Software Defined Load Balancer using POX Controller

Nikhil Peshwe(NXP180001)

Pratik Kamath(PXK170010)

Mihir Joshi(MXJ180007)

Pankaj Jhawar(PXJ180019)

Ameya Chitnis(AAC180004)

CONTENTS:

INTRODUCTION:

The main objective of our project is to build a load balancing software network using POX controller. The POX controller balances flow of packets based on information retrieved from interfaces of switches and nodes in a network. Statistics mainly consists of two things.

1. Bandwidth
2. Latency

In our project we have created a software network in “mininet” which consists of following components.

1. Switches (quantity: 3)
2. Hosts (quantity: 3)

Once we run the/ simulate the projects the HTTP traffic gets generated in network. Based on these HTTP traffic data collected/ retrieved by network we have also performed a flow modification (along with showing paths from source to destination) in our project. Initially we tried to run complete project using “mininet” on virtual machine in windows but we faced some issues and because of that we shifted to Ubuntu. We also tried to use “Open Day light controller as it was similar to labs in class but due lack of documentation/ information we couldn’t use it and moved to POX controller. It provides good support as well as a lot of documentation is available.

INITIAL SETUP:

1) Install an Ubuntu system (virtual machine) on your personal computer.
2) Install mininet on ubuntu using following commands:

```
git clone git://github.com/mininet/mininet
```

```
mininet/util/install.sh -a
```

3) Then, in the network settings set up Adapter1 as Bridged Adapter.
4) Now, start the Mininet and do ifconfig –a which gives us the host address.

TECHNOLOGY USED:

Simulated network using mininet.

Mininet has in-built POX controller. There is a lot of documentation on POX had so we preferred it over others.

SETUP CHALLENGES:

- We first decided to use flood light controller but went for POX controller due to poor documentation in java for flood light controller.
- Initially we got error while connecting to the POX controller

```
root@ubuntu:~/pox# ./pox.py
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
ERROR:openflow.of_01:Error 98 while binding socket: Address already in use
ERROR:openflow.of_01: You may have another controller running.
ERROR:openflow.of_01: Use openflow.of_01 --port= to run POX on another port.
```

- OpenDayLight has poor support and documentation for implementing the controller using karaf and is more inconvenient to implement compared to POX.

NETWORK ARCHITECTURE – TOPOLOGY:

Network Architecture contains 3 switches and 3 hosts. The switches are labelled as H1, H2, H3 and the switches are labelled as S1, S2, S3. H1 is connected to S1, H2 is connected to S2 and H3 is connected to S3. All switches are cyclic connected. S1 is connected to S2, S2 is connected to S3 and S3 is connected to S1. Initially a spanning tree is read is read by the controller from local file. After every 5 minutes the file is updated according to the congestion in the network.

The network is created using following command:

sudo mn --custom topo.py --topo mytopo --controller remote,port=6633 –switch ovsk

LOAD BALANCER APPLICATION:

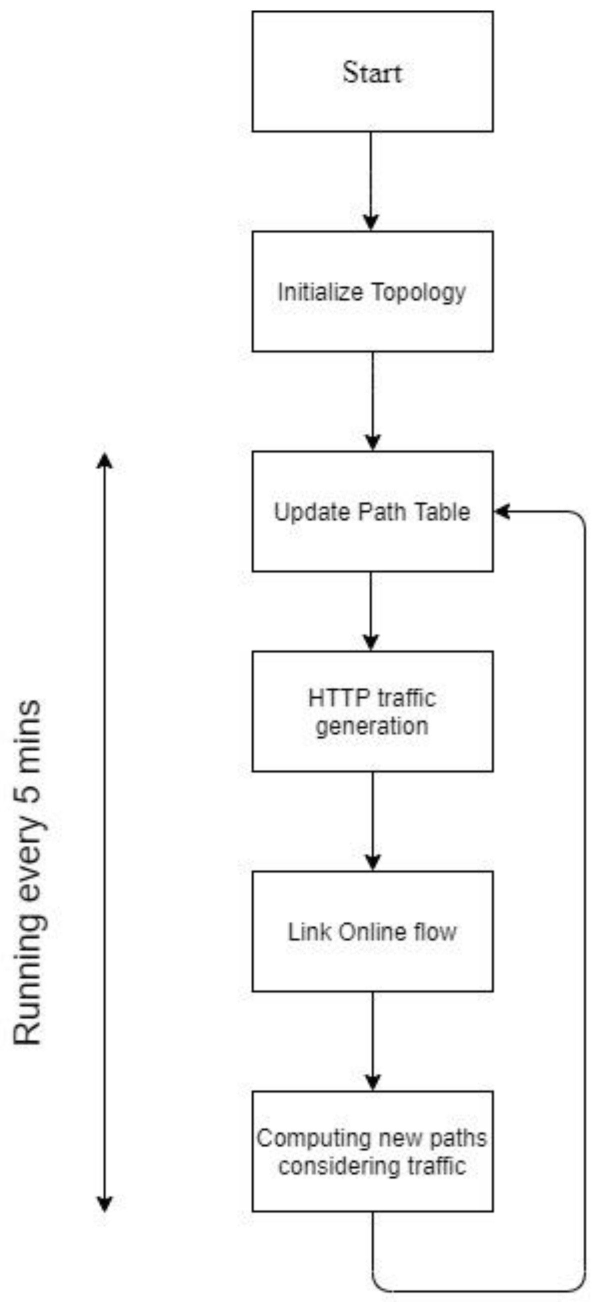


Fig:

Diagrammatic representation of Operation of load balancer

In this diagram, when the POX controller is started, the controller and the topology is created using topo.py. The controller will read the file to get the initial structure of the network.

We have 2 listeners handle_links() and handle_packetin(). Handle_links() is used to check the links added between different switches. Handle_packetin() method is to handle when a switch receives a packet. We update the flow in the pox library when the packet is received considering entries in page table. Traffic is generated in the network using https scripts. After traffic is added to the network we use test_flow_add() method to activate links between all switches. We use ping statistics to get new paths and add it in page table. This process is repeated every 5 minutes and new data flow paths are determined according to the traffic in the network.

INSTRUCTIONS TO EXECUTE THE PROGRAM:

Start Ubuntu and install mininet and create necessary directories to store respective files.

Follow given instructions:

Controller.py should be placed in pox folder of the pox installation.

1. Paths.file should also be placed in pox folder.
2. All the ping scripts and http traffic generation scripts and ComputeNewPaths.py should be copied to mininet home directory.
3. Now run the controller using following command inside pox installation directory: ./pox.py forwarding .hub openflow.discovery --eat-early-packets log.level –DEBUG. Running this command also includes openflow discovery with controller.
4. Use following command to create network: sudo mn --custom topo.py --topo mytopo --controller remote,port=6633– switch ovsk

5. Now run the http traffic generation scripts wait for 5 minutes and run ping scripts in the delay of 150 seconds to update the topology if required.
6. Now the traffic should be rerouted via different route.

RETRIVED STATISTICAL DATA:

We ran set of python scripts to get statistics from switches which will ping all the nodes from a particular node. After this ping operation we checked for latency between each node. Based on latency, we found an alternative path with lowest latency and used that path for packet transmission/ forwarding. These statistical data helped controller to decode alternative path.

Scripts:

These scripts are used to execute ping operation that is client 1 pings client 2.

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

The meaning of above command is explained below.

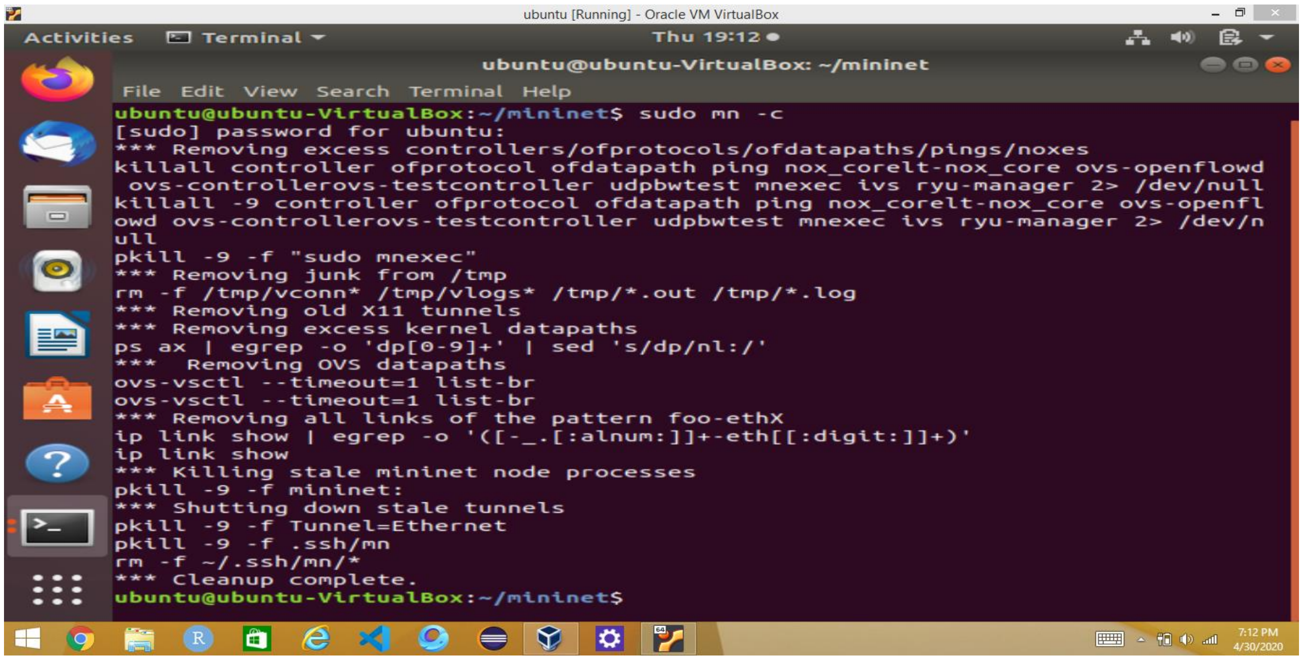
Sent 5 packets from to client 2 | choose the last entry of ping |find out ping time in ms| extract ping time |save in stat file

Similar was done between other hosts to calculate latencies.

EXECUTION EXAMPLES WITH SCREENSHOTS:

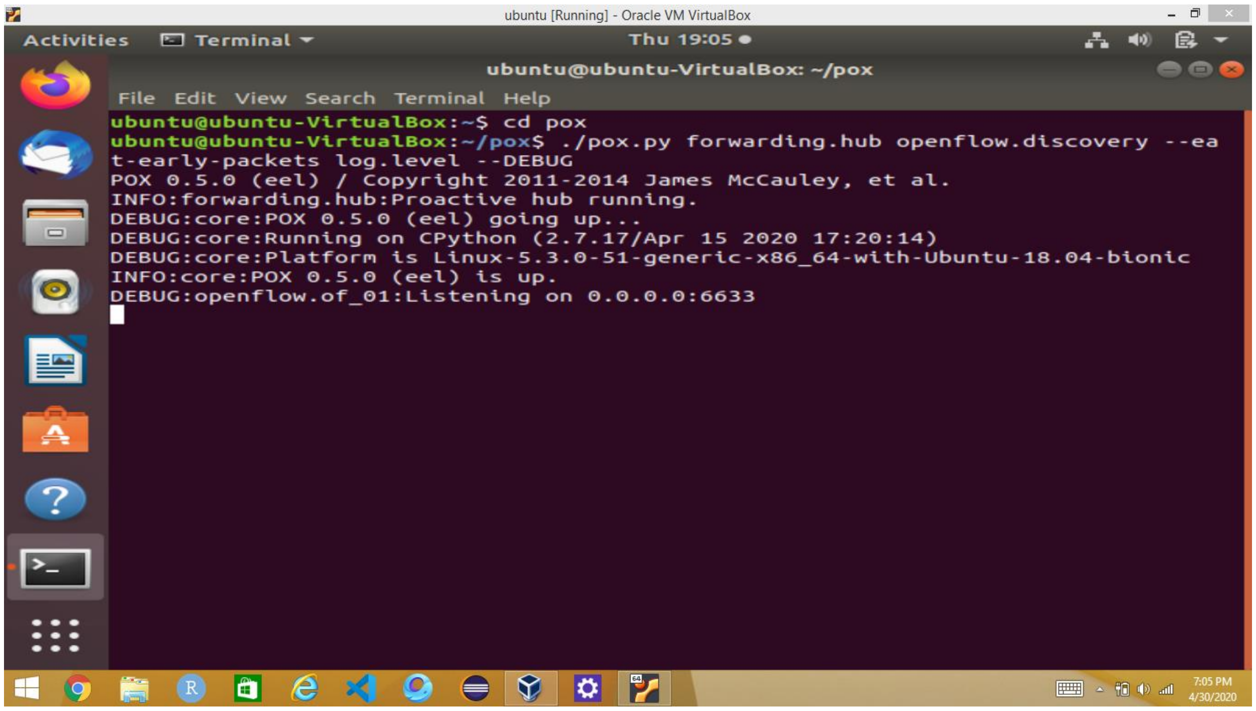
Mininet Clean Up

Kill existing controller and clear existing topology


ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:12
ubuntu@ubuntu-VirtualBox: ~/mininet
File Edit View Search Terminal Help
7:12 PM
4/30/2020

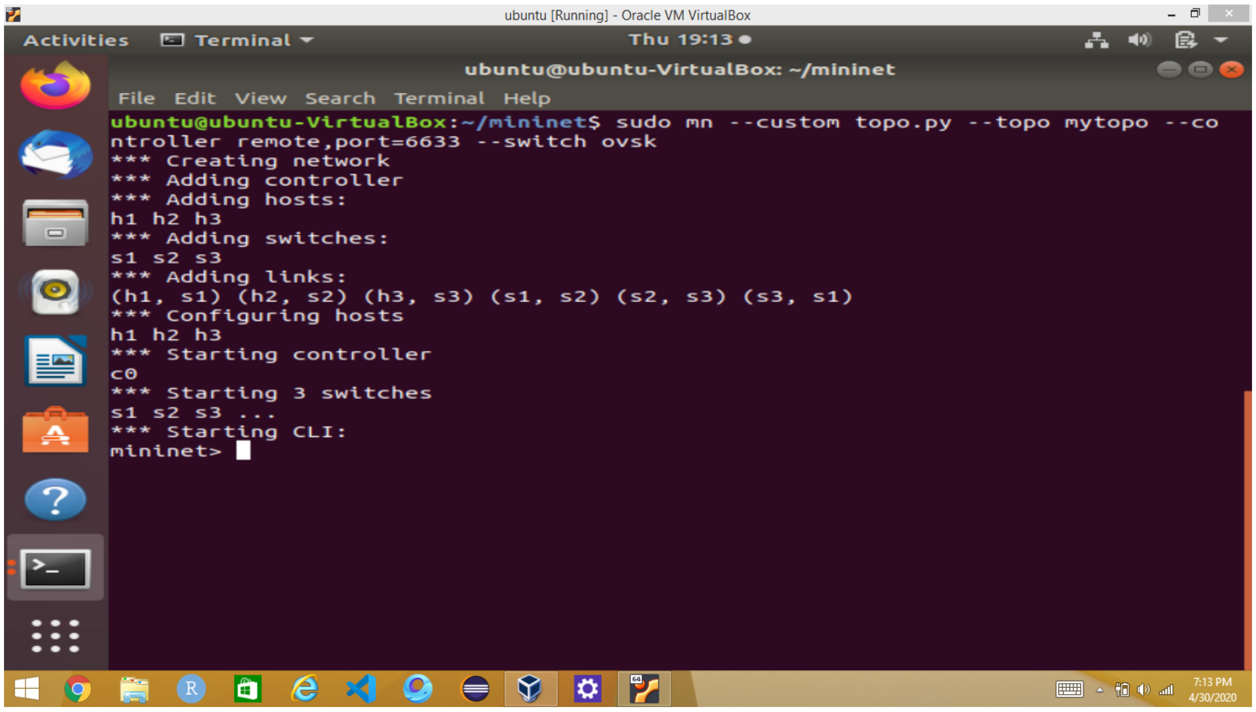
```
ubuntu@ubuntu-VirtualBox:~/mininet$ sudo mn -c
[sudo] password for ubuntu:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd
 ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openfl
owd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/n
ull
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ubuntu@ubuntu-VirtualBox:~/mininet$
```

Starting POX controller and the topology on mininet


ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:05
ubuntu@ubuntu-VirtualBox: ~/pox
File Edit View Search Terminal Help
7:05 PM
4/30/2020

```
ubuntu@ubuntu-VirtualBox:~$ cd pox
ubuntu@ubuntu-VirtualBox:~/pox$ ./pox.py forwarding.hub openflow.discovery --ea
t-early-packets log.level --DEBUG
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Apr 15 2020 17:20:14)
DEBUG:core:Platform is Linux-5.3.0-51-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

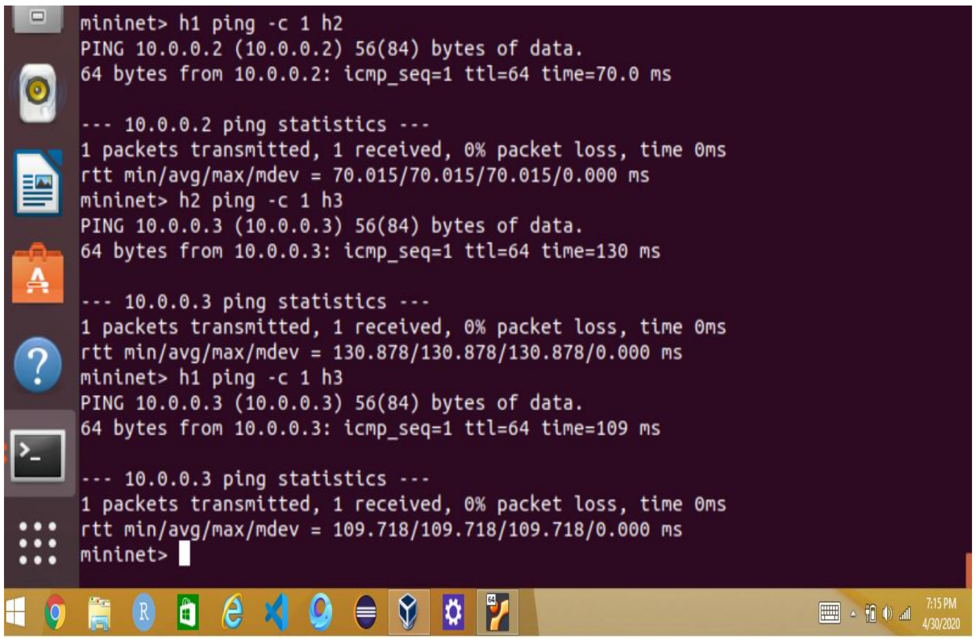
Create Custom Topology



ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:13
ubuntu@ubuntu-VirtualBox: ~/mininet
File Edit View Search Terminal Help
7:13 PM
4/30/2020

```
ubuntu@ubuntu-VirtualBox:~/mininet$ sudo mn --custom topo.py --topo mytopo --co
ntroller remote,port=6633 --switch ovsk
*** 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:
mininet>
```

Images of Nodes h1 and h3 running



```
mininet> h1 ping -c 1 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=70.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 70.015/70.015/70.015/0.000 ms
mininet> h2 ping -c 1 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=130 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 130.878/130.878/130.878/0.000 ms
mininet> h1 ping -c 1 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=109 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 109.718/109.718/109.718/0.000 ms
mininet>
```

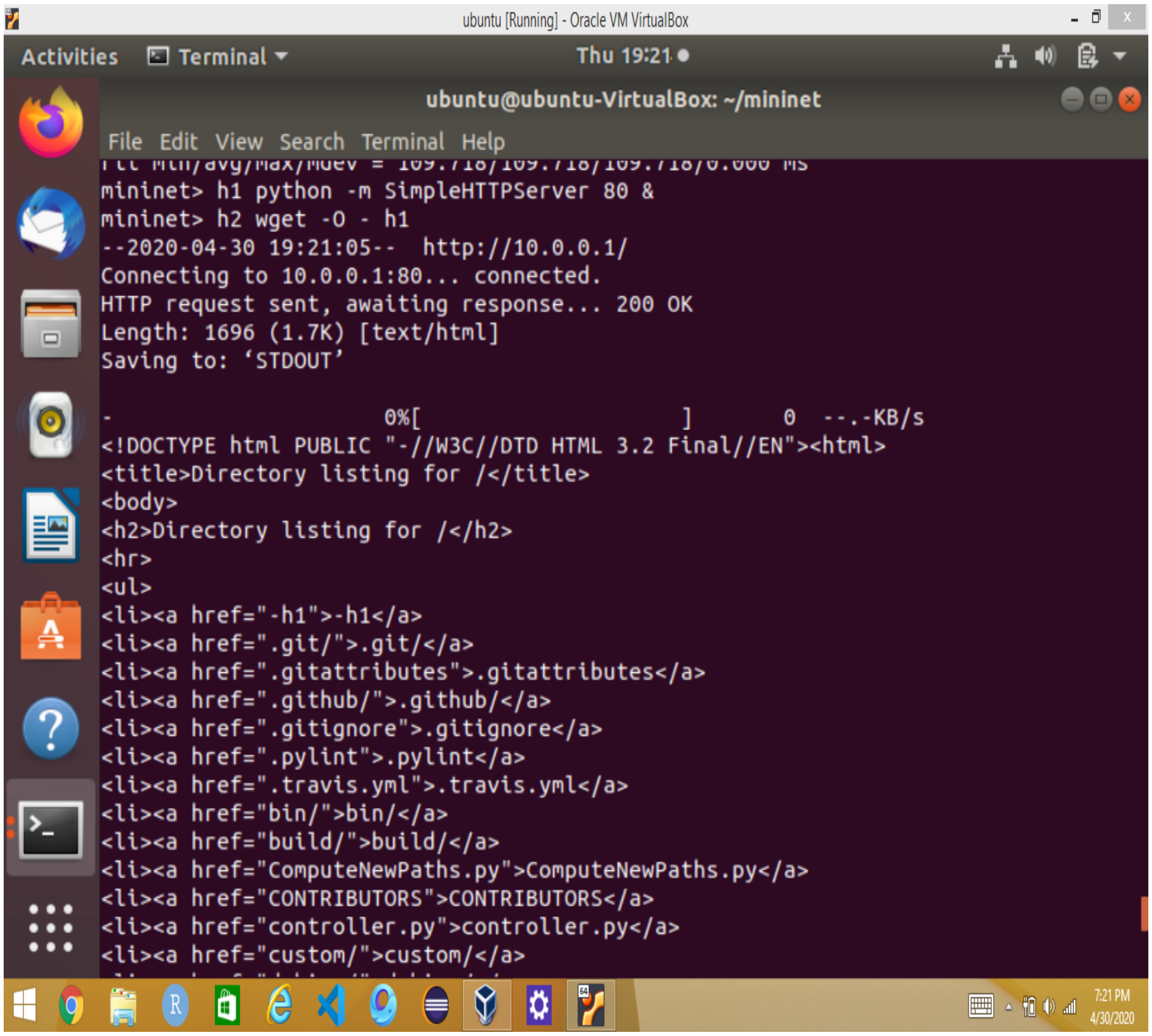
Traffic is generated by starting the http web server

mininet> h1 python -m SimpleHTTPServer 80 &

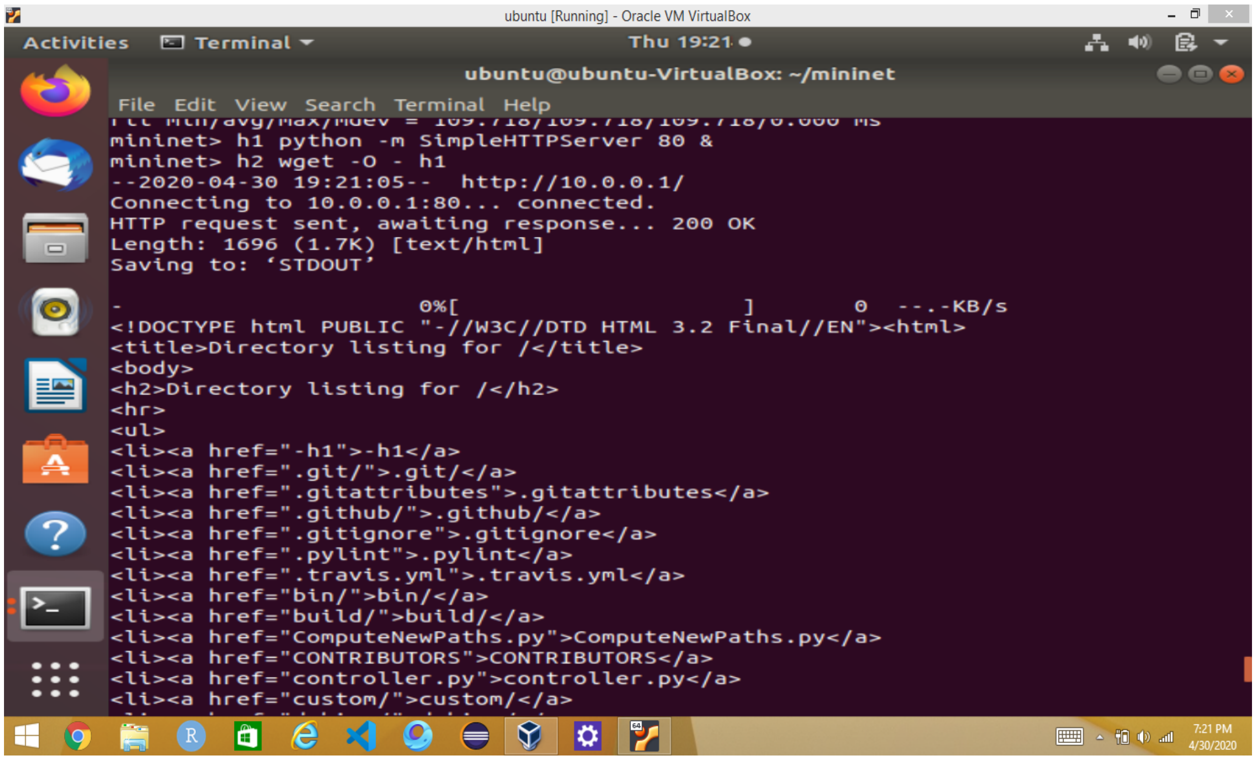
wget to transfer files from host h1 to h2 and h3 to generate some traffic in the network.

mininet> h2 wget -O - h1

mininet> h3 wget -O - h1

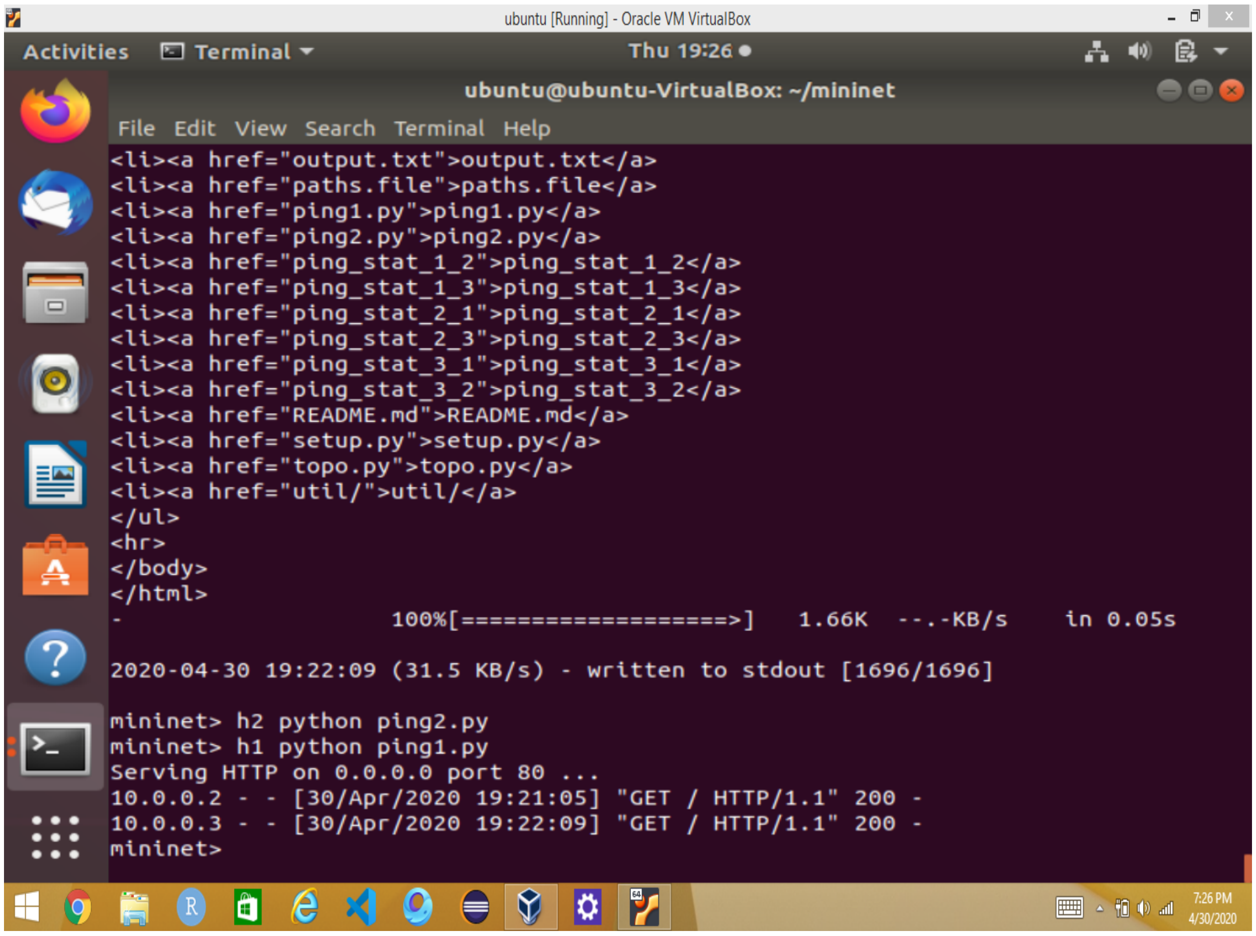

ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:21
ubuntu@ubuntu-VirtualBox: ~/mininet
File Edit View Search Terminal Help
7:21 PM
4/30/2020

```
rtt min/avg/max/mdev = 109.718/109.718/109.718/0.000 ms
mininet> h1 python -m SimpleHTTPServer 80 &
mininet> h2 wget -O - h1
--2020-04-30 19:21:05--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1696 (1.7K) [text/html]
Saving to: ‘STDOUT’

-                     0%[                    ]       0  --.-KB/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="-h1">-h1</a>
<li><a href=".git/">.git/</a>
<li><a href=".gitattributes">.gitattributes</a>
<li><a href=".github/">.github/</a>
<li><a href=".gitignore">.gitignore</a>
<li><a href=".pylint">.pylint</a>
<li><a href=".travis.yml">.travis.yml</a>
<li><a href="bin/">bin/</a>
<li><a href="build/">build/</a>
<li><a href="ComputeNewPaths.py">ComputeNewPaths.py</a>
<li><a href="CONTRIBUTORS">CONTRIBUTORS</a>
<li><a href="controller.py">controller.py</a>
<li><a href="custom/">custom/</a>
```

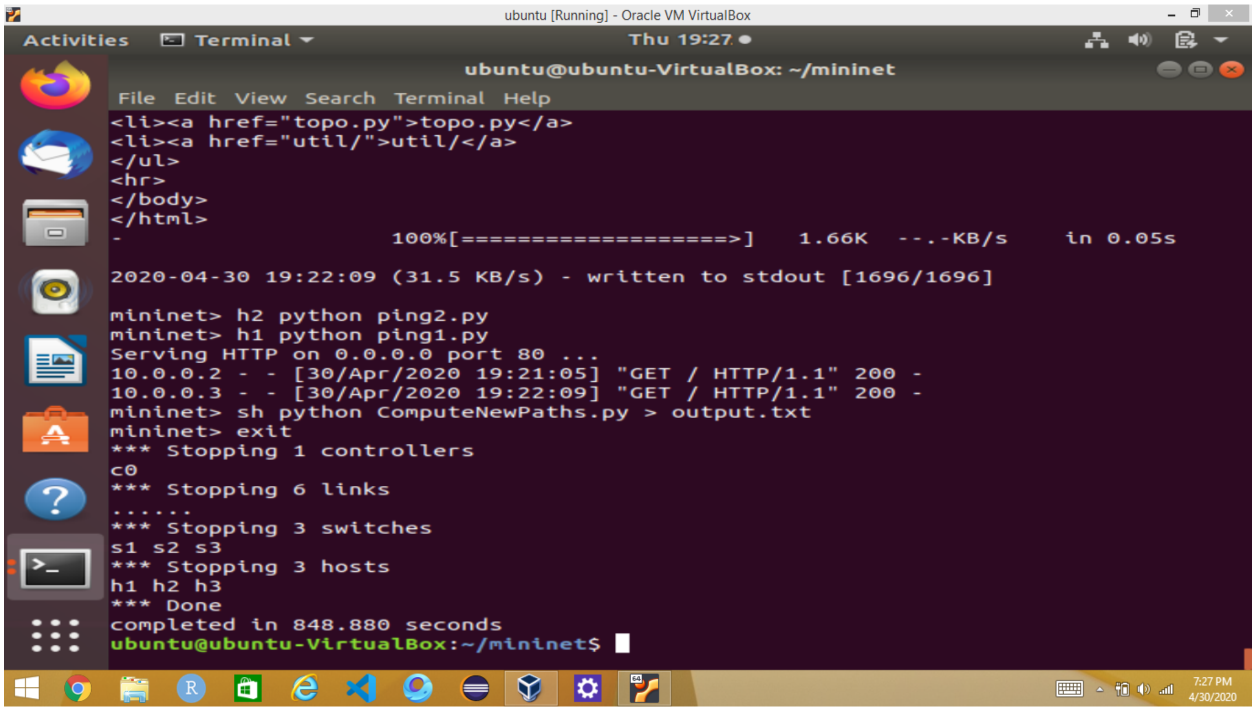


```
mininet> h1 python -m SimpleHTTPServer 80 &
mininet> h2 wget -O - h1
--2020-04-30 19:21:05--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1696 (1.7K) [text/html]
Saving to: 'STDOUT'

-                      0%[                          ]       0  --.-KB/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="-h1">-h1</a>
<li><a href=".git/">.git/</a>
<li><a href=".gitattributes">.gitattributes</a>
<li><a href=".github/">.github/</a>
<li><a href=".gitignore">.gitignore</a>
<li><a href=".pylint">.pylint</a>
<li><a href=".travis.yml">.travis.yml</a>
<li><a href="bin/">bin/</a>
<li><a href="build/">build/</a>
<li><a href="ComputeNewPaths.py">ComputeNewPaths.py</a>
<li><a href="CONTRIBUTORS">CONTRIBUTORS</a>
<li><a href="controller.py">controller.py</a>
<li><a href="custom/">custom/</a>
```

Running Ping Scripts


ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:26
ubuntu@ubuntu-VirtualBox: ~/mininet
File Edit View Search Terminal Help
7:26 PM
4/30/2020

```
<li><a href="output.txt">output.txt</a>
<li><a href="paths.file">paths.file</a>
<li><a href="ping1.py">ping1.py</a>
<li><a href="ping2.py">ping2.py</a>
<li><a href="ping_stat_1_2">ping_stat_1_2</a>
<li><a href="ping_stat_1_3">ping_stat_1_3</a>
<li><a href="ping_stat_2_1">ping_stat_2_1</a>
<li><a href="ping_stat_2_3">ping_stat_2_3</a>
<li><a href="ping_stat_3_1">ping_stat_3_1</a>
<li><a href="ping_stat_3_2">ping_stat_3_2</a>
<li><a href="README.md">README.md</a>
<li><a href="setup.py">setup.py</a>
<li><a href="topo.py">topo.py</a>
<li><a href="util/">util/</a>
</ul>
<hr>
</body>
</html>
-                    100%[===================>]   1.66K  --.-KB/s    in 0.05s

2020-04-30 19:22:09 (31.5 KB/s) - written to stdout [1696/1696]

mininet> h2 python ping2.py
mininet> h1 python ping1.py
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [30/Apr/2020 19:21:05] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [30/Apr/2020 19:22:09] "GET / HTTP/1.1" 200 -
mininet>
```



ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:27
ubuntu@ubuntu-VirtualBox: ~/mininet
File Edit View Search Terminal Help

```
<li><a href="topo.py">topo.py</a>
<li><a href="util/">util/</a>
</ul>
<hr>
</body>
</html>
-                    100%[===================>]   1.66K  --.-KB/s    in 0.05s

2020-04-30 19:22:09 (31.5 KB/s) - written to stdout [1696/1696]

mininet> h2 python ping2.py
mininet> h1 python ping1.py
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [30/Apr/2020 19:21:05] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [30/Apr/2020 19:22:09] "GET / HTTP/1.1" 200 -
mininet> sh python ComputeNewPaths.py > output.txt
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 6 links
......
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 848.880 seconds
ubuntu@ubuntu-VirtualBox:~/mininet$
```

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

mininet> h2 python ping2.py

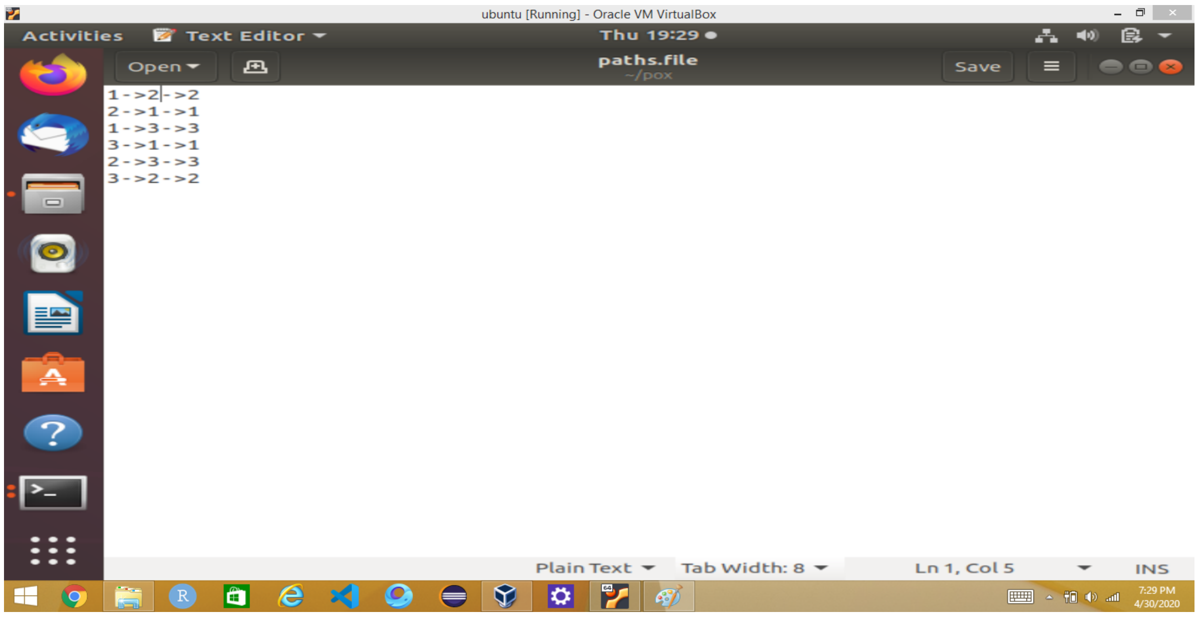
mininet> h1 python ping1.py

mininet> sh python ComputeNewPaths.py > output.txt

Note:

ComputeNewPaths.py - This file contains the logic that will generate alternative paths if required in case of any congestion. The final routes are stored in output.txt file.

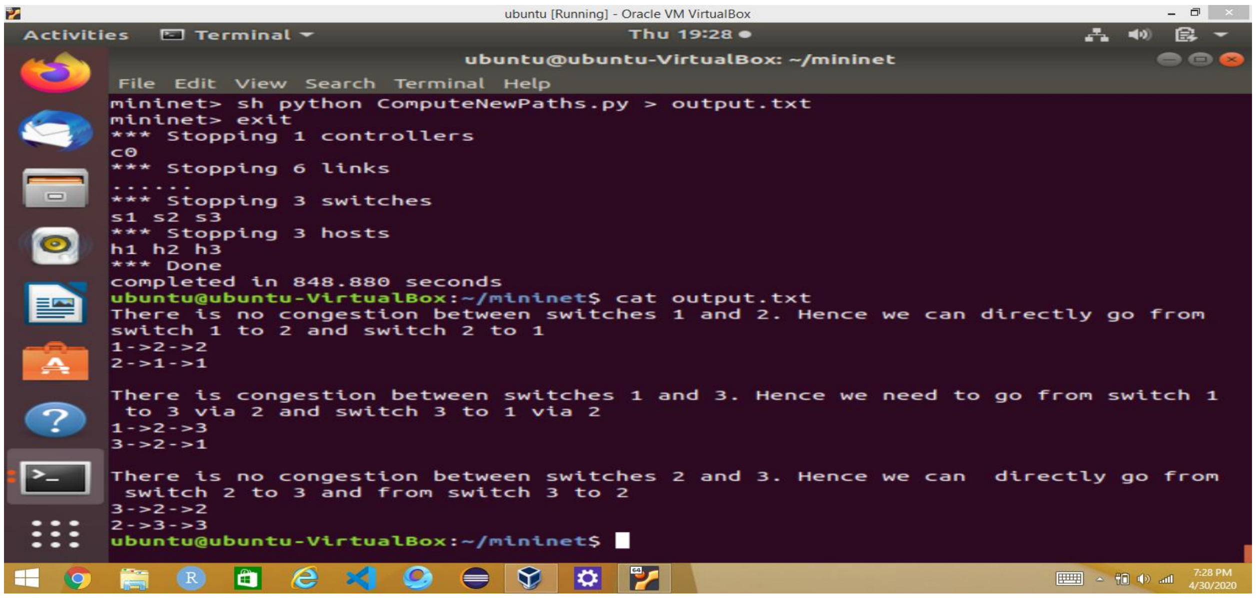
The file paths.file contains the initial routes (best ones before traffic is generated) as shown below:



```
1->2->2
2->1->1
1->3->3
3->1->1
2->3->3
3->2->2
```

The terminology used for paths is described as follow:

1 -> 2 ->3 : Source is 1, destination is 3 and it is via 2. Since, initially there is direct path between source and destination so the via part in path will be same as destination. e.g. 1 -> 2 -> 2. This means that data will go directly from 1 to 2.



```
mininet> sh python ComputeNewPaths.py > output.txt
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 6 links
......
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 848.880 seconds
ubuntu@ubuntu-VirtualBox:~/mininet$ cat output.txt
There is no congestion between switches 1 and 2. Hence we can directly go from
switch 1 to 2 and switch 2 to 1
1->2->2
2->1->1

There is congestion between switches 1 and 3. Hence we need to go from switch 1
 to 3 via 2 and switch 3 to 1 via 2
1->2->3
3->2->1

There is no congestion between switches 2 and 3. Hence we can  directly go from
 switch 2 to 3 and from switch 3 to 2
3->2->2
2->3->3
ubuntu@ubuntu-VirtualBox:~/mininet$
```

Based on the statistics, the controller modifies the flow to avoid congestion. Here in this example, path from 1 to 3 has more congestion. Hence packets will be directed to other routes to avoid congestion and that is the reason the packet will be routed as 1->2->3 and 3->2->1.

STATISTICS:

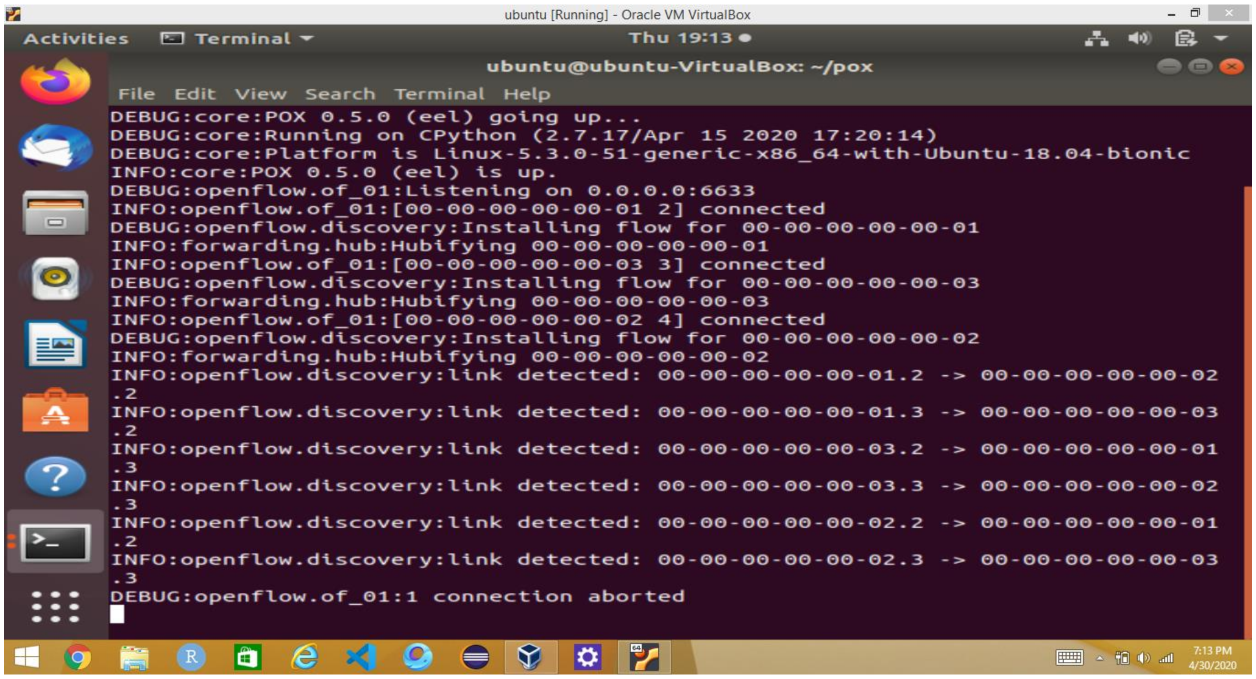
We ran set of python scripts to get statistics from switches which will ping all the nodes from a particular node. After this ping operation we collected latency (ping value) between each node. Based on ping values, we found an alternative path with lowest ping value and used that path for packet transmission/ forwarding. These statistical data helped controller to decode alternative path.

CHALLENGES ENCOUNTERED DURING PROJECT:

A huge traffic was generated when ARP request is broadcasted. This issue led to problem of congestion of packets and it took more time to get results from ping command. This problem can be solved by hardcoding the MAC address whenever ARP request is received and then it is sent to requested host. Handle_packetin method has a solution to handle/ deal with it.

Flow control statistics can be given by a method in POX controller, and statistics can be collected by adding listener named “flow status received”. But it doesn’t provide enough information to handle flow statistics. So, to get latency in network we wrote some scripts.

We also tried to extend number of hosts and controller, but we faced difficulties during calculation of ping time also we waited for long time for HTTP traffic generation, but something went wrong, and we couldn’t get any traffic/ results (or may be, we didn’t wait for sufficient time to generate traffic) as our main aim was to design an system that will work correctly in case of congestion and give appropriate results.


ubuntu [Running] - Oracle VM VirtualBox
Activities
Terminal
Thu 19:13
ubuntu@ubuntu-VirtualBox: ~/pox
File Edit View Search Terminal Help
7:13 PM
4/30/2020

```
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Apr 15 2020 17:20:14)
DEBUG:core:Platform is Linux-5.3.0-51-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
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:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-03
INFO:forwarding.hub:Hubifying 00-00-00-00-00-03
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:openflow.discovery:Installing flow for 00-00-00-00-00-02
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02
.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -> 00-00-00-00-00-03
.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-01
.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.3 -> 00-00-00-00-00-02
.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01
.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03
.3
DEBUG:openflow.of_01:1 connection aborted
```

CONCLUSION:

We designed a simple SDN Controller and implemented additional functionality on top of it. We also used mininet for network simulation. We faced various obstacles in the integration for example as mentioned in challenges we tried to use “Open Daylight controller” but we faced lot of issues and found various workarounds to get the expected working of network. We also tried to extend topology to 4 switches and 4 hosts,

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