



MINT709

Capstone Project Report

Design & Implementation of a Dynamic and Traffic Aware Load Balancer on SDN
controller (OpenDaylight)

September 2016

Japmeet Singh

Design & Implementation of a Dynamic and Traffic Aware Load Balancer on SDN
controller (OpenDaylight)

By

Japmeet Singh

A project submitted in partial fulfillment
of the requirements for the degree of
Masters of Science in Internetworking
at the
University of Alberta
December 2016

Abstract

Design & Implementation of a Dynamic and Traffic Aware Load Balancer on SDN controller (OpenDaylight)

By Japmeet Singh

University of Alberta, 2016

Acknowledgements

I would like to thank my project mentor and supervisor Prof. Gurpreet Nanda for his continuous guidance throughout the duration of my MSc, as well as during the project implementation. I would also like to thank my family for their continuous support and guidance.

Table of Contents

Introduction.....	6
Software Defined Networking.....	6
OpenFlow.....	7
The Need for Software Defined Load Balancing.....	9
Project Purpose	10
Lab Implementation	11
Issues & Challenges.....	26
Conclusion.....	27
Appendices & References.....	28
Mininet Network Topology.....	28
References.....	29
Table of Figures.....	30

Introduction

Software Defined Networking (SDN)

Software defined networking (SDN) is an emerging networking architecture that aims to be dynamic, centrally manageable, cost-effective, future proof and adaptable. SDN is ideal for today's network applications that require high bandwidth and are dynamic in nature. SDN aims to eventually replace traditional networking by separating the network control logic from the networking devices. This process is also referred to as decoupling the control plane from the forwarding plane. This allows the control logic to be highly centralized as well as highly programmable and it reduces the effort required to maintain the network. The SDN approach to a network also makes it easy to diagnose the network in the event of errors or failures.

A software defined network consists of devices that are referred to as 'Controllers'. Switches, that are the forwarding devices, send the packets they receive to the controller and the controller tells the switches what to do with those packets by installing 'rules' into the switches. Essentially, whenever a switch receives a packet that does not match any existing rule in the switch, it is sent to the controller for processing. Once the rules have been installed on the switches, the need to forward (those) packets to the controller is eliminated and the latency to perform the requested action is decreased. All communication that takes place in a software defined network takes place via OpenFlow messages.

Advantages of SDN

1. **Programmable:** As the control and the physical layers of the network elements are decoupled, the network control is programmable on the SDN controller, which is at the logical center of the network.
2. **Central:** Since the SDN controller is logically central to the network, it maintains a global network topology view at all times. This allows the controller to manage the entire network, without relying upon any other network element.

3. Vendor Neutral & Open Source:

Software Defined Networks when implemented via Open Source standards, are not restricted to vendor specific hardware, software and protocols. This in turn allows ‘off-the-shelf’ hardware to be used in order to deploy Software Defined Networks. Since the protocol/rules are defined by the Open Networking Foundation (ONF) and are implemented via the Network OS (ex. Mininet), vendor lock-ins can be easily avoided. Furthermore, Controllers such as OpenDaylight are open source projects, everybody can contribute to the development of the projects and the community is very large and diverse.

While SDN is a very promising direction for modern networks, it is not without its disadvantages. The primary challenge that Software Defined Networks face today is the challenge with the adoption of the new standard. Since SDN is a fairly new field of study and research, there are a lot of features missing that are very critical to the functioning of many large scale networks. Furthermore, many large scale networks do not, yet, see the feasibility to bring their services down in order to switch from traditional networking to software defined networking. The work around for that has been the deployment of ‘hybrid’ networks that can function in both SDN mode and the traditional mode (for redundancy).

OpenFlow

OpenFlow is the protocol that defines how the network elements talk with each other, for example: how the controller will talk with the switches and vice versa. OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network [1]. OpenFlow 1.0 was released on the 31st of December, 2009 and the development is managed by the Open Networking Foundation (ONF). The current OpenFlow version is 1.4, which includes multiple feature additions over version 1.0 like the addition of multiple flow tables, MPLS & VLAN tagging support, virtual ports, controller connection failure modes, the ability to use multiple controllers in the same network, per flow metrics, auxiliary connections to name a few.

An OpenFlow switch separates the control plane and the forwarding plane in a switch. OpenFlow switches only serve to forward data and the control logic is pushed to the OpenFlow switches via

OpenFlow messages. The data plane of an OpenFlow switch consists of a flow table that have flow rules according to which packets are matched and forwarded.

The Software defined networking architecture is shown in figure 1 below:

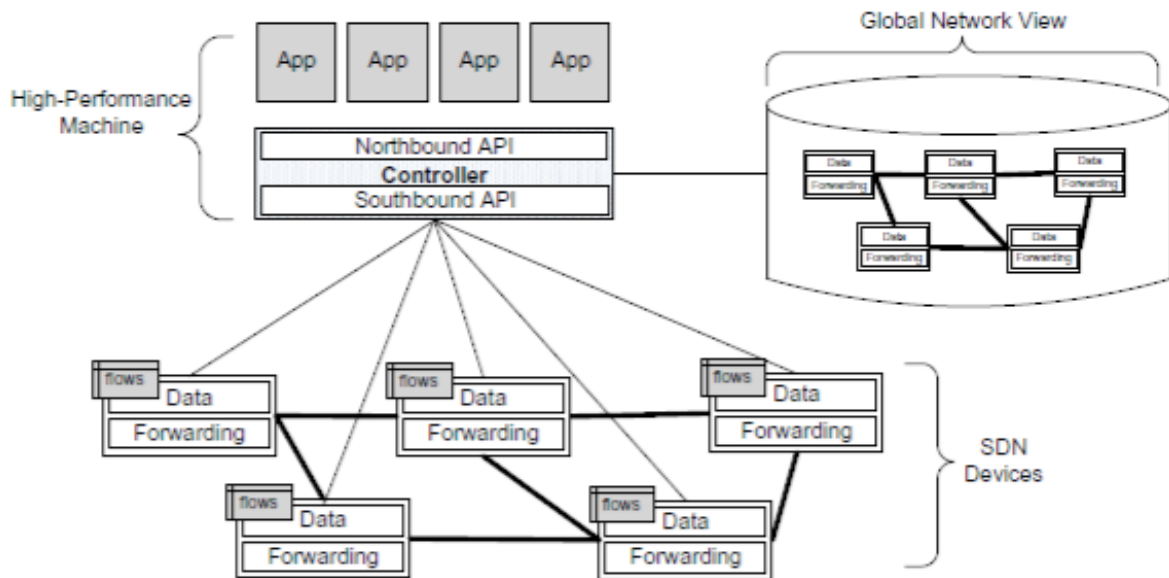


Figure 1

Connected to the Northbound APIs are user 'apps' or 'applications' which can perform a variety of functions. These apps are deployed on top of the controller and rules in the matching tables might be made to forward the data packets to these apps. A 'load balancer' is one of these applications that runs on top of the SDN controller, which manages the load in the network.

There are many different flavors of SDN controllers as well. Mininet, the most popular Network Operation System (NOS) is bundled with a few SDN controllers by default. These controllers are the OVS Controller (OVS: Open VSwitch) and the POX controller. Mininet comes ready for running other SDN controllers like NOX and RYU as well. There are other SDN controllers that can be downloaded separately and be used either within Mininet or run as a separate VM. An example of such a SDN controller is the OpenDaylight controller.

The need for Software Defined Load Balancing

Traditional networks use static switches, resulting in static load balancing techniques. The primary issue with this approach is that each packet has a pre defined path as per a pre defined flow. In the event of a link or a switch failure, the flows and the pre defined paths have to be reconfigured manually. This process becomes very complex and cumbersome as the network grows.

Controllers in a software defined networks have the complete topology information by listening to the switches and calculates paths with the least load. This enables the controllers to make intelligent decisions and frees the switches from any computational load. A challenge that traditional networks face very frequently is the challenge with network looping. In order to solve this problem, spanning tree protocols are run in traditional networks which remove the loops, at the expense of bandwidth.

SDN Load balancers aim to change that by utilizing all the links between switches while avoiding loops. In order to avoid loops, the SDN controller, OpenDaylight in my case, has to proactively install flows into the switches as looping is bound to happen if the switches attempt to discover hosts by the means of L2 Learning (flood packets until the switch builds a table with the hosts mac address and the out port for that host).

The OpenDaylight SDN controller uses a Link Aggregation Control Protocol (LACP) [2] to aggregate multiple links into one big link that connects multiple switches. This aggregated link's bandwidth is the sum of the individual bandwidths of each link. With the use of LACP in OpenDaylight, load balancing is successfully achieved in the scenario where there are multiple links between switches.

Project Purpose

The primary purpose of my project is to develop a load balancing mechanism for use in a network using the OpenDaylight controller. The network topology can be seen in figure 2 below:

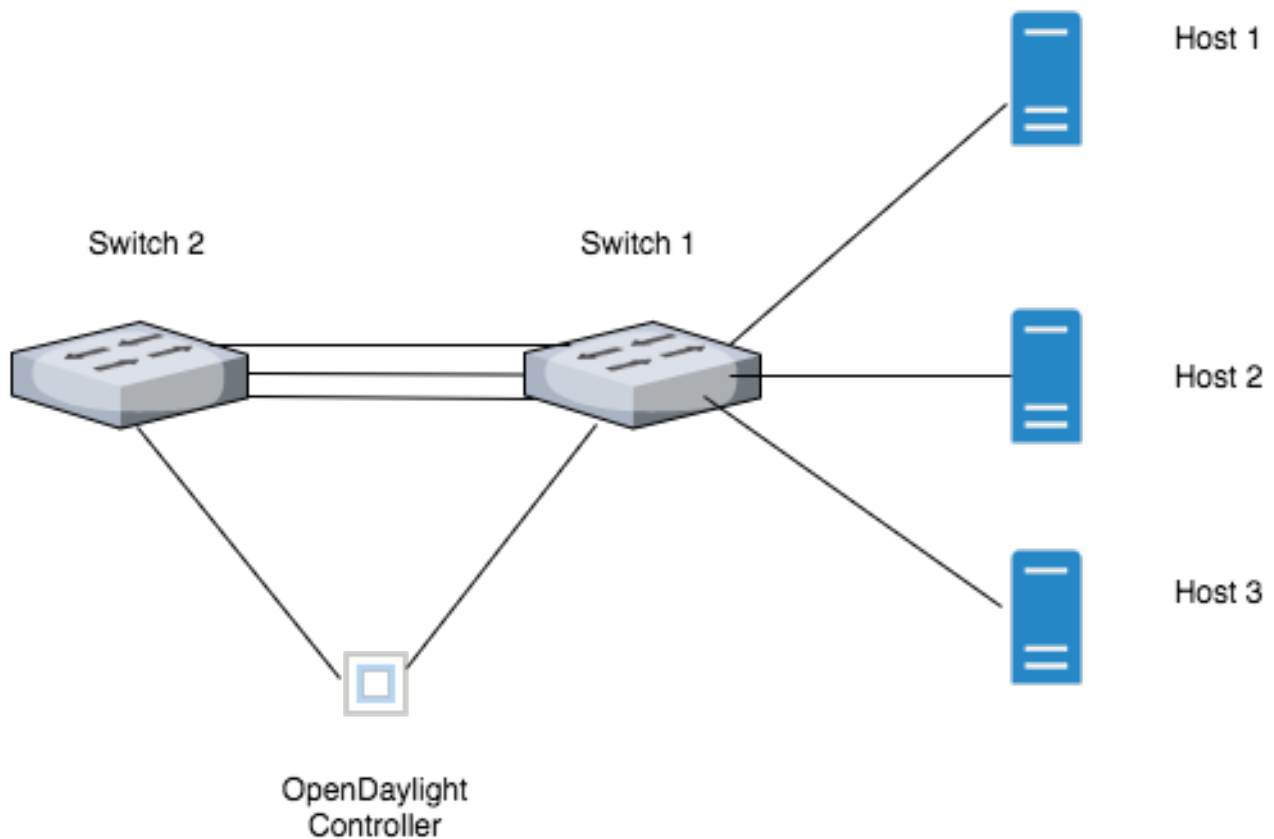


Figure 2

In the network topology in figure 2, if a SDN Load balancer is not used, loops will form between Switch 1 and Switch 2 and the hosts would be unable to talk to each other.

Since this is a SDN network with the OpenDaylight controller pushing rules and flows to the Switches (which are Open VSwitches), loops are avoided. The OpenDaylight controller is running the link aggregator service via the link aggregation control protocol (LACP) which combines the three links between both switches, into one high bandwidth link, thereby providing load aggregation as well as load balancing.

Lab implementation

This section explains all the steps involved in setting up the project and to get it working as required.

1. Download VirtualBox for the required platform from (I am using the Mac OS version):

<https://www.virtualbox.org/wiki/Downloads>

2. Upon successfully downloading and installing VirtualBox, download Mininet VM image from:

<http://onlab.vicci.org/mininet-vm/mininet-2.2.0-150106-ubuntu-14.04-server-amd64.zip>

3. Go to file and select the option to 'import appliance' as shown in figure 3:

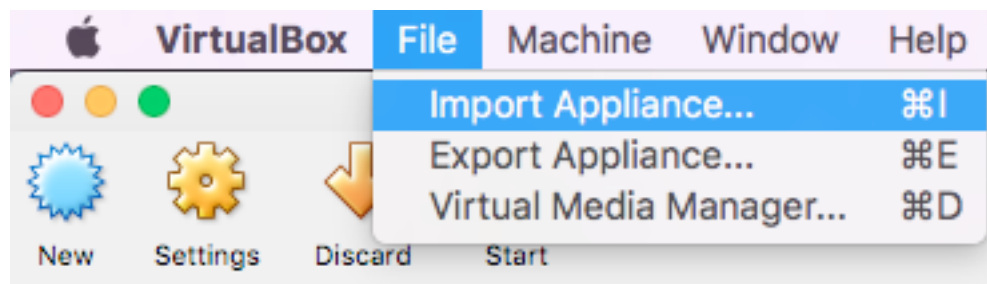


Figure 3

4. Choose the '.ovf' file to import as shown in figure 4:

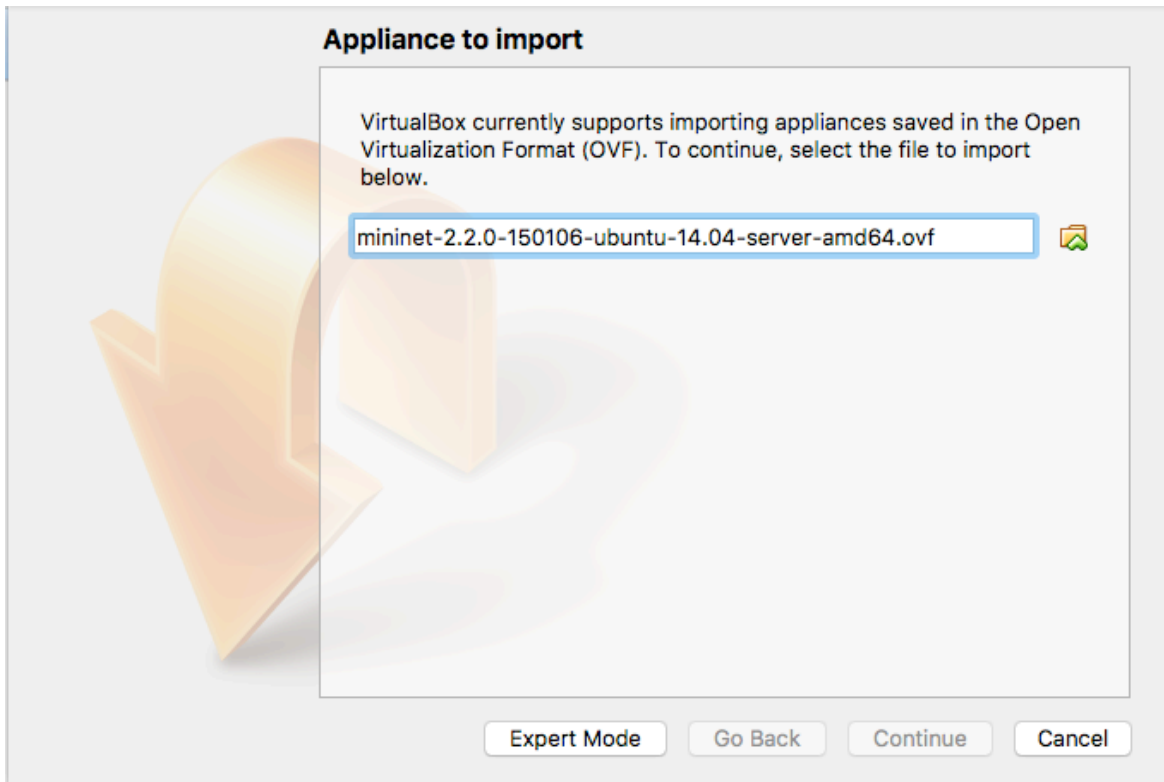


Figure 4

5. Configure the import settings. The VM needs a recommended 2GB of memory to perform. Click on import as shown in figure 5:

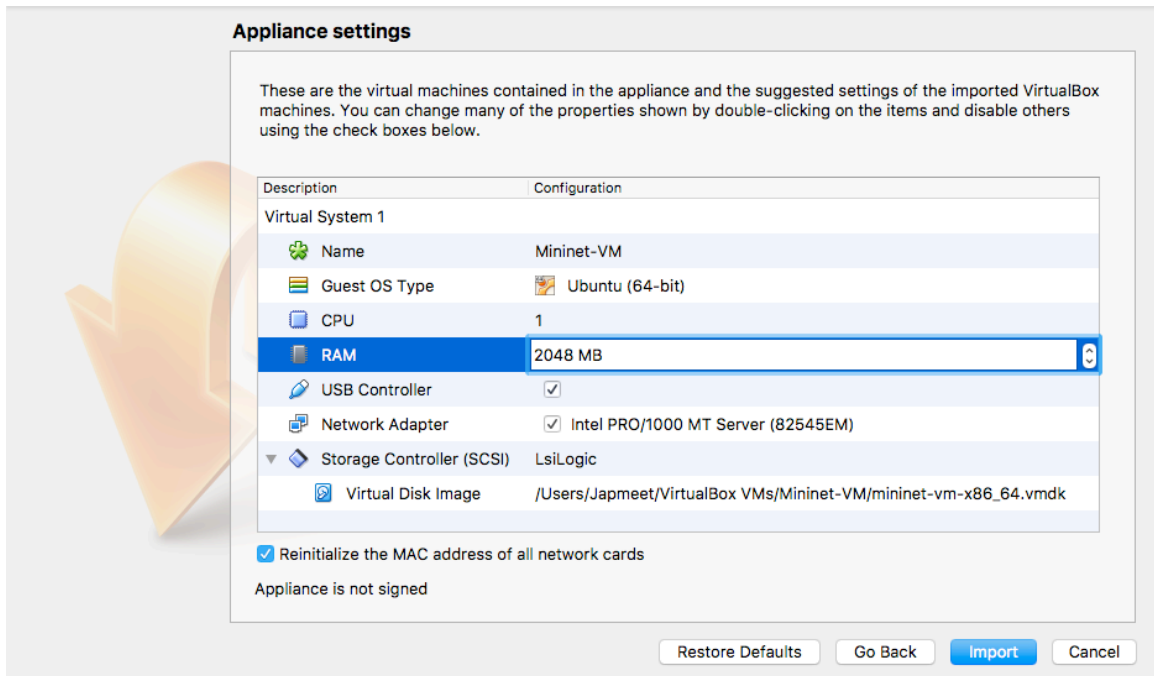


Figure 5

6. Once the import is complete, configure the VM and select the network option. Enable the second network adapter and choose 'Host-Only Adapter' as shown in figure 6:

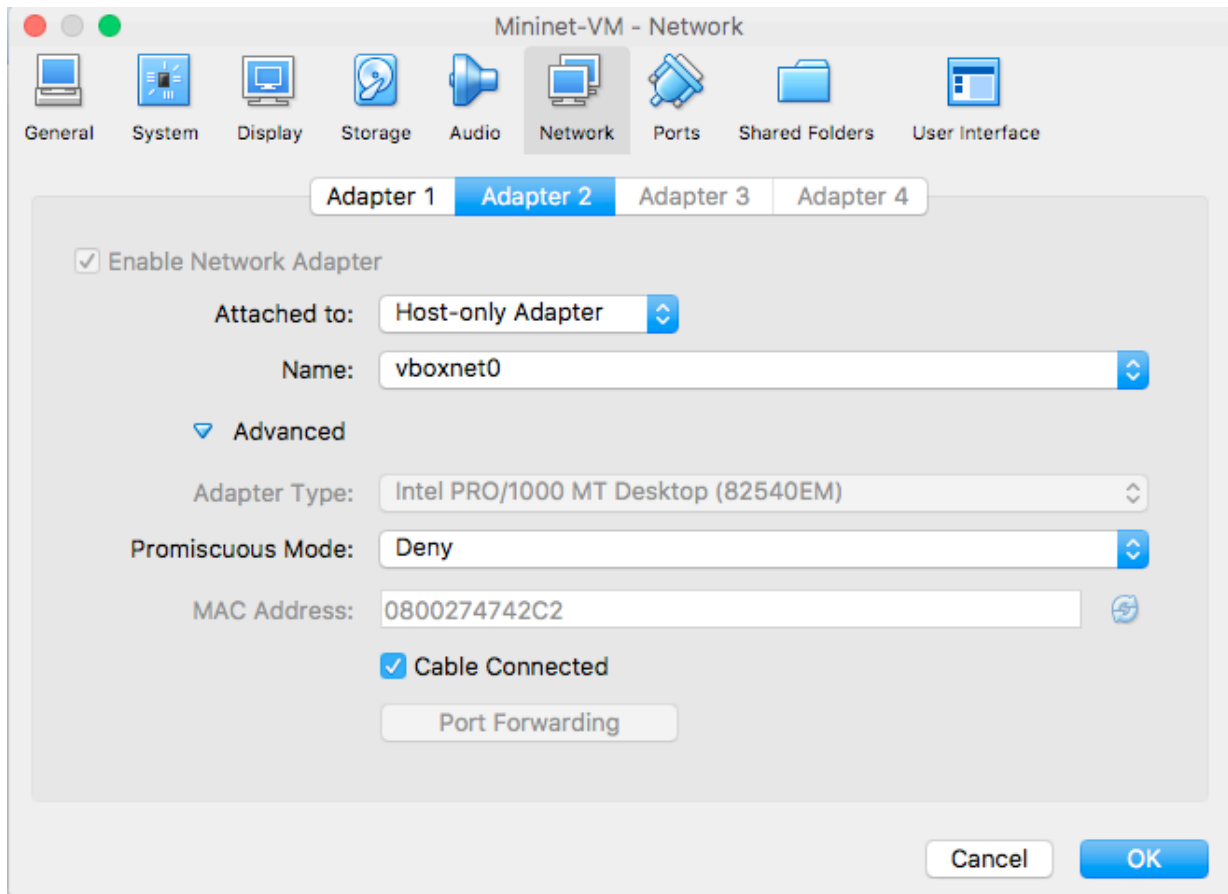


Figure 6

7. Click OK to save.

8. Double click on the VM to boot.

9. Upon boot, you will be greeted with the following screen as shown in figure 7:

```
Ubuntu 14.04 LTS mininet-vm tty1
mininet-vm login:
```

Figure 7

10. Use the username and password ‘mininet’ to log-in as shown in figure 8:

```
Ubuntu 14.04 LTS mininet-vm tty1
mininet-vm login: mininet
Password:
Last login: Thu Dec 15 07:50:19 PST 2016 on tty1
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ sudo dhclient eth1
RTNETLINK answers: File exists
mininet@mininet-vm:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:9d:a3:7a
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:272 errors:0 dropped:0 overruns:0 frame:0
          TX packets:328 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29973 (29.9 KB)  TX bytes:29828 (29.8 KB)

mininet@mininet-vm:~$ _
```

Figure 8

11. Run the command ‘sudo dhclient eth1’ to configure an IP for the 2nd network interface we added earlier.

12. Verify that the 2nd network interface has an IP address by using the command ‘ifconfig eth1’

13. Verify that the first network interface has an IP address by using the command ‘ifconfig eth0’

```
Ubuntu 14.04 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Thu Dec 15 07:50:19 PST 2016 on tty1
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ sudo dhclient eth1
RTNETLINK answers: File exists
mininet@mininet-vm:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:9d:a3:7a
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:272 errors:0 dropped:0 overruns:0 frame:0
          TX packets:328 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29973 (29.9 KB)  TX bytes:29828 (29.8 KB)

mininet@mininet-vm:~$
mininet@mininet-vm:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:47:42:c2
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2408 (2.4 KB)  TX bytes:684 (684.0 B)

mininet@mininet-vm:~$
```

Figure 9

14. Update mininet by running the ‘sudo apt-get update’ command as shown in figure 10:

```
root@mininet-vm:~# sudo apt-get update
Hit http://security.ubuntu.com trusty-security InRelease
Ign http://us.archive.ubuntu.com trusty InRelease
Get:1 http://security.ubuntu.com trusty-security/main Sources [123 kB]
Get:2 http://us.archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:3 http://security.ubuntu.com trusty-security/restricted Sources [4,613 B]
Get:4 http://security.ubuntu.com trusty-security/universe Sources [46.5 kB]
Get:5 http://us.archive.ubuntu.com trusty-backports InRelease [65.9 kB]
Get:6 http://security.ubuntu.com trusty-security/multiverse Sources [3,199 B]
Get:7 http://security.ubuntu.com trusty-security/main amd64 Packages [570 kB]
Get:8 http://us.archive.ubuntu.com trusty Release.gpg [933 B]
Get:9 http://us.archive.ubuntu.com trusty-updates/main Sources [388 kB]
Get:10 http://security.ubuntu.com trusty-security/restricted amd64 Packages [13.3 kB]
Get:11 http://security.ubuntu.com trusty-security/universe amd64 Packages [146 kB]
Get:12 http://security.ubuntu.com trusty-security/multiverse amd64 Packages [4,140 B]
Get:13 http://security.ubuntu.com trusty-security/main i386 Packages [528 kB]
Get:14 http://us.archive.ubuntu.com trusty-updates/restricted Sources [5,888 B]
Get:15 http://security.ubuntu.com trusty-security/restricted i386 Packages [13.1 kB]
Get:16 http://us.archive.ubuntu.com trusty-updates/universe Sources [170 kB]
Get:17 http://security.ubuntu.com trusty-security/universe i386 Packages [147 kB]
Get:18 http://security.ubuntu.com trusty-security/multiverse i386 Packages [4,301 B]
Get:19 http://us.archive.ubuntu.com trusty-updates/multiverse Sources [7,523 B]
Get:20 http://security.ubuntu.com trusty-security/main Translation-en [314 kB]
Get:21 http://us.archive.ubuntu.com trusty-updates/main amd64 Packages [934 kB]
Get:22 http://security.ubuntu.com trusty-security/multiverse Translation-en [2,201 B]
Get:23 http://security.ubuntu.com trusty-security/restricted Translation-en [3,349 B]
Get:24 http://security.ubuntu.com trusty-security/universe Translation-en [86.6
```

Figure 10

The apt-get update command updates the sources required by mininet in order to download and install software as well as update the operating system.

15. Once complete, run the ‘sudo apt-get install lxde xinit’ command as shown in figure 11:

```
mininet@mininet-vm:~$ sudo apt-get install lxde xinit_
```

Figure 11

This command downloads the 'lxde' GUI environment and also installs the 'xinit' command that is required to run lxde GUI environment from command shell.

16. Once the download is complete, run the 'sudo startx' command to run the lxde environment with root privileges as shown in figure 12:

```
mininet@mininet-vm:~$ sudo startx
```

Figure 12

Once lxde starts up, you will be able to see the desktop.

Once on the desktop click on the menu button, go to accessories and select 'gedit' as shown in figure 13:

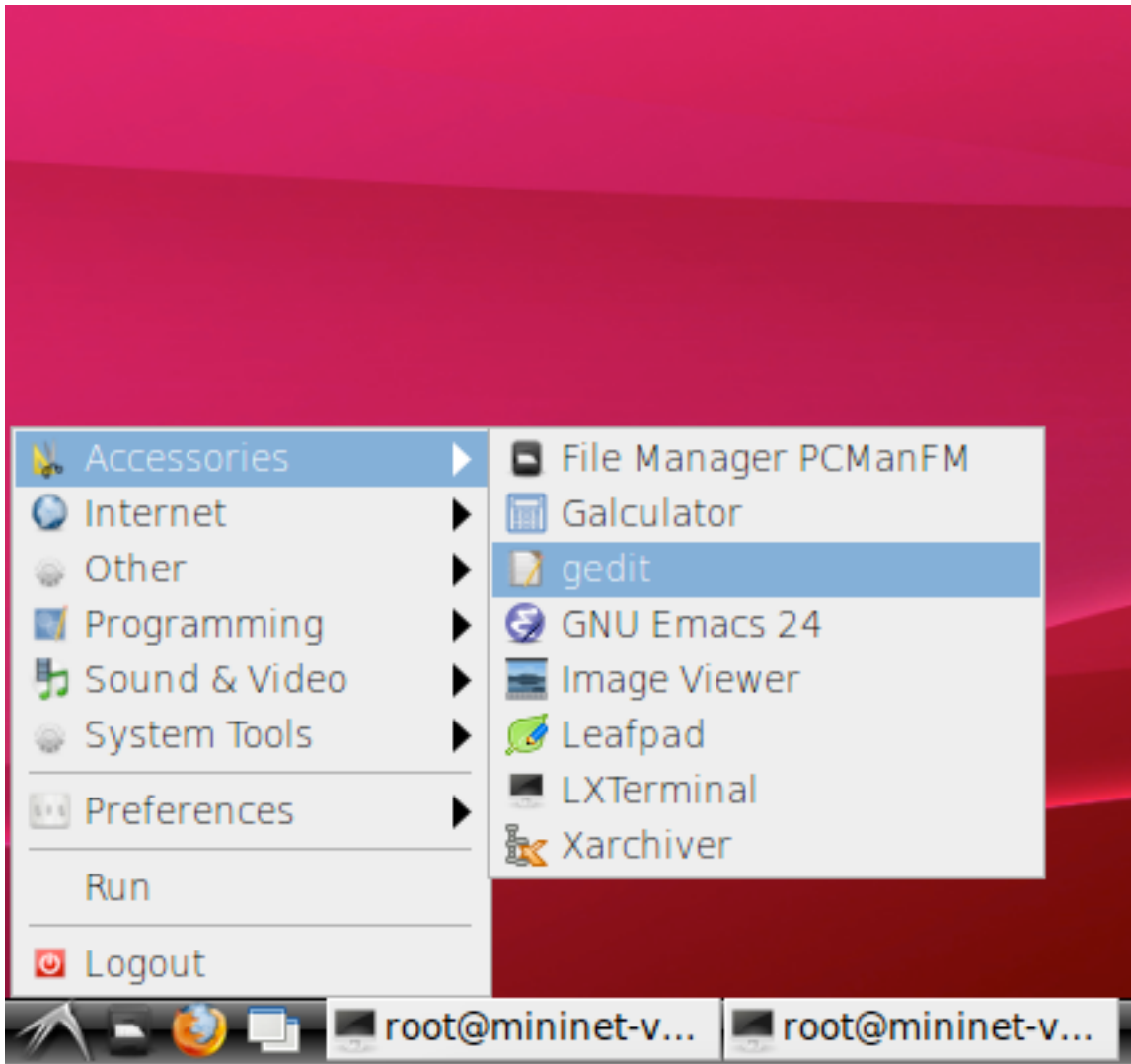
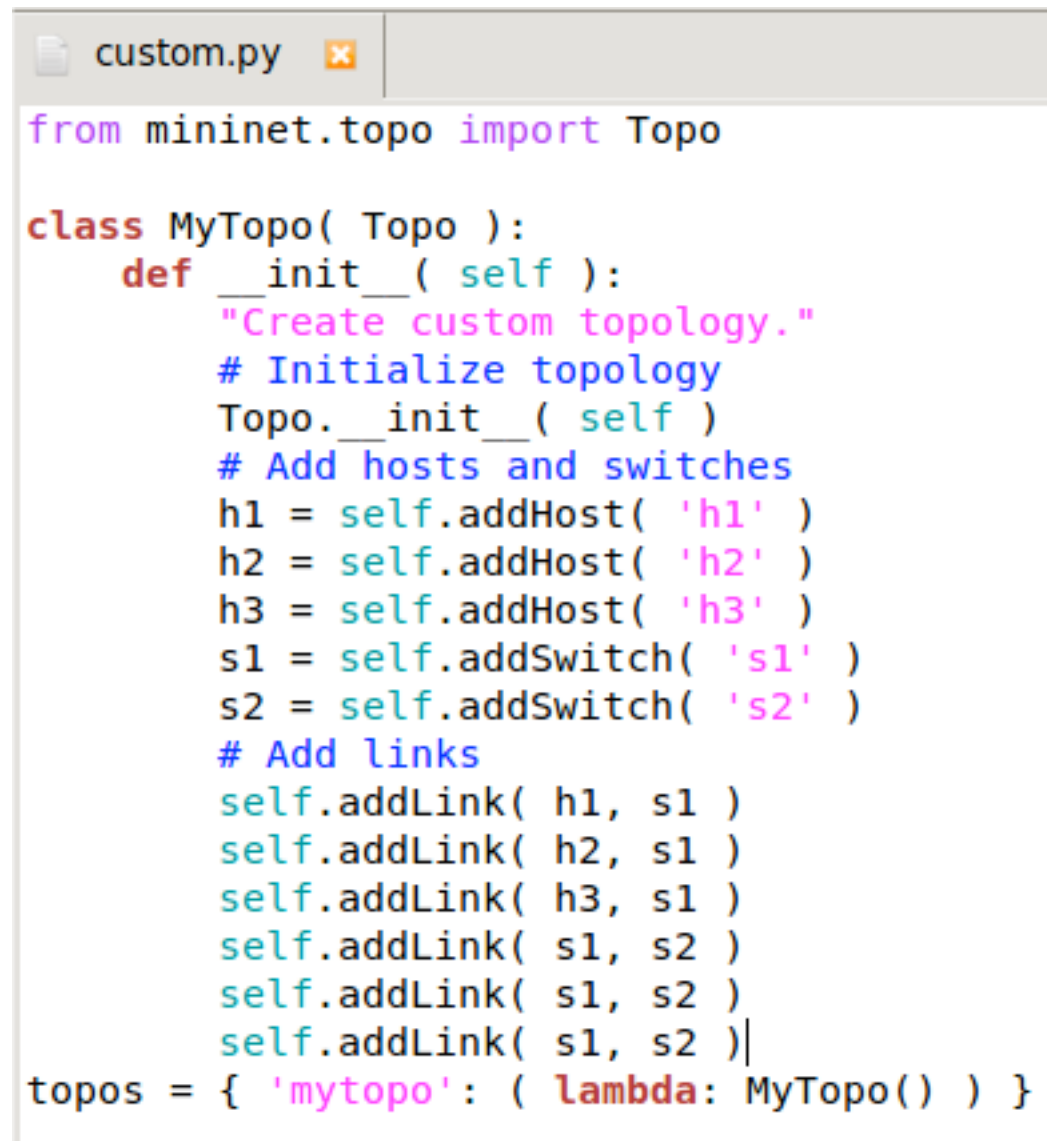


Figure 13

Alternatively, if you do not see the gedit application, open LXTerminal and type:
'sudo apt-get install gedit'

This will install the gedit application and you will be able to access it from within the accessories sub menu.

17. In the gedit application, write the following code and save it to the /mininet/custom directory:



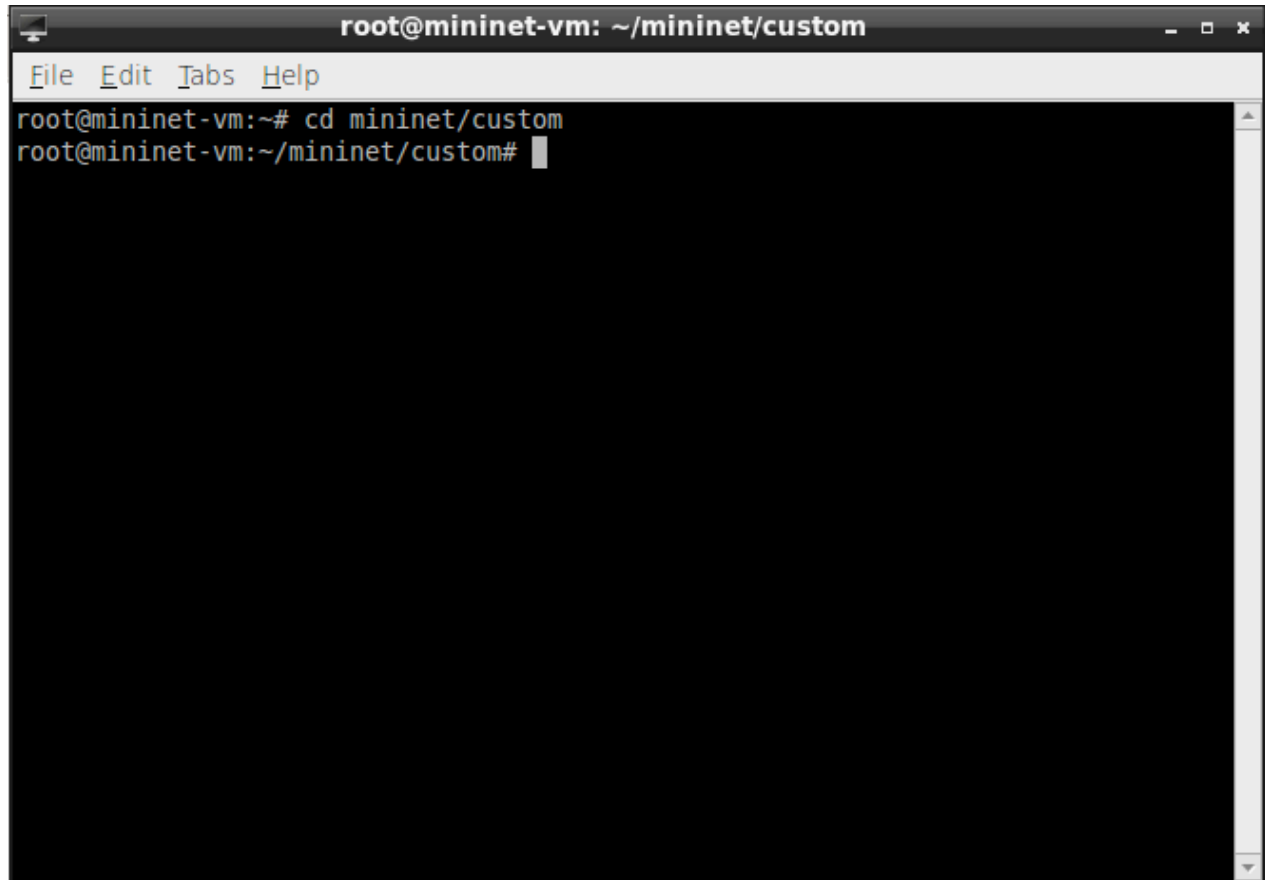
```
from mininet.topo import Topo

class MyTopo( Topo ):
    def __init__( self ):
        "Create custom topology."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        # Add links
        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( h3, s1 )
        self.addLink( s1, s2 )
        self.addLink( s1, s2 )
        self.addLink( s1, s2 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure 14

This code adds three hosts and two switches. The hosts and switches are connected as shown in the network diagram in figure 2.

18. Open a terminal window and type in the following commands as shown in figure 15:

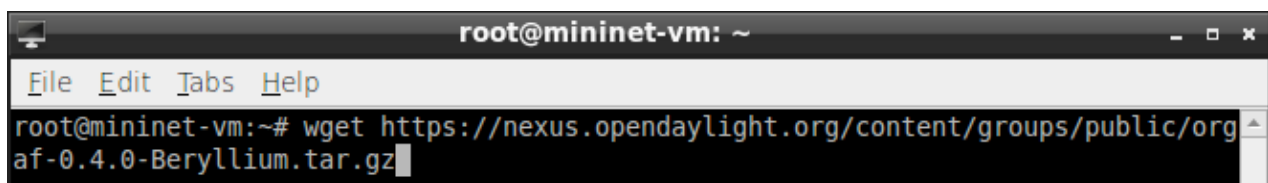
A terminal window titled 'root@mininet-vm: ~/mininet/custom'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal content shows the prompt 'root@mininet-vm:~#', followed by the command 'cd mininet/custom', and then the prompt 'root@mininet-vm:~/mininet/custom#' with a cursor. The rest of the terminal area is black.

```
root@mininet-vm: ~/mininet/custom
File Edit Tabs Help
root@mininet-vm:~# cd mininet/custom
root@mininet-vm:~/mininet/custom#
```

Figure 15

The `cd mininet/custom` command changes the current working directory to `/mininet/custom`.

19. Open another terminal window and type the commands as shown in figure 16:

A terminal window titled 'root@mininet-vm: ~'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal content shows the prompt 'root@mininet-vm:~#', followed by the command 'wget https://nexus.opendaylight.org/content/groups/public/org.opendaylight.ryu.af-0.4.0-Beryllium.tar.gz', and then the prompt 'af-0.4.0-Beryllium.tar.gz' with a cursor. The rest of the terminal area is black.

```
root@mininet-vm: ~
File Edit Tabs Help
root@mininet-vm:~# wget https://nexus.opendaylight.org/content/groups/public/org
af-0.4.0-Beryllium.tar.gz
```

Figure 16

20. This command downloads the opendaylight SDN controller from the opendaylight.org website.

The downloaded file is compressed in a .tar format. This command extracts the tar file into a folder with the same name.



```
root@mininet-vm: ~  
File Edit Tabs Help  
root@mininet-vm:~# tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz
```

Figure 17

21. Type ls to see the file list in the home folder as show in figure 18:



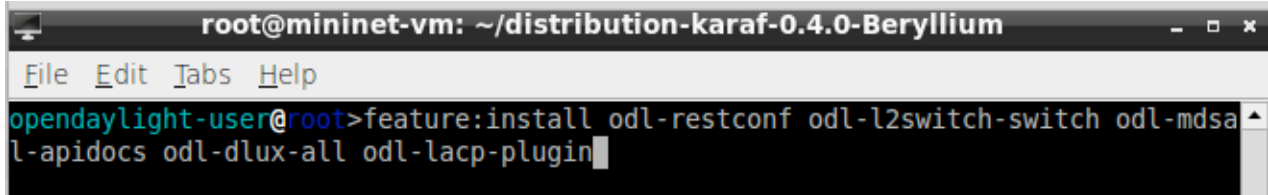
```
root@mininet-vm: ~/distribution-karaf-0.4.0-Beryllium  
File Edit Tabs Help  
root@mininet-vm:~# ls  
bashrc Documents oflops  
bashrc~ Downloads oftest  
Desktop install-mininet-vm.sh openflow  
distribution-karaf-0.4.0-Beryllium loxigen pox  
distribution-karaf-0.4.0-Beryllium.tar.gz mininet Templates  
root@mininet-vm:~# cd distribution-karaf-0.4.0-Beryllium/  
root@mininet-vm:~/distribution-karaf-0.4.0-Beryllium# ./bin/karaf
```

Figure 18

22. Type cd distributon-karaf-0.4.0-Beryllium to change the working directory to that of the OpenDaylight controller.

OpenDaylight is packaged into a karaf container. Typing ./bin/karaf starts the OpenDaylight SDN controller.

23. After the OpenDaylight controller starts, type in the following command as shown in figure 19:



```
root@mininet-vm: ~/distribution-karaf-0.4.0-Beryllium
File Edit Tabs Help
opendaylight-user@root>feature:install odl-restconf odl-l2switch-switch odl-mdsa
l-apidocs odl-dlux-all odl-lacp-plugin
```

Figure 19

This command installs the following modules into the OpenDaylight controller [3]:

1. odl-restconf: Allows access to the restconf API
2. odl-l2switch-switch: Provides network functionality similar to an Ethernet switch
3. odl-mdsal-apidocs: Allows access to the YANG API
4. odl-dlux-all: Installs the OpenDaylight GUI
5. odl-lacp-plugin: Installs the Link Aggregation Control Protocol on the OpenDaylight controller.

24. Upon launching the OpenDaylight controller and installing the features mentioned above, open Firefox and type ‘127.0.0.1:8181/index.html’ and log in using the username and password ‘admin’ to access the dlux GUI.

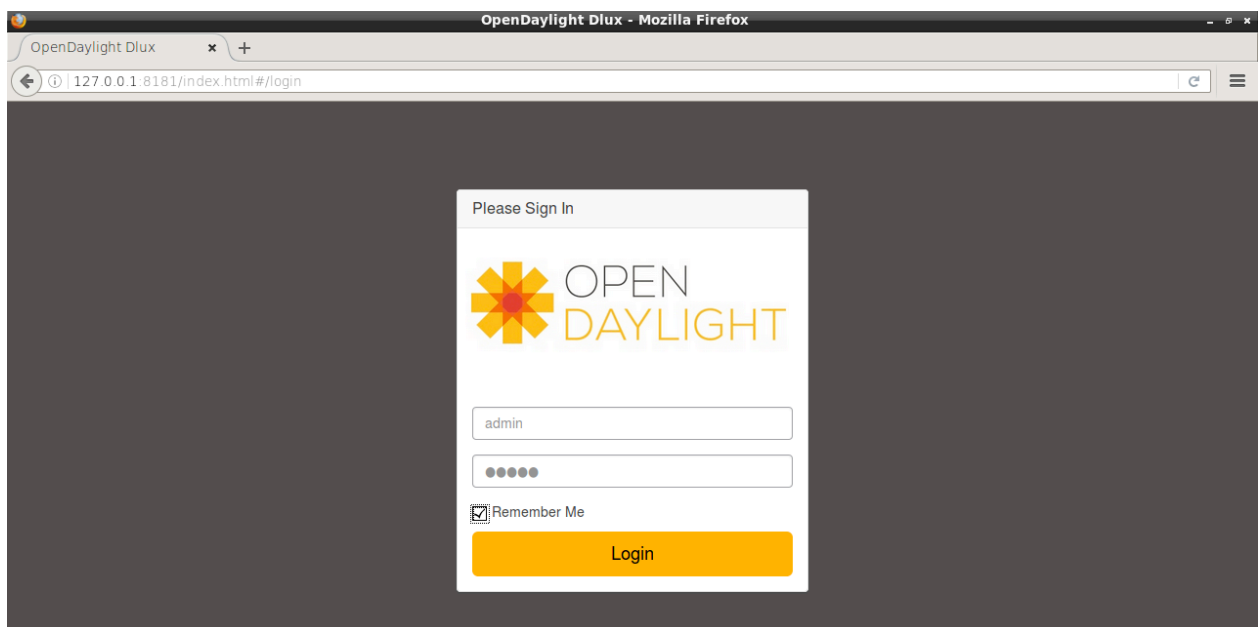
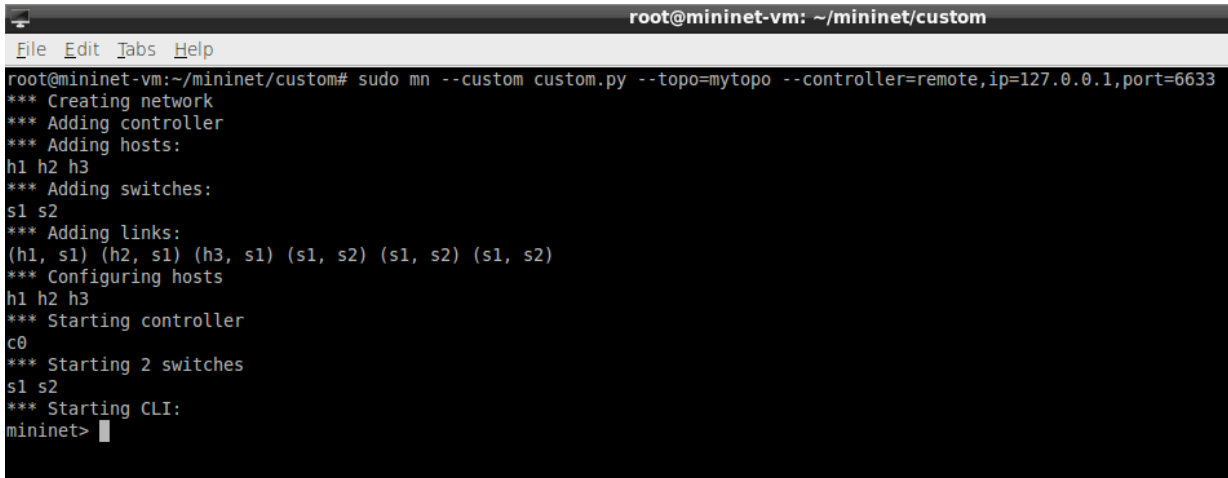


Figure 20

From this GUI, the user can access the graphical representation of the topology, access the rest config to push flows and retrieve topology information and also access the YANG visualizer.

25. Open the other terminal window (the one pointing towards /mininet/custom) and type the following commands as shown in figure 21:



```
root@mininet-vm: ~/mininet/custom
File Edit Tabs Help
root@mininet-vm:~/mininet/custom# sudo mn --custom custom.py --topo=mytopo --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (s1, s2) (s1, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet> █
```

Figure 21

26. This command launches the custom topology we programmed earlier with the following options:

1. --topo=mytopo: Pass an object to instantiate the topology we created
2. --controller=remote: Tell the mininet system to use a remote controller and not the default controller. The parameters ip and port tell the system where the controller is hosted.

Here we can see that the hosts H1 H2 and H3 have been added and switches S1 and S2 have also been added.

The links (h1,s1), (h2,s1), (h3,s1), (s1,s2), (s1,s2), (s1,s2) have also been created and the controller c0 has been added to the topology as well.

The links and the network information can be obtained by running the command 'links' and 'net' separately in the terminal as shown in figure 22:



```
Mininet-VM [Running]
root@mininet-vm: ~/mininet/custom
File Edit Tabs Help
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
s1-eth4<->s2-eth1 (OK OK)
s1-eth5<->s2-eth2 (OK OK)
s1-eth6<->s2-eth3 (OK OK)
mininet>
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:s2-eth1 s1-eth5:s2-eth2 s1-eth6:s2-eth3
s2 lo: s2-eth1:s1-eth4 s2-eth2:s1-eth5 s2-eth3:s1-eth6
c0
mininet>
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s2
mininet> █
```

Figure 22

The ‘links’ command shows the status of the links. This command is ideal to debug a network in case a link between the network element goes down.

The ‘net’ command shows how the network elements are connected to each other and on what interfaces. This command is ideal to debug the network in the event a host or a switch goes down.

In order to push the topology to the controller, the first few packets that are sent to the switches are sent up to the controller for processing. During this first step, the controller installs flows into the switches and creates a network map that can be visualized on the OpenDaylight DLUX GUI.

27. Run the following ping commands to generate traffic that will flow from between the hosts:

```
root@mininet-vm: ~/mininet/custom
File Edit Tabs Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.590 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.052 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.052/0.247/0.590/0.243 ms
mininet>
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.601 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.147 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.051/0.213/0.601/0.227 ms
mininet>
mininet>
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.295 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.142 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.118 ms
^C
```

Figure 23

The pingall command is a quick command to ping all hosts from all other hosts. Running that command makes the controller aware of the topology and the flows are pushed to the switches.

Run individual ping commands between every host to make sure each host can reach all the other hosts.

28. Within the DLUX GUI, click on the nodes option and click on the Node Connectors option on openflow:1 switch

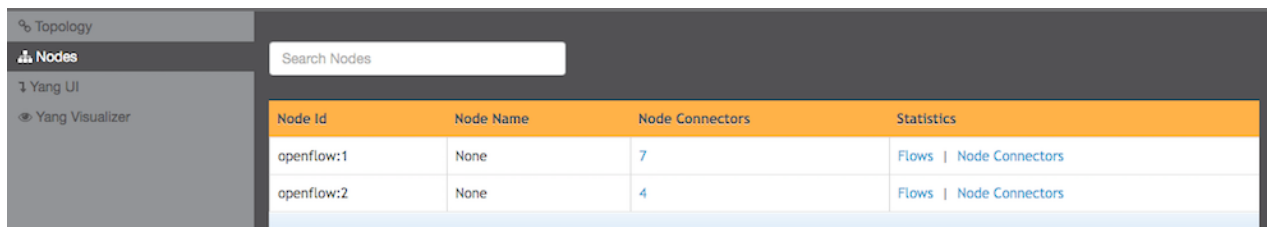


Figure 24

The Node Connectors screen will open and will show all the ports on the switch ‘openflow:1’ and the statistics of all the ports:

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:1:3	17	48	1386	4017	0	0	0	0	0	0	0	0
openflow:1:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0
openflow:1:4	27	27	2295	2295	0	0	0	0	0	0	0	0
openflow:1:5	27	27	2295	2295	0	0	0	0	0	0	0	0
openflow:1:6	27	43	2295	3527	0	0	0	0	0	0	0	0
openflow:1:2	16	48	1316	4017	0	0	0	0	0	0	0	0
openflow:1:1	15	45	1218	3723	0	0	0	0	0	0	0	0

Figure 25

Go back and click on Node Connectors option for ‘openflow:2’ switch:

Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
openflow:2:3	45	29	3697	2465	0	0	0	0	0	0	0	0
openflow:2:2	29	29	2465	2465	0	0	0	0	0	0	0	0
openflow:2:1	29	29	2465	2465	0	0	0	0	0	0	0	0
openflow:2:LOCAL	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26

As we can see in figure 26 on both switches there is traffic generated on each port (in the columns Rx Pkts and Tx Pkts) which means all links are being utilized and thus, the load balancing is being achieved on all of these links when communicating from one host to another.

Issues and Challenges

The challenges faced during the project in the implementation of this project were the identification of the limitations of mininet, the default mininet controller (OVS Controller) as well as the POX controller.

Initially, efforts were made to develop a load balancing scheme for the default OVS Controller. However, upon reading the documentation [4] as well as dry running the source code [5], it was discovered that the default OVS Controller (controller-8) [6] only worked in the L2 learning mode and any attempt to utilize multiple links between the two switches resulted in loops. That meant in order to implement multi-link load balancing in the default OVS Controller, one would have to re-code the controller from the ground up. It was decided that we were going to focus our efforts elsewhere as better and more modular SDN controllers are already available and the default OVS Controller is there to introduce people to the basic concepts of SDN.

The second controller that we looked at was the POX controller. POX is an open source development platform for Python based SDN control applications such as OpenFlow SDN Controllers [7]. POX supports modules that can be used to run the controller in a certain mode and while POX supported load balancing, it did not support load balancing over multiple links [8] as the POX controller fundamentally functioned in a L2 learning mode as well. Furthermore, POX does not support OpenFlow 1.3 and only supports OpenFlow 1.0, which is a deal breaker as load balancing over a network is not possible, in a reasonable and non-experimental way, in OpenFlow 1.0 [9]. POX is able to perform load balance across multiple servers, i.e. only multi-server round-robin load balancing was available with the POX controller and thus, a different controller had to be chosen.

The challenge faced with mininet was that mininet did not have the ability to implement multiple links between switches and hosts. Upon further research, it was discovered that this feature was later introduced in mininet version 2.2.0 and higher so the entire implementation had to be re-done on this newer version of mininet.

Conclusion

Load balancing is a technique that is in place today to provide high network throughput as well as provide a means to utilize extra links in a network that would otherwise be redundant. With the advent of Software Defined Networks, load balancers and load balancing techniques can now utilize the power of SDN controllers that have supreme control over the network topology, allowing for greater control over how the data is handled in a network. We looked at Load Balancing via Link Aggregation Control Protocol in the Open Daylight SDN controller, balancing load over multiple links in this project, which is dynamic and represents a real world use case.

Organizations often run multiple links between multiple switches in order to provide redundancy as well as link aggregation and load balancing. However, traditional approaches are more cumbersome as each of the above mentioned features have to be implemented individually.

With the dawn of SDN based load balancing techniques in controllers such as Open Daylight, all of the above mentioned techniques can be implemented within a single configuration. This reduces the complexity of the network as well as the operating costs, which in the end is a primary goal for Software Defined Networks.

Appendices & References

Mininet network topology

```
from mininet.topo import Topo
class MyTopo( Topo ):
    def __init__( self ):
        "Create custom topology."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        # Add links
        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( h3, s1 )
        self.addLink( s1, s2 )
        self.addLink( s1, s2 )
        self.addLink( s1, s2 )
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

References

[1] Nick McKeown; et al. (April 2008). "OpenFlow: Enabling innovation in campus networks". ACM Communications Review. Retrieved 2009-11-02.

[2] https://wiki.opendaylight.org/view/LACP:Lithium:User_Guide#LACP

[3] <http://www.brianlinkletter.com/using-the-openswitch-sdn-controller-with-the-mininet-network-emulator/>

[4] <http://www.manualpages.de/FreeBSD/FreeBSD-ports-9.0-RELEASE/man8/ovs-controller.8.html>

[5] https://sourcecodebrowser.com/openswitch/1.1.0~pre2.g2.ea763e0e/ovs-controller_8c_source.html

[6] <https://github.com/osrg/openswitch/blob/master/utilities/ovs-controller.8.in>

[7] <http://searchsdn.techtarget.com/definition/POX>

[8] <https://www.mail-archive.com/pox-dev@lists.noxrepo.org/msg01093.html>

[9] <http://sdnhub.org/tutorials/pox/>

<https://www.opennetworking.org/sdn-resources/>

<https://wiki.opendaylight.org/>

<http://www.brianlinkletter.com/>

Table of Figures

Figure 1: SDN Architecture

Figure 2: Project Network Topology

Figure 3: Virtual Box Import Appliance Screen# 1

Figure 4: Virtual Box Import Appliance Screen# 2

Figure 5: Virtual Box Import Appliance Screen# 3

Figure 6: Virtual Machine Settings Page

Figure 7: Mininet Login Screen

Figure 8: Mininet DHCP Configuration

Figure 9: Mininet Network Adapter Configuration

Figure 10: Mininet Update Screen

Figure 11: Mininet: GUI Installation

Figure 12: Mininet: GUI Launch Command

Figure 13: Mininet: Launching Gedit

Figure 14: Mininet: Custom Topology Script

Figure 15: Mininet: Navigating To /custom/

Figure 16: Mininet: Download OpenDaylight

Figure 17: Mininet: Extracting OpenDaylight from tar file

Figure 18: Mininet: Running OpenDaylight

Figure 19: OpenDaylight: Installing features

Figure 20: OpenDaylight: Accessing DLUX GUI

Figure 21: Mininet: Running Custom Topology

Figure 22: Mininet: Links, Net and Nodes Commands

Figure 23: Mininet: Ping Commands

Figure 24: OpenDaylight: DLUX GUI Nodes Screen

Figure 25: OpenDaylight: Openflow:1 Node Connector Statistics Screen

Figure 26: OpenDaylight: Openflow:2 Node Connector Statistics Screen