Software Defined Load Balancer

Team members:

Kiran Kancheti (kxk143630)

Sandeep Batchu (sxb148430)

Siri Venkat Vemuri(sxv141130)

Pranesh Vyas (pxv141830)

Contents

Abstract

This project develops a Load balancing POX Controller which makes flow balancing decisions based on statistics gathered from interfaces of switches and nodes in the network. Statistics include bandwidth and latency. A network is created with three switches in mininet with three hosts, and based on the statistics gathered from the network, flow modifications are done. This project has been initially implemented with floodlight controller, but due to insufficient documentation, we shifted to OpenDaylight controller. While installing opendaylight controller, we faced many road blocks for installing and building in windows to add a module, and we used POX controller which has good support and documentation. Mininet is used with Oracle VM for creating the network.

Initial Set Up

Install Oracle VM on the machine

Import the Mininet 2.2.1 OVF into the VM

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

Now, start the Mininet and do **ifconfig –a** which gives us the host address

We can use this host address to connect to Putty and WinSCP

Technology

Mininet was used to simulate the network

POX controller was in built in Mininet and also POX had sound documentation so we preferred it over others

Set Up Challenges

Flood Light Controller

This is the first controller we tried to use because of Java language, but due to insufficient documentation, we shifted to pox controller.

OpenDaylight

OpenDaylight has very poor support and documentation for implementing the controller and is deployed using Apache Karaf.

ISSUES

Below are the errors which we got while installing opendaylight.

```
opendaylight-user*@*root>feature:in-stall odl-bgpcep-all
>
> Error executing command: Could not start bundle
> mvn:org.opendaylight.controller/config-m-anager/0.2.8-Helium-SR3 in
> feature(s) odl-config-manager-0.2.8-Helium-SR3: The activator
> org.opendaylight.controller.config.manag-er.impl.osgi.ConfigManagerActivator
> for bundle org.opendaylight.controller.config-manag-er is invalid
```

For increasing memory, following option is used. set MAVEN_OPTS=-Xmx512m -XX:MaxPermSize=128m

Then, we had a roadblock because of the following error:

"Failed to execute goal org.codehaus.enunciate:maven-enunciate-plugin"

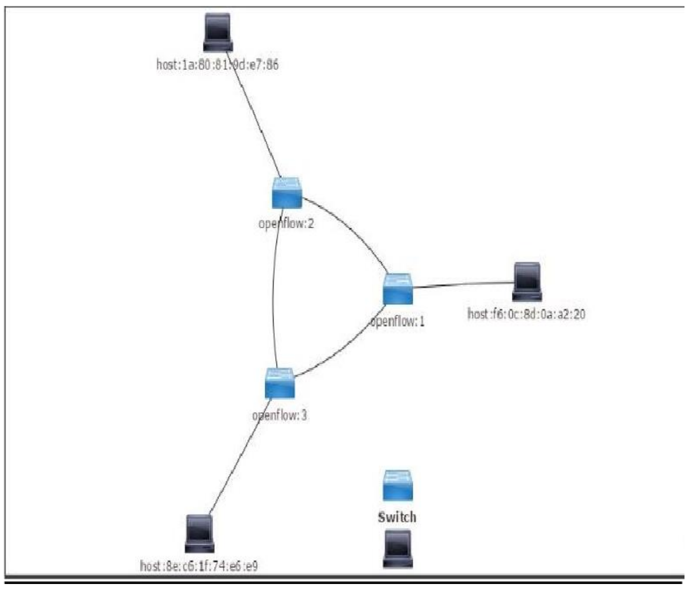
Wireshark: We had issues while setting up Wireshark. So, we took the tcp dump and analyzed the traffic using tcp commands.

Network Architecture – Topology

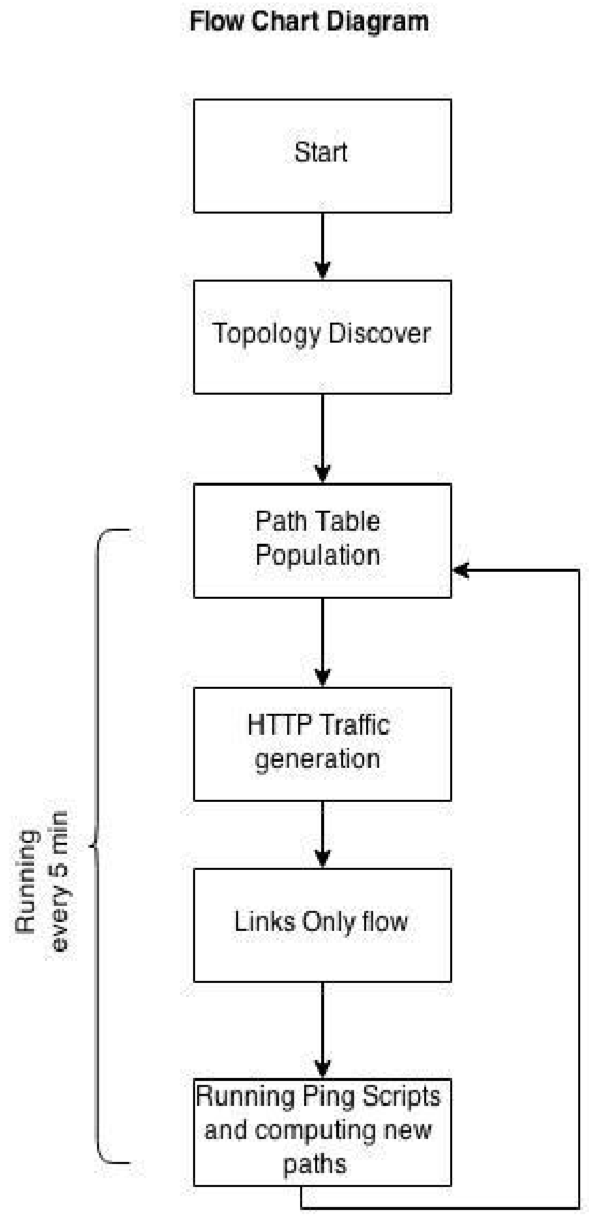
Network contains architecture contains three switches and three hosts as shown in the figure shown below. Three hosts are labelled h1, h2 and h3. Switches are labelled as S1, S2 and S3.Initially, a spanning tree is read by the controller from a local file. For every 5 minutes, this file is updated based on the congestion in the traffic.

This network is started using the following command.

```
sudo mn --custom topo.py --topo mytopo --controller remote
```



Load Balancer Application



In the diagram shown above, after the controller is started, topology of the network is discovered using the populate_flow_table() method. This function will read paths.file to get default spanning tree for the network. This data is populated in path_table data structure stored in controller.
Now, we add two listeners, one for LinkEvent which triggers handle_links method when a link is added between switches and PacketIn event which triggers handle_packetin when a switch gets a packet.
handle_links method updates portmap data structure with values of ports to be forwarded for every link between switches.
handle_packetin method is called when controller receives a packet from switch. After receiving the packet, based on the entries in path_table data structure, we update the flow by using ofp_flow_mod() method in pox library.
Now, after the network is ready, traffic is generated using http scripts. After a

significant traffic is generated for 5 minutes. Now, the links between all switches are activated by calling test_flow_add() method. After that, ping statistics is run to get the new paths and is updated in paths.file. This process continues for every 5 minutes.

Compiling Instructions

- Controller.py should be placed in pox folder of the pox installation.
- Paths.file should also be placed in pox folder.
- All the ping scripts and http traffic generation scripts and stat.py should be copied to mininet home directory.
- Now run the controller using following command inside pox installation directory:
    - **./pox.py log.level –DEBUG controller openflow.discovery**
    - Running this command also includes openflow discovery with controller.
- Now run mininet using following command:
    - **sudo mn --custom topo.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 different route.

Statistics

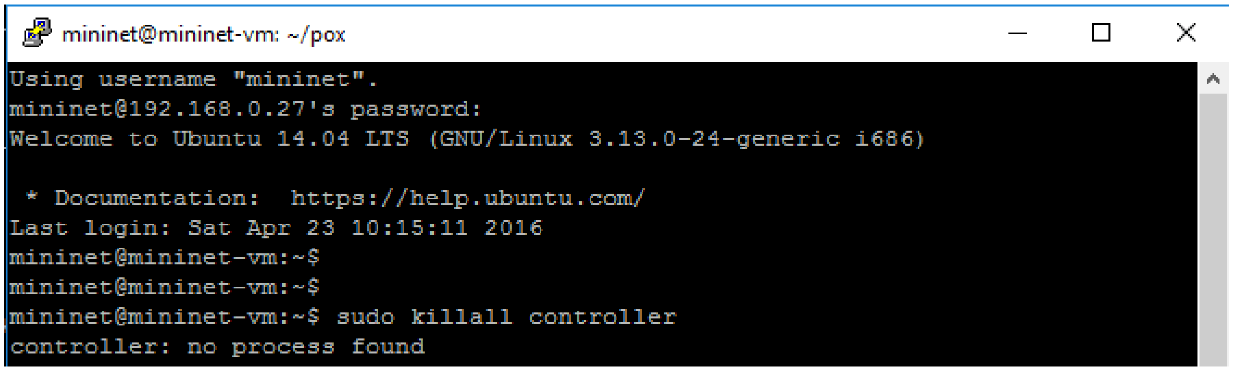
To get the statistics from the switches, a set of python scripts are written which will ping all the nodes from a particular node. After pinging the nodes, average latency between each node is collected. Depending upon the average latency, an alternative path with lowest overall latency is selected.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

```
import urllib, os
os.system("ping -c 5 10.0.0.2 | tail -1| awk '{print $4}' | cut -d '/' -f 2 > ping_stat_1_2")
os.system("ping -c 5 10.0.0.3 | tail -1| awk '{print $4}' | cut -d '/' -f 2 > ping_stat_1_3")
```

Execution example with Screenshots

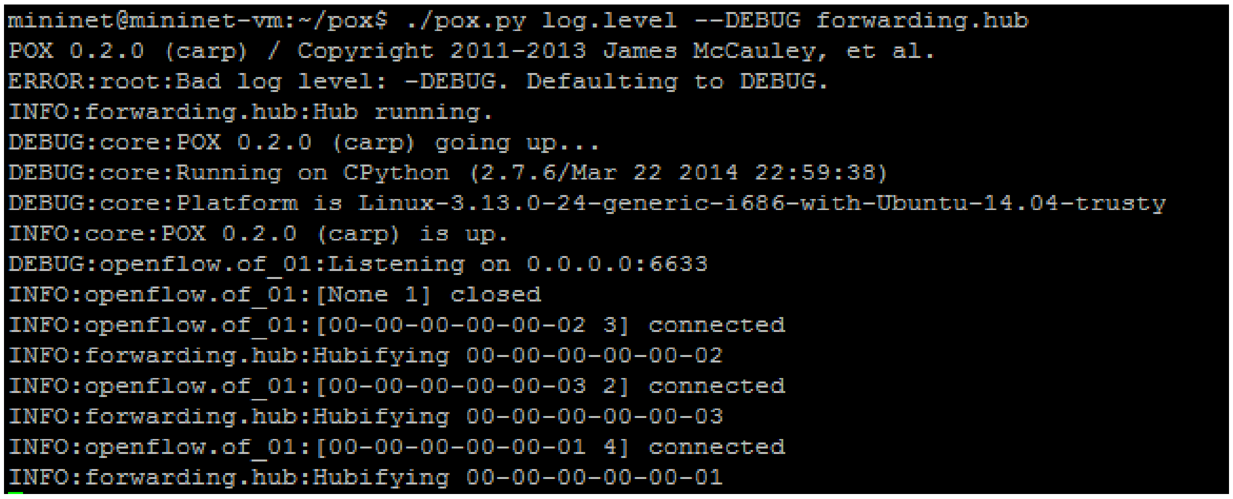
Mininet Clean Up

Kill existing controller and clear existing topology



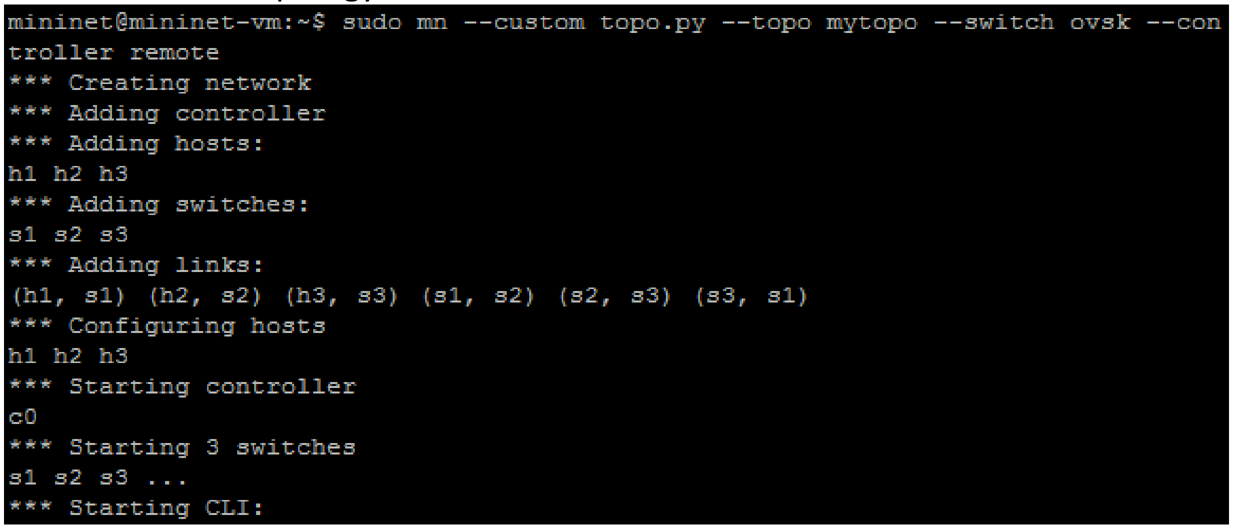
```
Using username "mininet".
mininet@192.168.0.27's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic i686)

 * Documentation:  https://help.ubuntu.com/
Last login: Sat Apr 23 10:15:11 2016
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo killall controller
controller: no process found
```

Starting POX controller and the topology on mininet



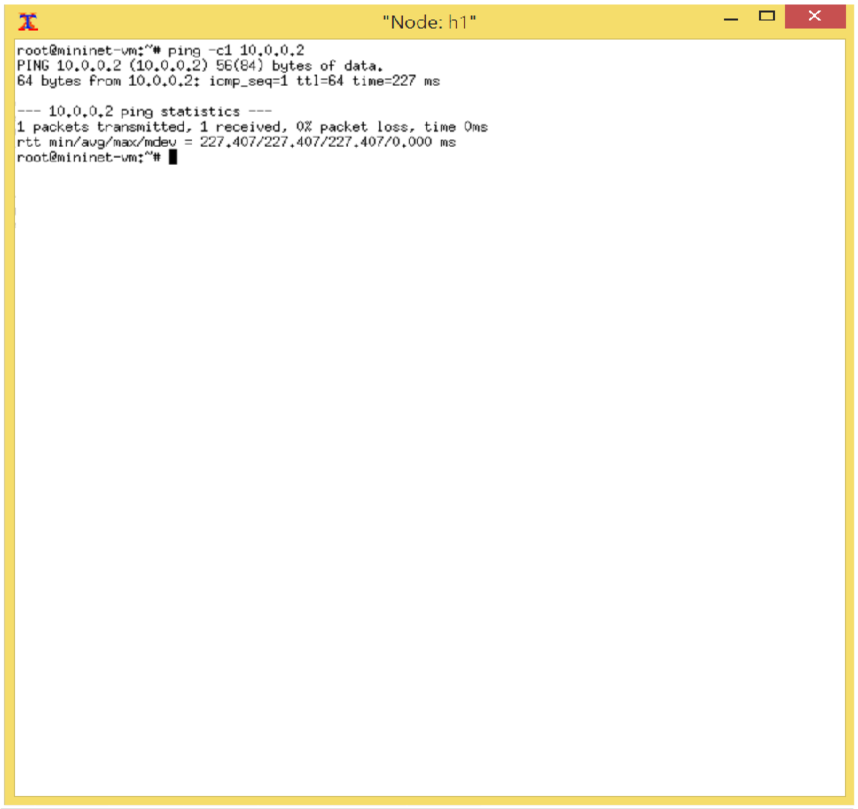
```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG forwarding.hub
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
ERROR:root:Bad log level: -DEBUG. Defaulting to DEBUG.
INFO:forwarding.hub:Hub running.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:38)
DEBUG:core:Platform is Linux-3.13.0-24-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

Create Custom Topology

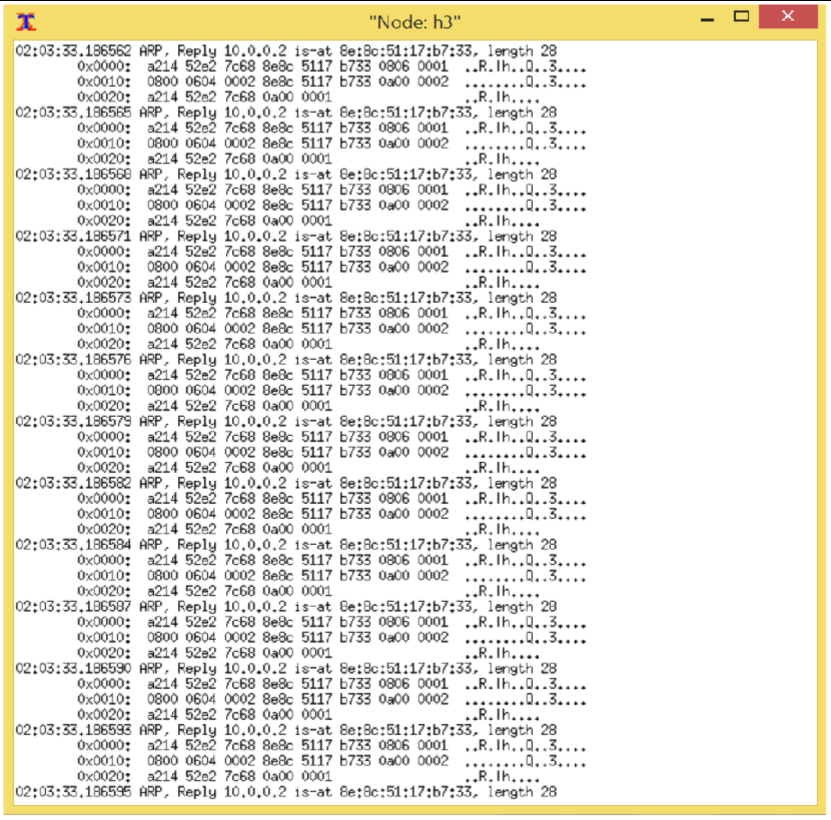


```
mininet@mininet-vm:~$ sudo mn --custom topo.py --topo mytopo --switch ovsk --con
troller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s2) (s2, s3) (s3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

Images of Nodes h1 and h3 running



```
root@mininet-vm:~# ping -c1 10.0.0.2
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=227 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 227.407/227.407/227.407/0.000 ms
root@mininet-vm:~#
```



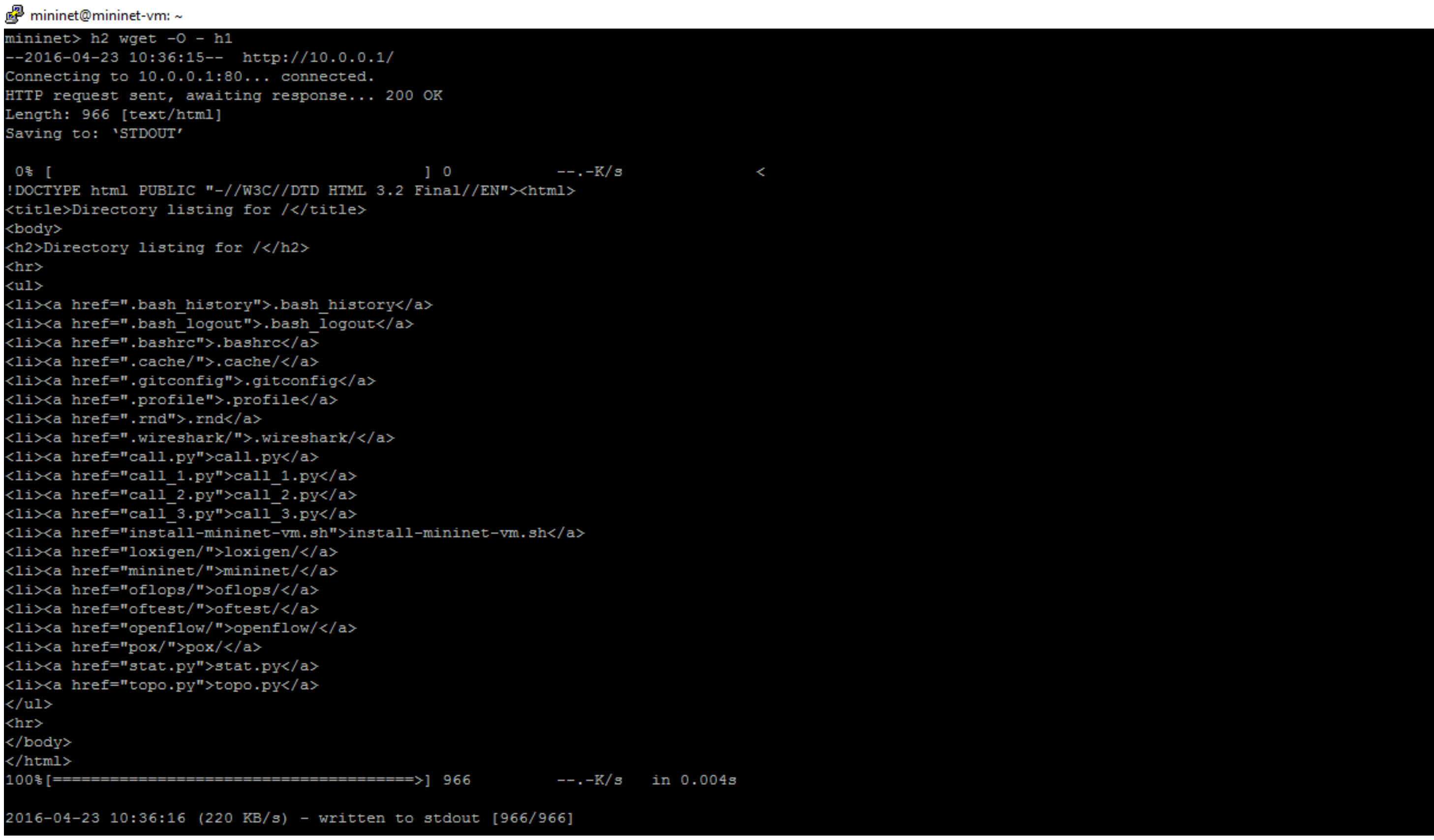
```
02:03:33.186562 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186565 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186568 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186571 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186573 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186576 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186579 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186582 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186584 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186587 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186590 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186593 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
        0x0000:  a214 52e2 7c68 8e8c 5117 b733 0806 0001  ..R.|h..Q..3....
        0x0010:  0800 0604 0002 8e8c 5117 b733 0a00 0002  ........Q..3....
        0x0020:  a214 52e2 7c68 0a00 0001                 ..R.|h....
02:03:33.186595 ARP, Reply 10.0.0.2 is-at 8e:8c:51:17:b7:33, length 28
```

Traffic is generated by starting the http web server

mininet> h1 python -m SimpleHTTPServer 80 &

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

mininet> h2 wget -O - h1



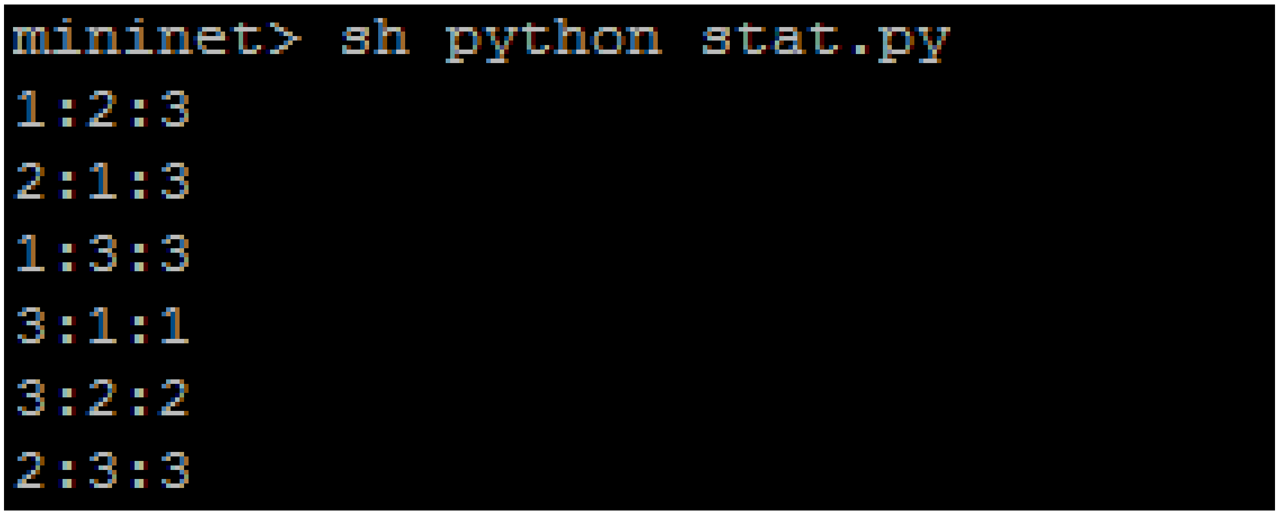
```
mininet@mininet-vm: ~
mininet> h2 wget -O - h1
--2016-04-23 10:36:15--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 966 [text/html]
Saving to: 'STDOUT'

 0% [                                       ] 0           --.-K/s              <
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gitconfig">.gitconfig</a>
<li><a href=".profile">.profile</a>
<li><a href=".rnd">.rnd</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href="call.py">call.py</a>
<li><a href="call_1.py">call_1.py</a>
<li><a href="call_2.py">call_2.py</a>
<li><a href="call_3.py">call_3.py</a>
<li><a href="install-mininet-vm.sh">install-mininet-vm.sh</a>
<li><a href="loxigen/">loxigen/</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="stat.py">stat.py</a>
<li><a href="topo.py">topo.py</a>
</ul>
<hr>
</body>
</html>
100%[======================================>] 966         --.-K/s   in 0.004s

2016-04-23 10:36:16 (220 KB/s) - written to stdout [966/966]
```

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

```
mininet@mininet-vm:~$ h2 python call_2.py
mininet@mininet-vm:~$ h3 python call_3.py
mininet@mininet-vm:~$ h1 python call_1.py
```

Script in call_1.py

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



```
mininet> sh python stat.py
1:2:3
2:1:3
1:3:3
3:1:1
3:2:2
2:3:3
```

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

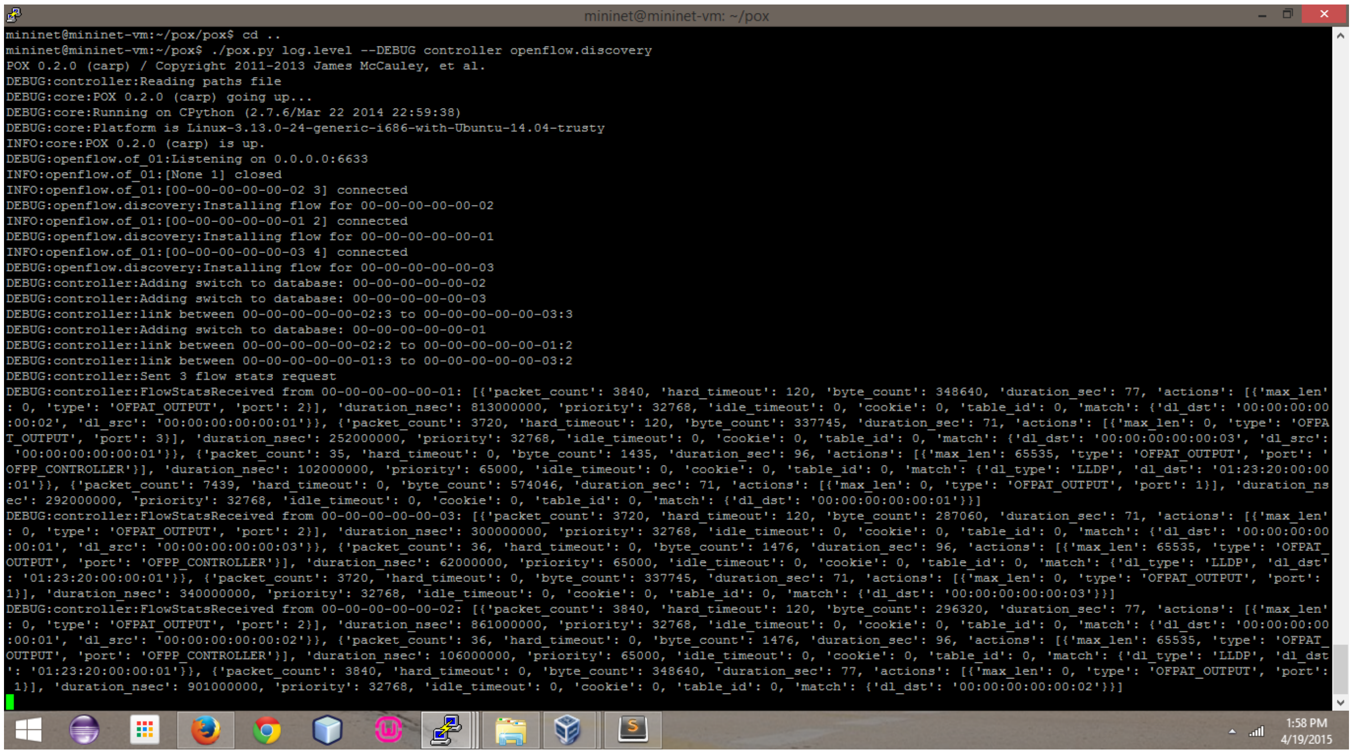
Statistics

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. This statistics are used by controller to modify flows.

```
import urllib, os
# This script is used for pinging the client 2 and 3 from 1
os.system("ping -c 5 10.0.0.2 | tail -1| awk '{print $4}' | cut -d '/' -f 2 >>
ping_stat_1_2")
os.system("ping -c 5 10.0.0.3 | tail -1| awk '{print $4}' | cut -d '/' -f 2 >>
ping_stat_1_3")mininet@mininet-vm:~$
```

Challenges Faced

- While sending ARP request is broadcasted, it was generating huge traffic. This caused congestion and taking more time for ping to work. To solve this problem, whenever ARP request is received, MAC address is hardcoded and sent to the requested host. This is handled in handle_packetin method. Following is the screenshot of wireshark where we can see that after a ARP broadcast, we send a ofp_packet_out message.
- POX controller provided a method which gives flow statistics which can be gathered by adding listener “FlowStatsReceived”. But this method doesn’t give enough information to handle flow statistics. So we wrote some scripts which can give latency in the network. Following is the controller log of the flow statistics:



```
mininet@mininet-vm:~/pox/pox$ cd ..
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG controller openflow.discovery
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:controller:Reading paths file
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:38)
DEBUG:core:Platform is Linux-3.13.0-24-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-03
DEBUG:controller:Adding switch to database: 00-00-00-00-00-02
DEBUG:controller:Adding switch to database: 00-00-00-00-00-03
DEBUG:controller:link between 00-00-00-00-00-02:3 to 00-00-00-00-00-03:3
DEBUG:controller:Adding switch to database: 00-00-00-00-00-01
DEBUG:controller:link between 00-00-00-00-00-02:2 to 00-00-00-00-00-01:2
DEBUG:controller:link between 00-00-00-00-00-01:3 to 00-00-00-00-00-03:2
DEBUG:controller:Sent 3 flow stats request
DEBUG:controller:FlowStatsReceived from 00-00-00-00-00-01: [{'packet_count': 3840, 'hard_timeout': 120, 'byte_count': 348640, 'duration_sec': 77, 'actions': [{'max_len'
: 0, 'type': 'OFPAT_OUTPUT', 'port': 2}], 'duration_nsec': 813000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00
:00:02', 'dl_src': '00:00:00:00:00:01'}}, {'packet_count': 3720, 'hard_timeout': 120, 'byte_count': 337745, 'duration_sec': 71, 'actions': [{'max_len': 0, 'type': 'OFPA
T_OUTPUT', 'port': 3}], 'duration_nsec': 252000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00:00:03', 'dl_src':
 '00:00:00:00:00:01'}}, {'packet_count': 35, 'hard_timeout': 0, 'byte_count': 1435, 'duration_sec': 96, 'actions': [{'max_len': 65535, 'type': 'OFPAT_OUTPUT', 'port': '
OFPP_CONTROLLER'}], 'duration_nsec': 102000000, 'priority': 65000, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_type': 'LLDP', 'dl_dst': '01:23:20:00:00
:01'}}, {'packet_count': 7439, 'hard_timeout': 0, 'byte_count': 574046, 'duration_sec': 71, 'actions': [{'max_len': 0, 'type': 'OFPAT_OUTPUT', 'port': 1}], 'duration_ns
ec': 292000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00:00:01'}}]
DEBUG:controller:FlowStatsReceived from 00-00-00-00-00-03: [{'packet_count': 3720, 'hard_timeout': 120, 'byte_count': 287060, 'duration_sec': 71, 'actions': [{'max_len'
: 0, 'type': 'OFPAT_OUTPUT', 'port': 2}], 'duration_nsec': 300000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00
:00:01', 'dl_src': '00:00:00:00:00:03'}}, {'packet_count': 36, 'hard_timeout': 0, 'byte_count': 1476, 'duration_sec': 96, 'actions': [{'max_len': 65535, 'type': 'OFPAT_
OUTPUT', 'port': 'OFPP_CONTROLLER'}], 'duration_nsec': 62000000, 'priority': 65000, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_type': 'LLDP', 'dl_dst'
: '01:23:20:00:00:01'}}, {'packet_count': 3720, 'hard_timeout': 0, 'byte_count': 337745, 'duration_sec': 71, 'actions': [{'max_len': 0, 'type': 'OFPAT_OUTPUT', 'port':
1}], 'duration_nsec': 340000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00:00:03'}}]
DEBUG:controller:FlowStatsReceived from 00-00-00-00-00-02: [{'packet_count': 3840, 'hard_timeout': 120, 'byte_count': 296320, 'duration_sec': 77, 'actions': [{'max_len'
: 0, 'type': 'OFPAT_OUTPUT', 'port': 2}], 'duration_nsec': 861000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00
:00:01', 'dl_src': '00:00:00:00:00:02'}}, {'packet_count': 36, 'hard_timeout': 0, 'byte_count': 1476, 'duration_sec': 96, 'actions': [{'max_len': 65535, 'type': 'OFPAT_
OUTPUT', 'port': 'OFPP_CONTROLLER'}], 'duration_nsec': 106000000, 'priority': 65000, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_type': 'LLDP', 'dl_dst
': '01:23:20:00:00:01'}}, {'packet_count': 3840, 'hard_timeout': 0, 'byte_count': 348640, 'duration_sec': 77, 'actions': [{'max_len': 0, 'type': 'OFPAT_OUTPUT', 'port':
 1}], 'duration_nsec': 901000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_dst': '00:00:00:00:00:02'}}]
```

Conclusion

In this project, we got hands on experience to use SDN Controller and add functionality on top of it. In addition mininet is also used for network simulation. We faced various obstacles in the integration and found various workarounds to get the desired funcitonality.

References

1. http://mininet.org/walkthrough/
2. http://archive.openflow.org/wp/learnmore/
3. http://conferences.sigcomm.org/sigcomm/2013/papers/sigcomm/p541.pdf
4. http://www.opendaylight.org/software
5. http://networkstatic.net/installing-mininet-opendaylight-open-vswitch/
6. https://openflow.stanford.edu/display/ONL/POX+Wiki