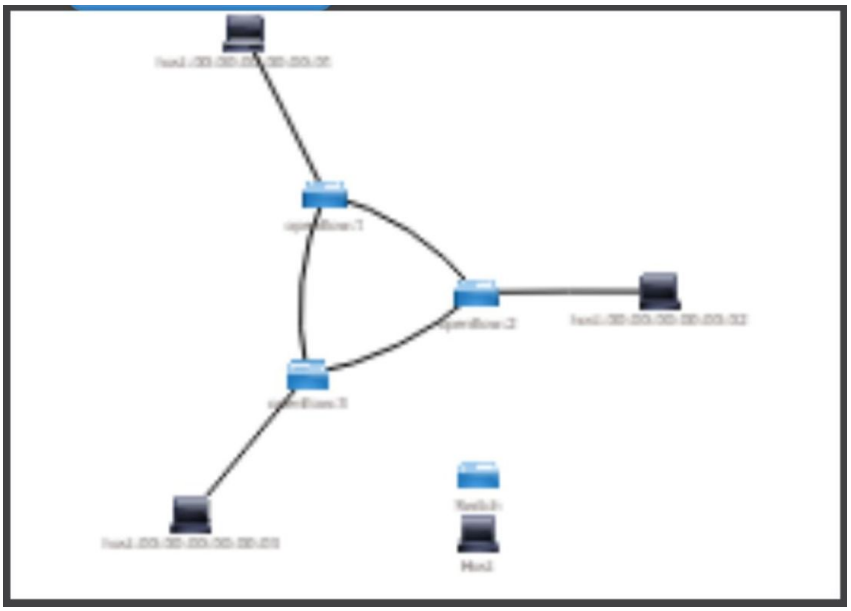
Network Architecture - Topology

In mininet, a network topology is created which consists of three switches and three hosts. The hosts are labeled as h1, h2, h3, and switches are labeled as S1, S2, S3.

Initially, a spanning tree is read by the controller from a local file. This file is updated every five minutes based on the congestion in the traffic

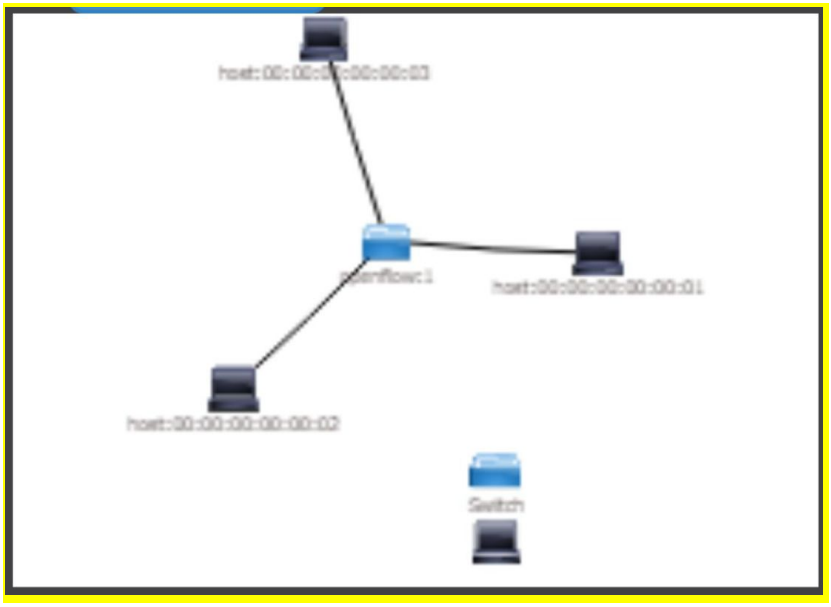
RING TOPOLOGY

sudo mn --custom topo_ring.py --topo mytopo --controller remote --mac --switch ovsk

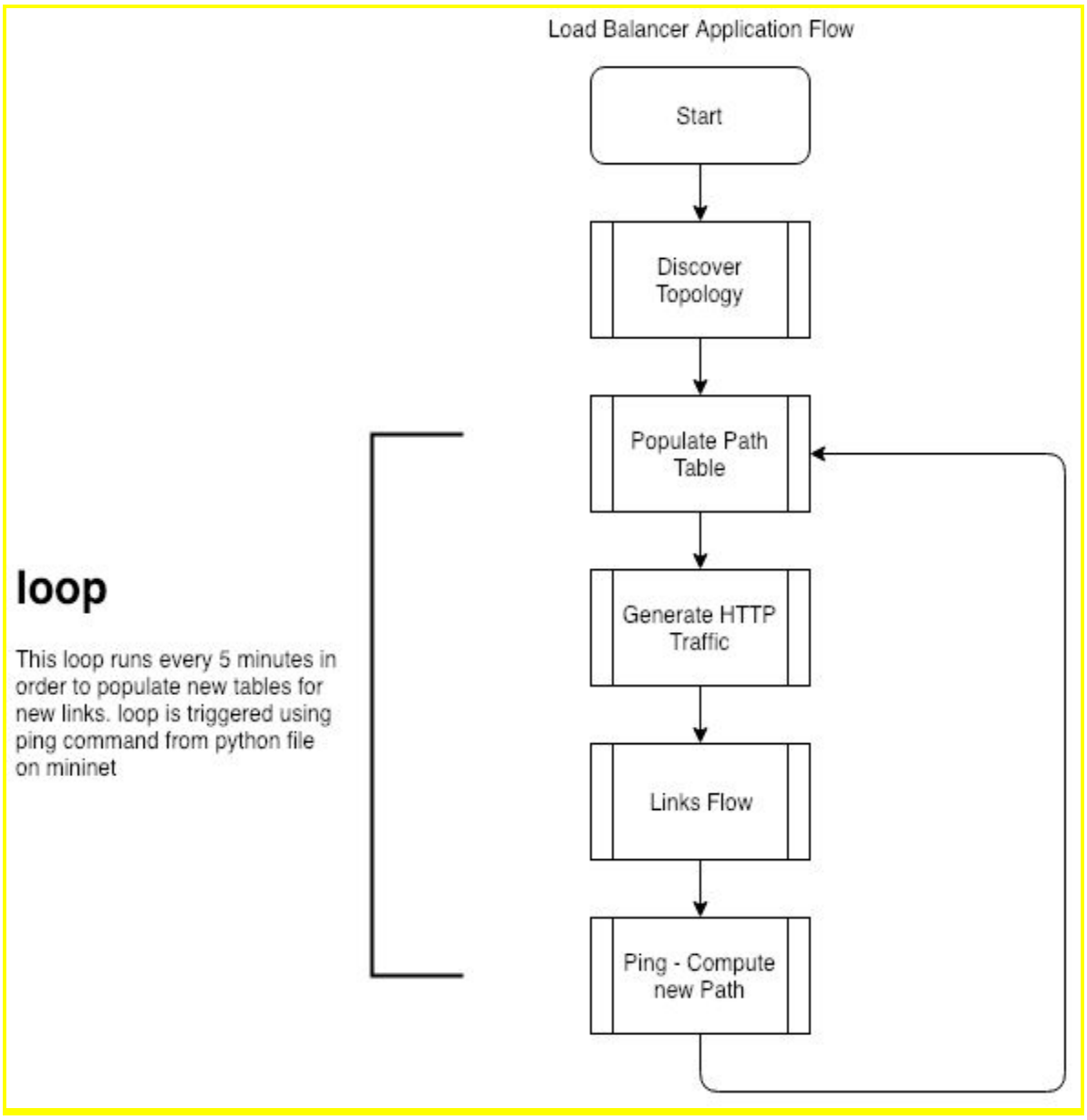


STAR TOPOLOGY

sudo mn --custom topo_star.py --topo mytopo --controller remote --mac --switch ovsk



Application Flow



The above diagram explains the flow of the whole project. After setting up the whole system the controller is started. The topology of the network is discovered by running the python script which includes the flow_table_population() method. This function takes the paths as part of input from the paths.txt to get the spanning tree for the network. The data we get is then populated in the data structure called path_table inside the controller.

We use two listeners here:
1) **Link Event** - which triggers **links_handle()** method when a link is pushed between switches.

2) **Packet Event**- which triggers **packets_handle()** whenever a packet is received by the switch.

links_handle() method updates port map data structure with values of ports to be forwarded for every link between switches.

whereas, **packets_handle()** after receiving the packet, based on the entries in the path_table data structures, flow is updated by using **update_flow_mod()** method in pox library.

Now our network is ready. Http scripts are used to generate traffic. We have made a loop of 5 minutes in order to generate a significant amount of traffic. During the loop, the links between all the switches are activated using a method called **add_test_flow()**, and the table is populated for this loop.

After that, new paths are taken by running ping statistics and the paths.file is updated accordingly. This loop continues till the significant amount of data is generated.

We visualize these statistics on console to see how the whole process behave given a certain input.

Compiling Instructions

- Controller.py should be placed in pox folder of the pox installation.
- Paths.txt should also be placed in pox folder.
- All the ping scripts and HTTP traffic generation scripts and stats.py should be copied to the mininet home directory.
- Now run the controller using the following command inside pox installation directory:

```
./pox.py log.level --DEBUG controller OpenFlow.discovery openflow.of_01
```

Running this command also includes OpenFlow discovery with the controller.

- Now run mininet using the following command:

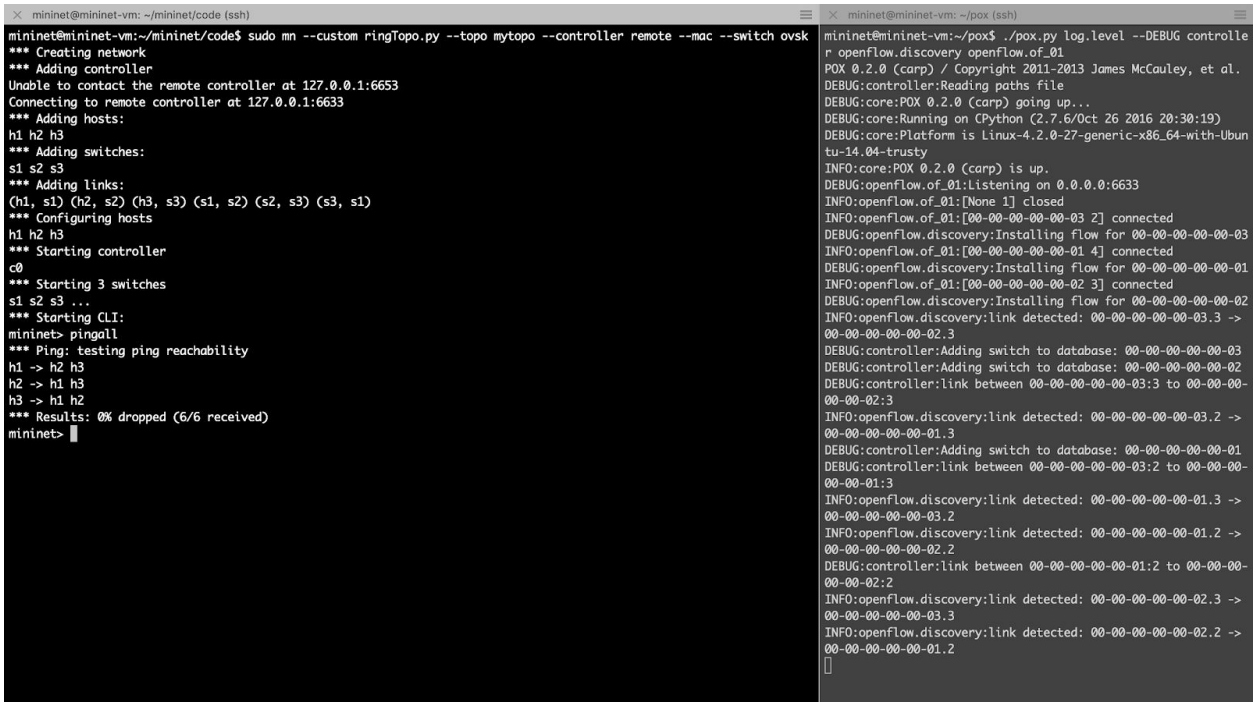
```
sudo mn --custom ringTopo.py --topo mytopo --controller remote --mac --switch ovsk
sudo mn --custom starTopo.py --topo mytopo --controller remote --mac --switch ovsk
```

- Now run the HTTP traffic generation scripts and after 5 minutes, run ping scripts in the delay of 150 seconds to update the topology if required.
- Now the traffic should be rerouted via a different route.

Screenshots

Starting POX controller and the topology on mininet

Create Custom Topology

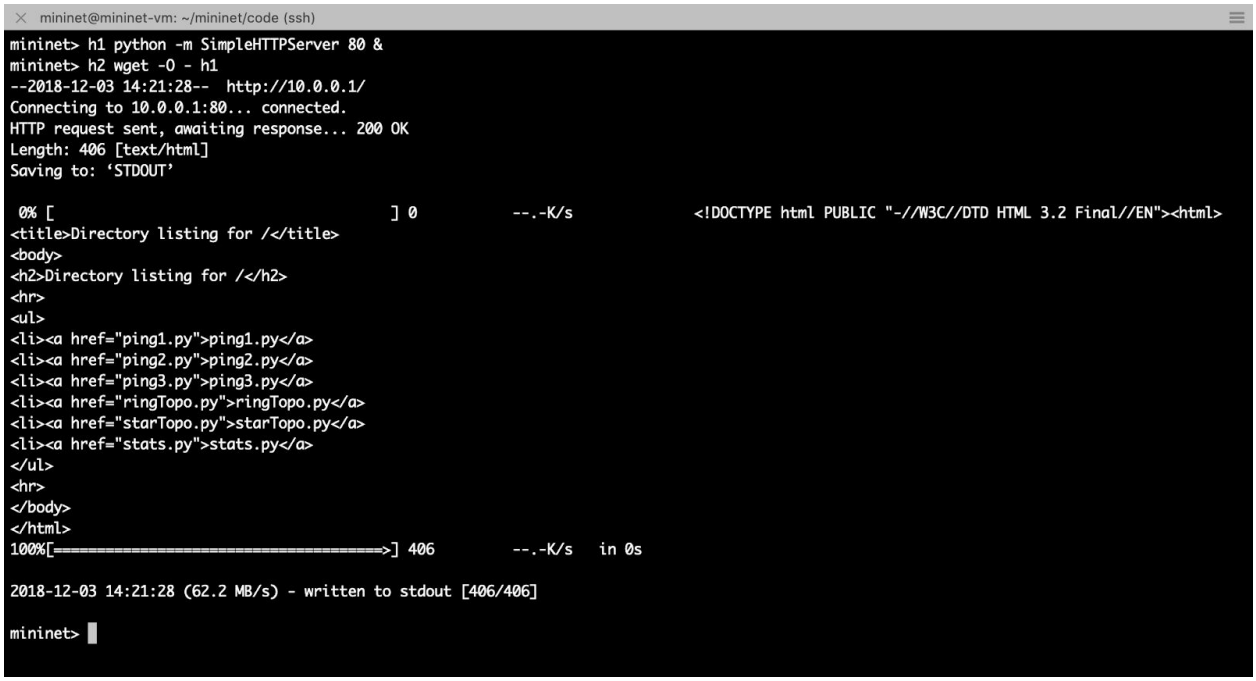


Traffic is generated by starting the HTTP web server:

```
mininet> h1 python -m SimpleHTTPServer 80
```

and wget to get files from hosts h2 and h3 to generate some traffic in the network:

```
mininet> h2 wget -O - h1
```



This traffic is generated for some time and after some time, stats are collected by running following script:

```
mininet> h1 python ping1.py
mininet> h2 python ping2.py
mininet> h3 python ping3.py
```

From the output of above command from different hosts, the original file is updated to below:

mininet>sh python stats.py

Based on the stats, the controller modifies the flow. Here in this example, the path from 1 to 3 has more congestion. So, the packet is routed via 1->2->3.

Statistics

To get the statistics from the switches, a set of python scripts are written which will ping all the nodes from a particular node. After pinging the nodes, average latency between each node is collected. Depending upon the average latency, an alternative path with lowest overall latency is selected.

These statistics are used by the controller to modify flows.

import urllib, os

This script is used for pinging the client 2 from 1

os.system("ping -c 5 10.0.0.2 | tail -1| awk '{print $4}' | cut -d '/' -f 2 >> ping_stat_1_2")

This traffic is generated for some time and after some time, stats are collected by running the following script: mininet> h1 python ping1.py mininet> h2 python ping2.py mininet> h3 python ping3.py mininet>sh python stat.py

From the output of the above command from different hosts, the original file is updated to below: mininet>sh python stat.py

1:2:2

2:1:1

1:3:2

3:1:2

3:2:2

2:3:3

Based on the stats, the controller modifies the flow. Here in this example, the path from 1 to 3 has more congestion. So, the packet is routed via 1->2->3.

Challenges

Challenges faced while setting up the whole project from a closed network

Few of our group members tried to work on the project from their respective places/home network. The WiFi provider their take machine address for verification rather than verification by IP address. This led to the VM machine connection problem using bridge adapter.

Solution

VPN is a smart solution to this problem as we can use university network from our place using VPN.

Challenges faced while setting up OpenDayLight (ODL)

Initially, we tried setting up OpenDayLight with Java 8 environment and Carbon SR1. With this configuration, we were not able to set up localhost:8181/dlux/index.html

The page was accessible with localhost:8181/index.html, but the topologies were not visible

Solution

Instead of Java 8 and Carbon SR1, we tried setting up OpenDayLight with Java 7 and Helium SR2 and localhost:8181/dlux/index.html gave expected results.

Challenges faced while setting up POX Controller

While setting up POX controller, we encountered following errors

```
ERROR:openflow.of_01:Error 98 while binding 0.0.0.0:6633: Address already in use
ERROR:openflow.of_01: You may have another controller running.
ERROR:openflow.of_01: Use openflow.of_01 --port=<port> to run POX on another port
```

Solution

The above errors are fixed using the following commands

```
netstat -lnptu
kill -9 <PID>
```

Reference Link: https://github.com/noxrepo/pox/issues/128

When ARP request is broadcasted, it absolutely was generating Brobdingnagian traffic. This caused congestion and taking longer for the ping to figure out. To resolve this issue, whenever Arp request is received, MAC address is hardcoded and sent to the requested host. this is often handled in handle_packetin methodology.

Conclusion

Thus, we implemented software defined POX controller which makes flow balancing decisions based on statistics such as bandwidth and latency gathered from the interfaces of nodes and switches in the network which is simulated in mininet.

References

1. http://mininet.org/walkthrough/
2. https://www.opendaylight.org/
3. https://openflow.stanford.edu/display/ONL/POX+Wiki.html
4. http://networkstatic.net/installing-mininet-opendaylight-open-vswitch/
5. https://searchnetworking.techtarget.com/definition/Mininet