# CS 4001/7001 Cloud Computing
**Spring 2015**
**Lab # 3: QoS Configuration and Load Balancing using Software-Defined Networking**

*Dr. Prasad Calyam (Contact: calyamp@missouri.edu)*

## 1. Purpose of the Lab

Install and configure Mininet SDN emulator with 2 traffic engineering experiment applications to understand how to program 'flow spaces' within networks to: (i) comply with enterprise network capacity provisioning policies, and (ii) balance the utilization of network resources.

## 2. References to guide lab work

[1] OpenFlow Tutorial: http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial
[2] Mininet Walkthrough: http://mininet.org/walkthrough/
[3] OpenvSwitch Virtual Switch: http://openvswitch.org/
[4] Floodlight Controller Project: http://www.projectfloodlight.org/floodlight/
[5] SDN Controller Application for QoS Control: http://www.openflowhub.org/display/floodlightcontroller/How+to+implement+Quality+Of+Service+using+Floodlight
[6] SDN Controller Application for Load Balancer: http://www.openflowhub.org/display/floodlightcontroller/Load+Balancer

## 3. Lab Steps and output collection guidelines

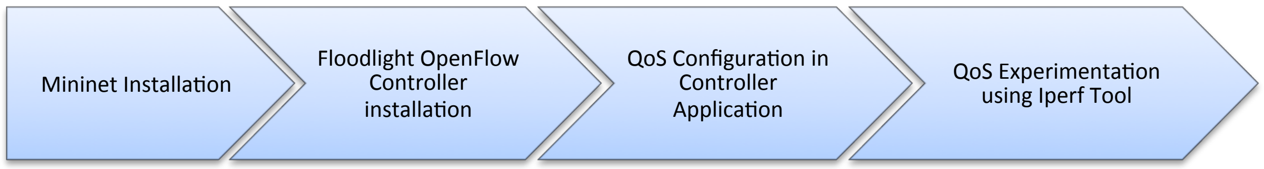

Mininet Installation
Floodlight OpenFlow Controller installation
QoS Configuration in Controller Application
QoS Experimentation using Iperf Tool

**Figure 1: Lab Experiment #1 (QoS Control) Steps Overview**

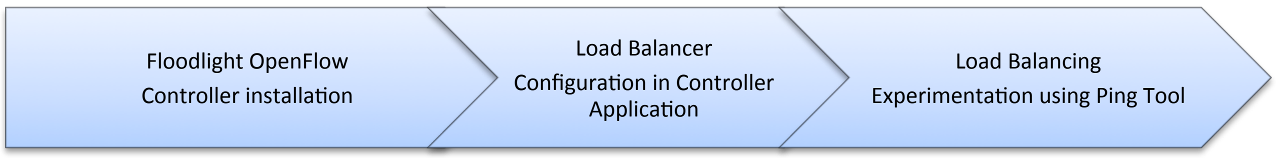

Floodlight OpenFlow Controller installation
Load Balancer Configuration in Controller Application
Load Balancing Experimentation using Ping Tool

**Figure 2: Lab Experiment #2 (Load Balancer) Steps Overview**

Figure 1 shows the required steps to successfully configure Mininet [2] with virtual switches [3] and use the first SDN controller application that involves QoS configurations to rate-limit the capacity on network-edge links connecting client/server nodes. You will use a Floodlight Controller [4] module [5] to configure your experiment and verify bandwidth capacity provisioning at network-edges using the Iperf Tool.

Figure 2 shows the required steps to successfully use the second SDN controller application that load balances network traffic between client/server nodes to show scalable handling of traffic flows. You will use another Floodlight Controller module [6] to configure your experiment and verify balanced usage of network resources using the Ping Tool.

## 3.1 Mininet Installation

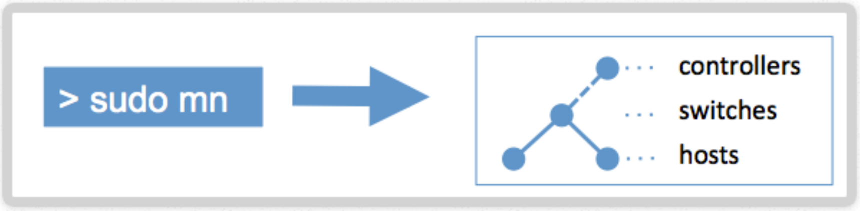


**Figure 3: Mininet Overview [2]**

You should use the Linux VM for the following instructions to install Mininet.

*Note :* Mac users can download VirtualBox, install the Linux image provided in GENI Lab 1. Follow the '**VirtualBox Installation instructions**' file uploaded in Blackboard. Post installation, you can install Mininet with the following instructions.

$sudo su

$sudo apt-get install git

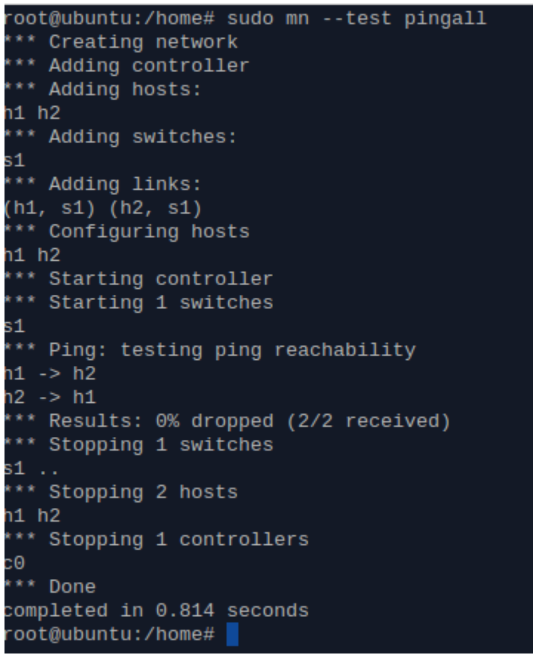
$git clone git://github.com/mininet/mininet.git

$cd mininet/util

$./install.sh –a

Verify Mininet installation:

$sudo mn --test pingall



```
root@ubuntu:/home# sudo mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 switches
s1 ..
*** Stopping 2 hosts
h1 h2
*** Stopping 1 controllers
c0
*** Done
completed in 0.814 seconds
root@ubuntu:/home#
```

## 3.2 Floodlight OpenFlow Controller Installation

There are several flavors of OpenFlow controllers available for various environments. For this lab, we will be using the Java based Floodlight controller (whose architecture is shown through Figures 4 and 5) to configure and manage our network topology of OpenFlow switches and end-hosts.

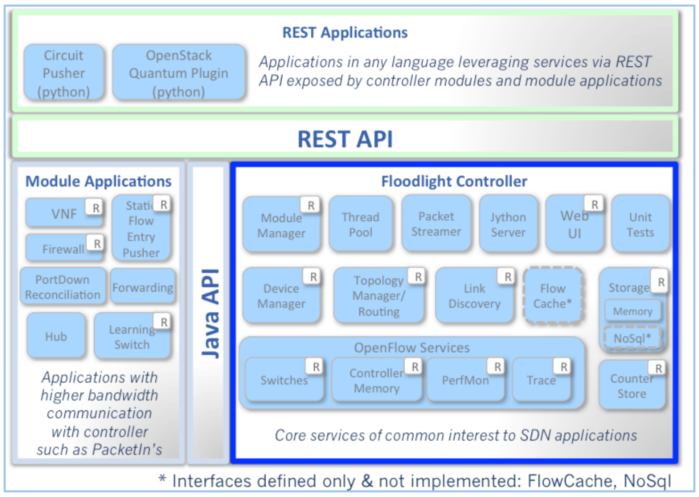


**Figure 4: Floodlight Controller Architecture [3]**

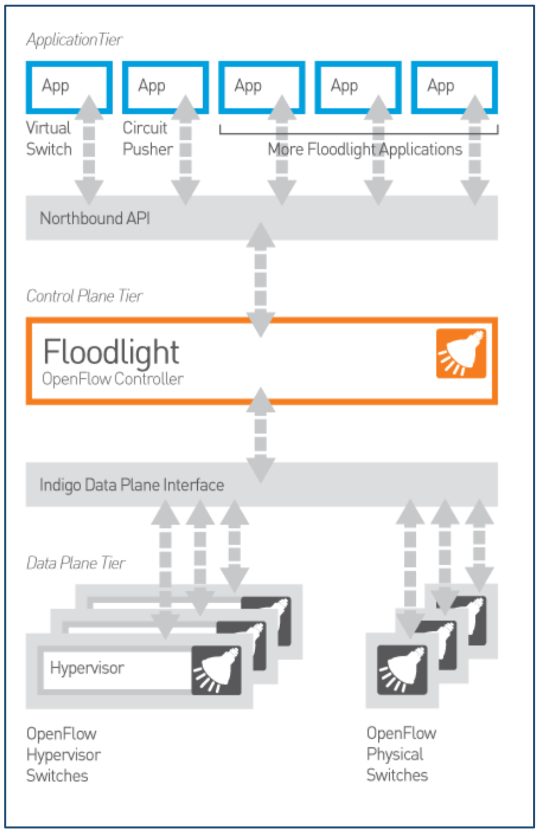


**Figure 5: Multiple applications can be built on top of Floodlight controller to address various networking requirements and policies [3]**

In your home directory, install dependencies such as Apache ant, and JDK (if not already installed) using the below commands:

$apt-get install ant

$apt-get install default-jdk

$apt-get install python-simplejson

$apt-get install curl

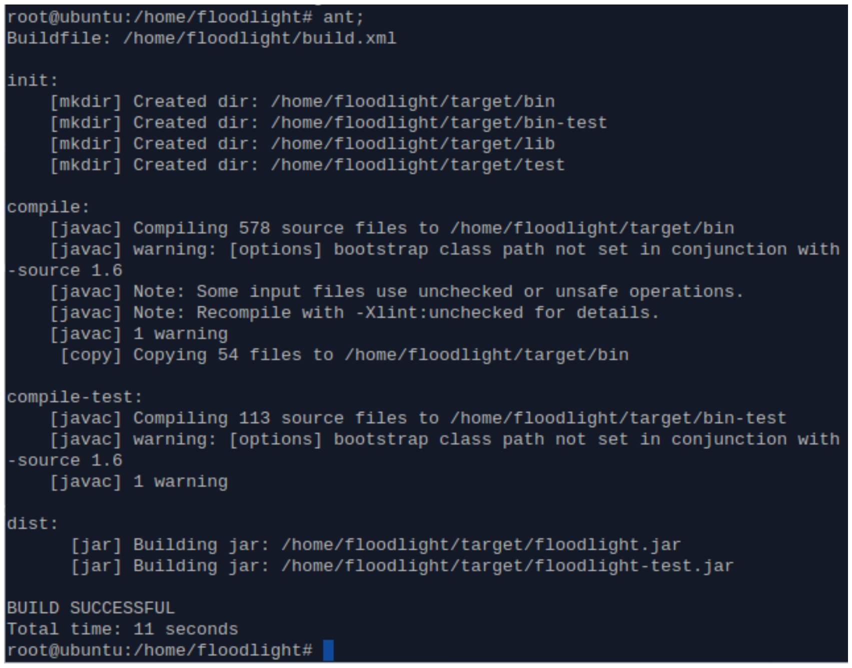
Install Floodlight controller with QoS module:

$git clone https://github.com/wallnerryan/floodlight-qos-beta.git

$cd floodlight-qos-beta

Build the controller by running the ant command:

$ant;

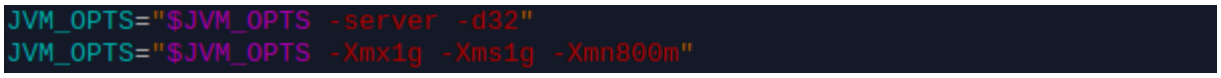


```
root@ubuntu:/home/floodlight# ant;
Buildfile: /home/floodlight/build.xml

init:
    [mkdir] Created dir: /home/floodlight/target/bin
    [mkdir] Created dir: /home/floodlight/target/bin-test
    [mkdir] Created dir: /home/floodlight/target/lib
    [mkdir] Created dir: /home/floodlight/target/test

compile:
    [javac] Compiling 578 source files to /home/floodlight/target/bin
    [javac] warning: [options] bootstrap class path not set in conjunction with
-source 1.6
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
    [javac] 1 warning
     [copy] Copying 54 files to /home/floodlight/target/bin

compile-test:
    [javac] Compiling 113 source files to /home/floodlight/target/bin-test
    [javac] warning: [options] bootstrap class path not set in conjunction with
-source 1.6
    [javac] 1 warning

dist:
      [jar] Building jar: /home/floodlight/target/floodlight.jar
      [jar] Building jar: /home/floodlight/target/floodlight-test.jar

BUILD SUCCESSFUL
Total time: 11 seconds
root@ubuntu:/home/floodlight#
```

Edit the floodlight.sh file and change the server parameter from d64 to d32 as shown below:

$vi floodlight.sh

Change -d64 to -d32

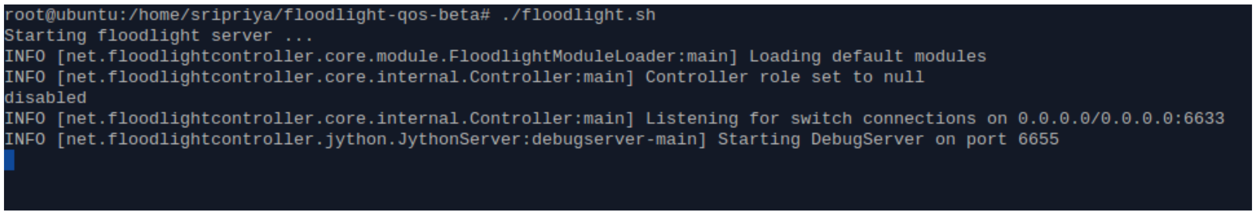


```
JVM_OPTS="$JVM_OPTS -server -d32"
JVM_OPTS="$JVM_OPTS -Xmx1g -Xms1g -Xmn800m"
```

Start the Floodlight controller:

$./floodlight.sh

You should see your controller running as shown below:



```
root@ubuntu:/home/sripriya/floodlight-qos-beta# ./floodlight.sh
Starting floodlight server ...
INFO [net.floodlightcontroller.core.module.FloodlightModuleLoader:main] Loading default modules
INFO [net.floodlightcontroller.core.internal.Controller:main] Controller role set to null
disabled
INFO [net.floodlightcontroller.core.internal.Controller:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
INFO [net.floodlightcontroller.jython.JythonServer:debugserver-main] Starting DebugServer on port 6655
```

You can access the Floodlight controller UI using the below URL:

http://localhost:8080/ui/index.html

### 3.3 QoS Configuration in Controller Application

#### 3.3.1 Network Topology Creation

Copy the provided 'topo-6sw-4host.py' file from Blackboard to <yourpath>/mininet/custom/ directory. You may require admin privileges to copy. This file will use the OpenvSwitch (i.e., software implementation of OpenFlow switch) to setup your OpenFlow switch topology as shown in Figure 6.

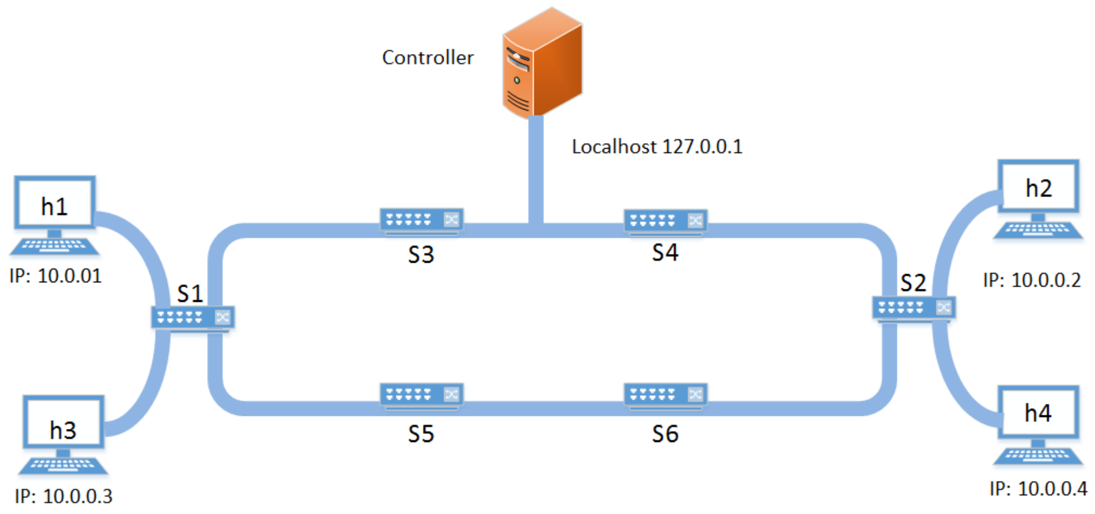
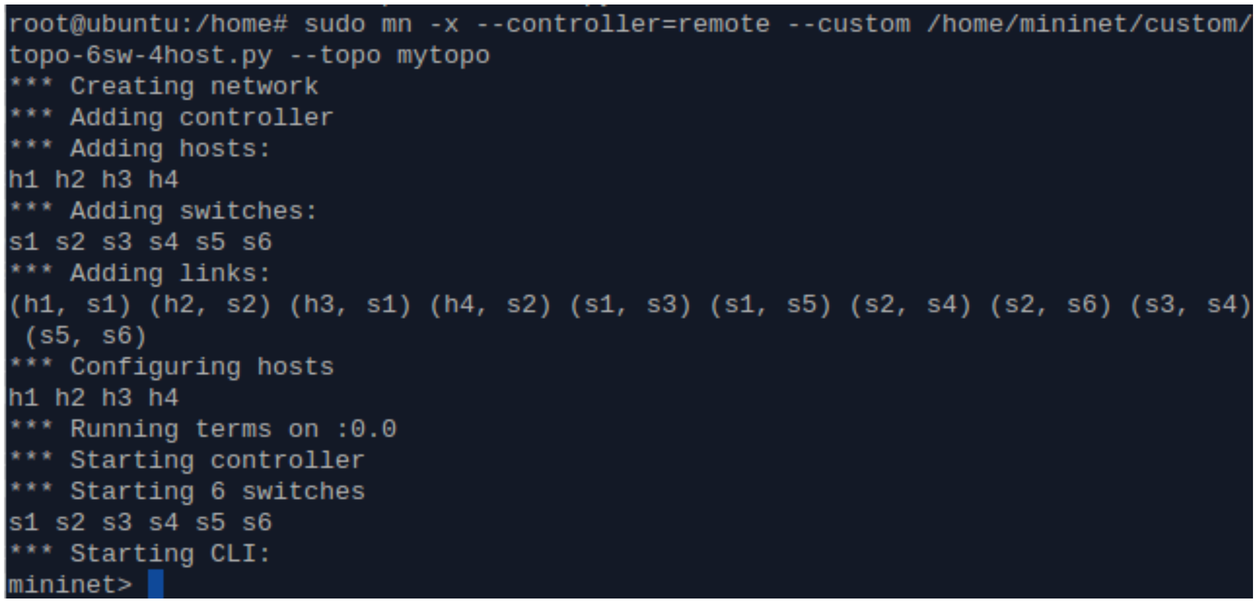


**Figure 6: OpenFlow Switch Topology for QoS Experimentation (6 switches, 4 hosts and 1 controller)**

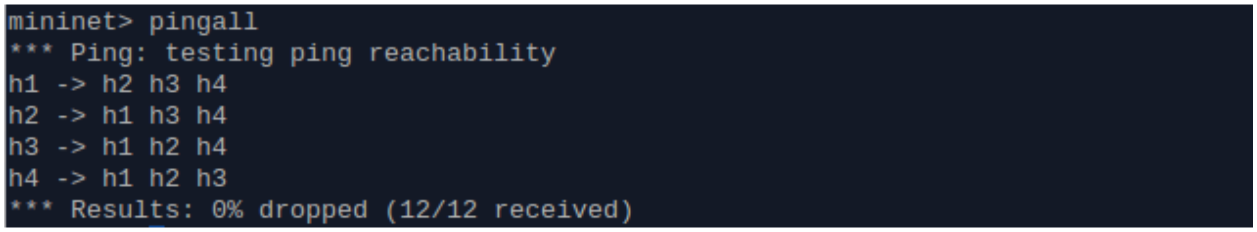
Reserve the topology using the below command:

$sudo mn -x --controller=remote --custom <yourpath>mininet/custom/topo-6sw-4host.py --topo mytopo

This will also bring up terminals for the 6 switches, 4 hosts and 1 controller


```
root@ubuntu:/home# sudo mn -x --controller=remote --custom /home/mininet/custom/
topo-6sw-4host.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s2) (h3, s1) (h4, s2) (s1, s3) (s1, s5) (s2, s4) (s2, s6) (s3, s4)
 (s5, s6)
*** Configuring hosts
h1 h2 h3 h4
*** Running terms on :0.0
*** Starting controller
*** Starting 6 switches
s1 s2 s3 s4 s5 s6
*** Starting CLI:
mininet>
```

Verify connectivity among the hosts by giving the command “pingall” on mininet CLI


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Open your bowser and visit http://localhost:8080/ui/index.html, navigate through the tabs. You should see the 6 switches and 4 hosts on your UI.

### 3.3.2 QoS Configuration

The configuration and setup files required to run the QoS module will be in your home directory in the path:

floodlight-qos-beta/apps/qos/

Using the OpenvSwitch commands to set the network policies, we will setup 3 queues (Q0, Q1 and Q2) on every switch and configure at network-edge bandwidth capacity using the ‘ovs-vsctl’ command.

Q0 – default queue

Q1 – queue 1 rate limiting bandwidth to 50 Mbps

Q2 – queue 2 rate limiting bandwidth to 40 Mbps

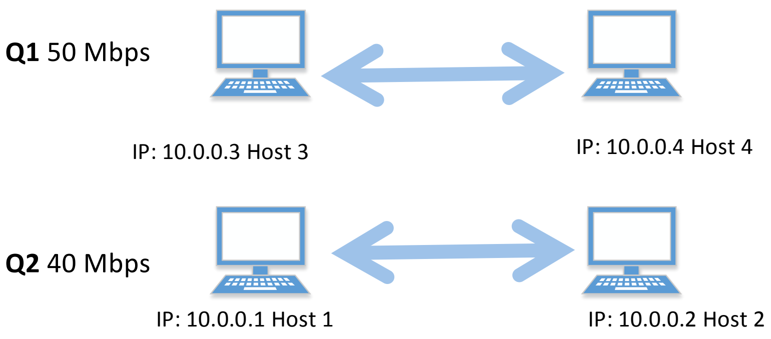
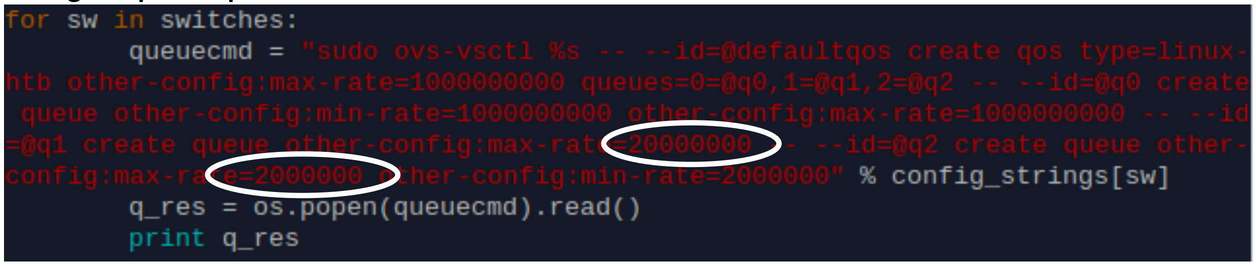


**Figure 7: Illustration of host connections with different queues and rate limits**

Open the mininet-add-queues.py file using the command:

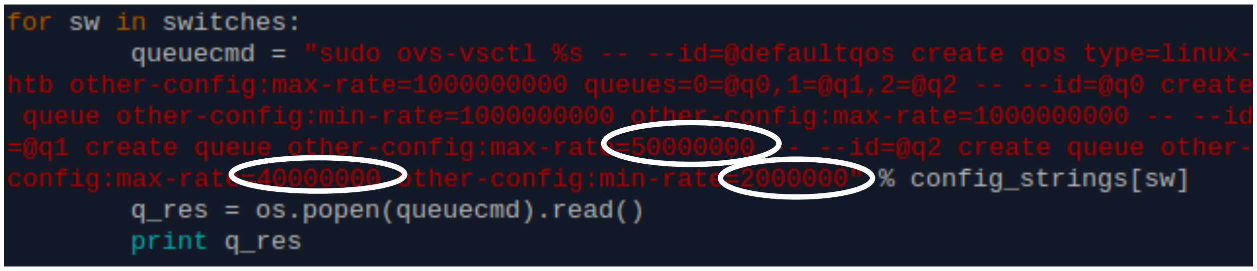
$vim /home/floodlight-qos-beta/apps/qos/mininet-add-queues.py

**Configure q1 and q2 max rate From:**



```
for sw in switches:
        queuecmd = "sudo ovs-vsctl %s -- --id=@defaultqos create qos type=linux-
htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create
 queue other-config:min-rate=1000000000 other-config:max-rate=1000000000 -- --id
=@q1 create queue other-config:max-rate=20000000 -- --id=@q2 create queue other-
config:max-rate=2000000 other-config:min-rate=2000000" % config_strings[sw]
        q_res = os.popen(queuecmd).read()
        print q_res
```

**To:**

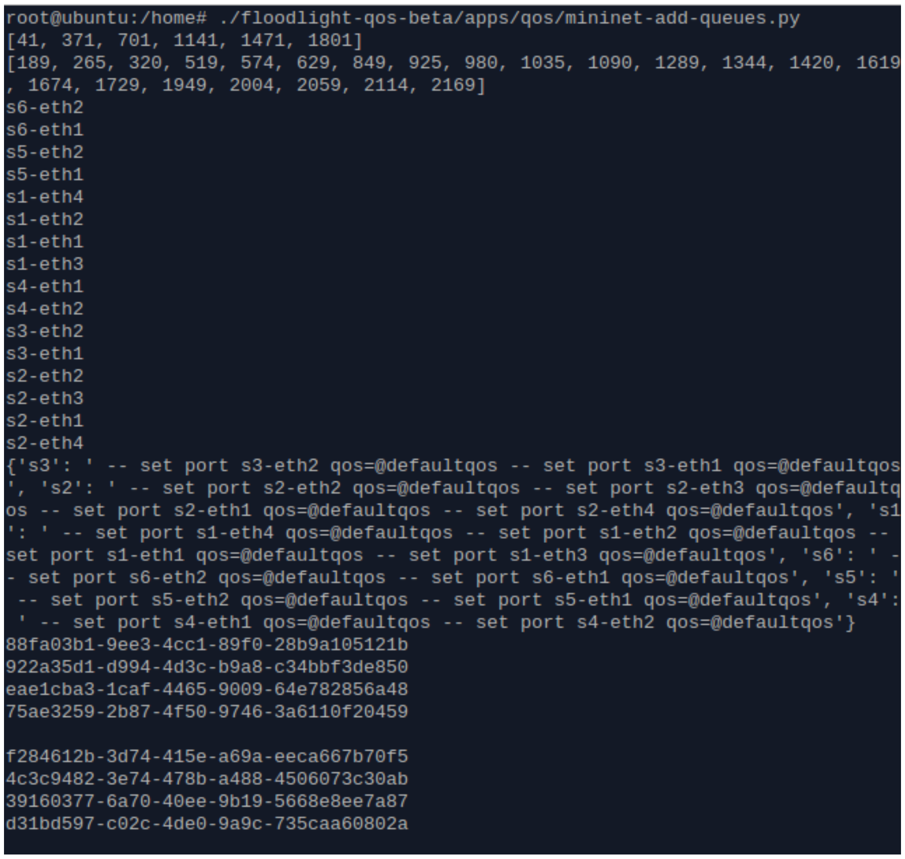


```
for sw in switches:
        queuecmd = "sudo ovs-vsctl %s -- --id=@defaultqos create qos type=linux-
htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create
 queue other-config:min-rate=1000000000 other-config:max-rate=1000000000 -- --id
=@q1 create queue other-config:max-rate=50000000 -- --id=@q2 create queue other-
config:max-rate=40000000 other-config:min-rate=2000000" % config_strings[sw]
        q_res = os.popen(queuecmd).read()
        print q_res
```

Now run the script to add the configured queues of the switches for the OpenFlow switch topology as per Figure 7.

$./floodlight-qos-beta/apps/qos/mininet-add-queues.py

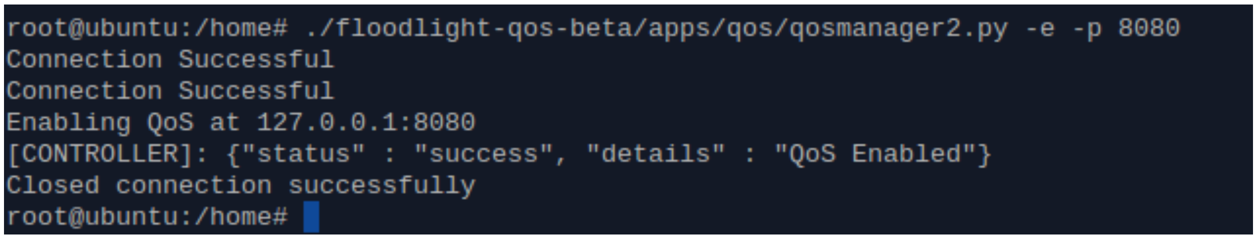
The output should look as shown below.



```
root@ubuntu:/home# ./floodlight-qos-beta/apps/qos/mininet-add-queues.py
[41, 371, 701, 1141, 1471, 1801]
[189, 265, 320, 519, 574, 629, 849, 925, 980, 1035, 1090, 1289, 1344, 1420, 1619
, 1674, 1729, 1949, 2004, 2059, 2114, 2169]
s6-eth2
s6-eth1
s5-eth2
s5-eth1
s1-eth4
s1-eth2
s1-eth1
s1-eth3
s4-eth1
s4-eth2
s3-eth2
s3-eth1
s2-eth2
s2-eth3
s2-eth1
s2-eth4
{'s3': ' -- set port s3-eth2 qos=@defaultqos -- set port s3-eth1 qos=@defaultqos
', 's2': ' -- set port s2-eth2 qos=@defaultqos -- set port s2-eth3 qos=@defaultq
os -- set port s2-eth1 qos=@defaultqos -- set port s2-eth4 qos=@defaultqos', 's1
': ' -- set port s1-eth4 qos=@defaultqos -- set port s1-eth2 qos=@defaultqos --
set port s1-eth1 qos=@defaultqos -- set port s1-eth3 qos=@defaultqos', 's6': ' -
- set port s6-eth2 qos=@defaultqos -- set port s6-eth1 qos=@defaultqos', 's5': '
 -- set port s5-eth2 qos=@defaultqos -- set port s5-eth1 qos=@defaultqos', 's4':
 ' -- set port s4-eth1 qos=@defaultqos -- set port s4-eth2 qos=@defaultqos'}
88fa03b1-9ee3-4cc1-89f0-28b9a105121b
922a35d1-d994-4d3c-b9a8-c34bbf3de850
eae1cba3-1caf-4465-9009-64e782856a48
75ae3259-2b87-4f50-9746-3a6110f20459

f284612b-3d74-415e-a69a-eeca667b70f5
4c3c9482-3e74-478b-a488-4506073c30ab
39160377-6a70-40ee-9b19-5668e8ee7a87
d31bd597-c02c-4de0-9a9c-735caa60802a
```

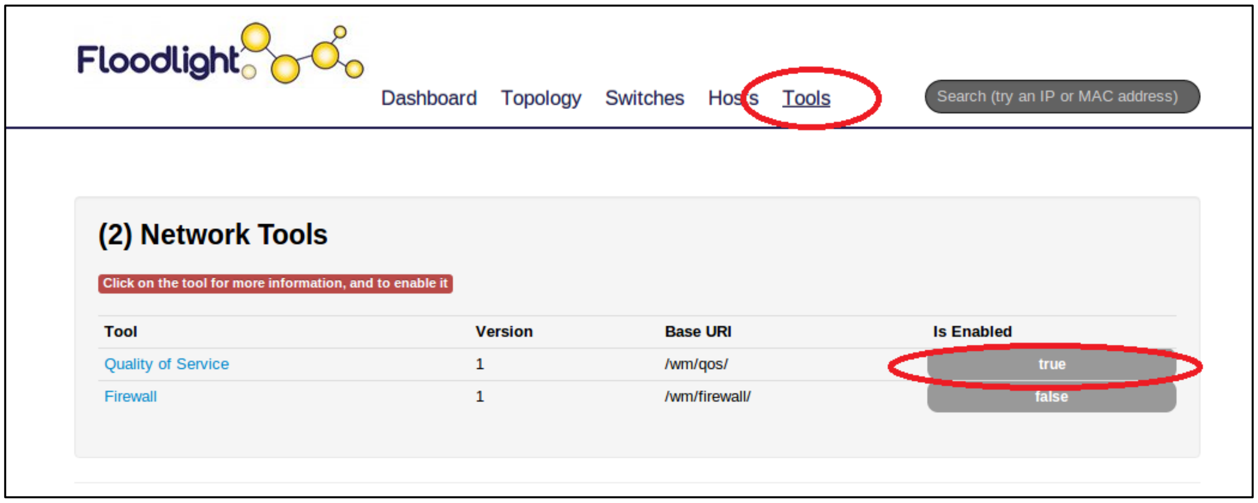
Let us enable the QoS module using the command:

# ./floodlight-qos-beta/apps/qos/qosmanager2.py -e -p 8080



```
root@ubuntu:/home# ./floodlight-qos-beta/apps/qos/qosmanager2.py -e -p 8080
Connection Successful
Connection Successful
Enabling QoS at 127.0.0.1:8080
[CONTROLLER]: {"status" : "success", "details" : "QoS Enabled"}
Closed connection successfully
root@ubuntu:/home#
```

You can also verify that the QoS module is enabled by navigating to the Tools tab in the Floodlight UI as shown in Figure 8; the QoS "Is Enabled" property should show "true" setting.



(2) Network Tools

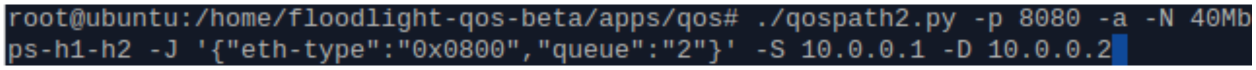
Click on the tool for more information, and to enable it

| Tool | Version | Base URI | Is Enabled |
|---|---|---|---|
| Quality of Service | 1 | /wm/qos/ | true |
| Firewall | 1 | /wm/firewall/ | false |

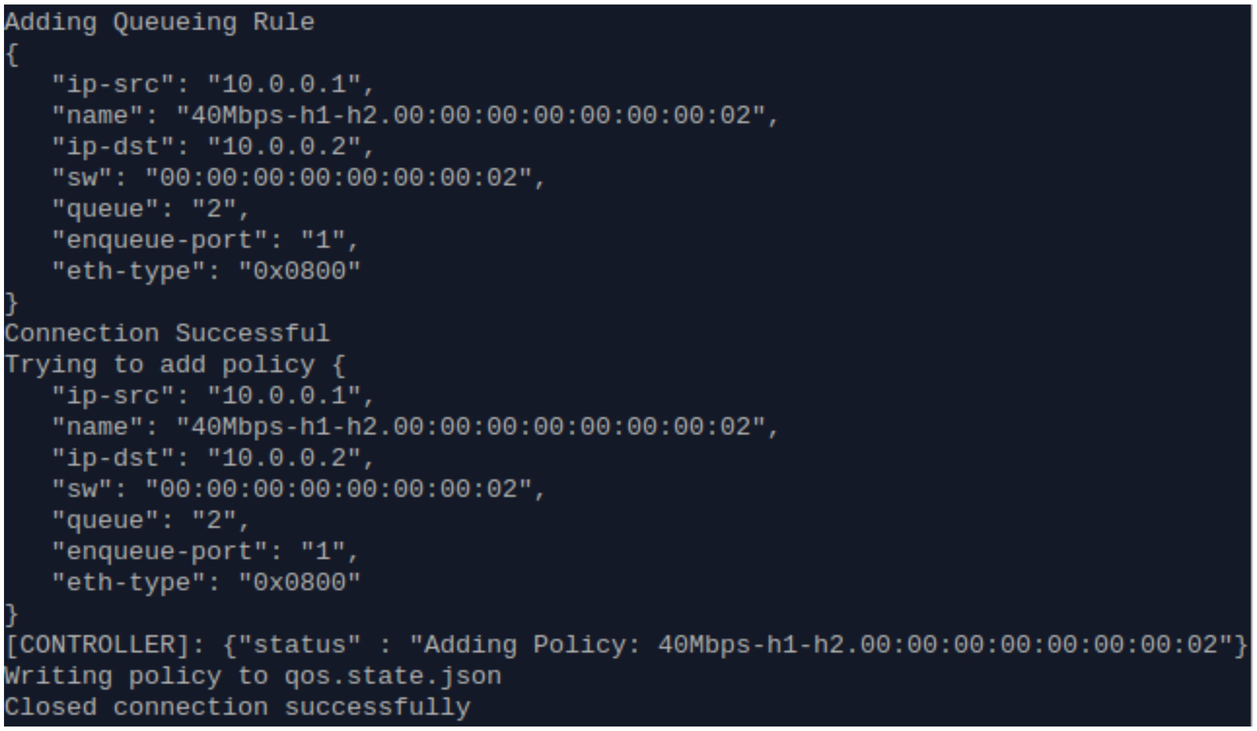
**Figure 8: Floodlight Network Tools Tab**

Now establish a flow from host1 to host2 with a bandwidth of 40Mbps using queue 2 on the switch using the command:

$./qospath2.py -p 8080 -a -N 40Mbps-h1-h2 -J '{"eth-type":"0x0800","queue":"2"}' -S 10.0.0.1 -D 10.0.0.2



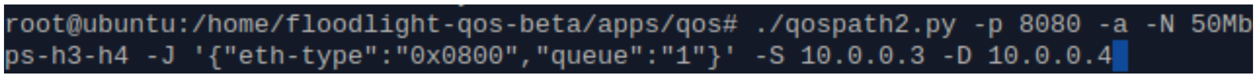
```
root@ubuntu:/home/floodlight-qos-beta/apps/qos# ./qospath2.py -p 8080 -a -N 40Mb
ps-h1-h2 -J '{"eth-type":"0x0800","queue":"2"}' -S 10.0.0.1 -D 10.0.0.2
```



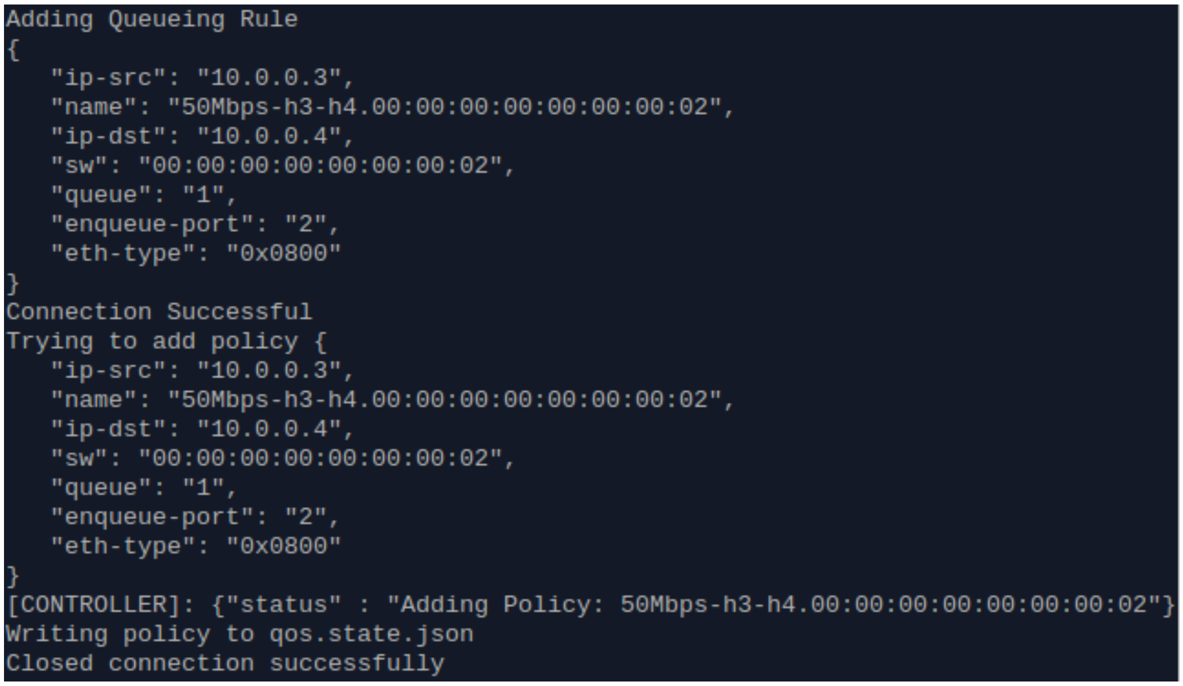
```
Adding Queueing Rule
{
   "ip-src": "10.0.0.1",
   "name": "40Mbps-h1-h2.00:00:00:00:00:00:00:02",
   "ip-dst": "10.0.0.2",
   "sw": "00:00:00:00:00:00:00:02",
   "queue": "2",
   "enqueue-port": "1",
   "eth-type": "0x0800"
}
Connection Successful
Trying to add policy {
   "ip-src": "10.0.0.1",
   "name": "40Mbps-h1-h2.00:00:00:00:00:00:00:02",
   "ip-dst": "10.0.0.2",
   "sw": "00:00:00:00:00:00:00:02",
   "queue": "2",
   "enqueue-port": "1",
   "eth-type": "0x0800"
}
[CONTROLLER]: {"status" : "Adding Policy: 40Mbps-h1-h2.00:00:00:00:00:00:00:02"}
Writing policy to qos.state.json
Closed connection successfully
```

Establish a flow rule from host3 to host4 with 50Mbps bandwidth using queue 1

$./qospath2.py -p 8080 -a -N 50Mbps-h3-h4 -J '{"eth-type":"0x0800","queue":"1"}' -S 10.0.0.3 -D 10.0.0.4



```
root@ubuntu:/home/floodlight-qos-beta/apps/qos# ./qospath2.py -p 8080 -a -N 50Mb
ps-h3-h4 -J '{"eth-type":"0x0800","queue":"1"}' -S 10.0.0.3 -D 10.0.0.4
```

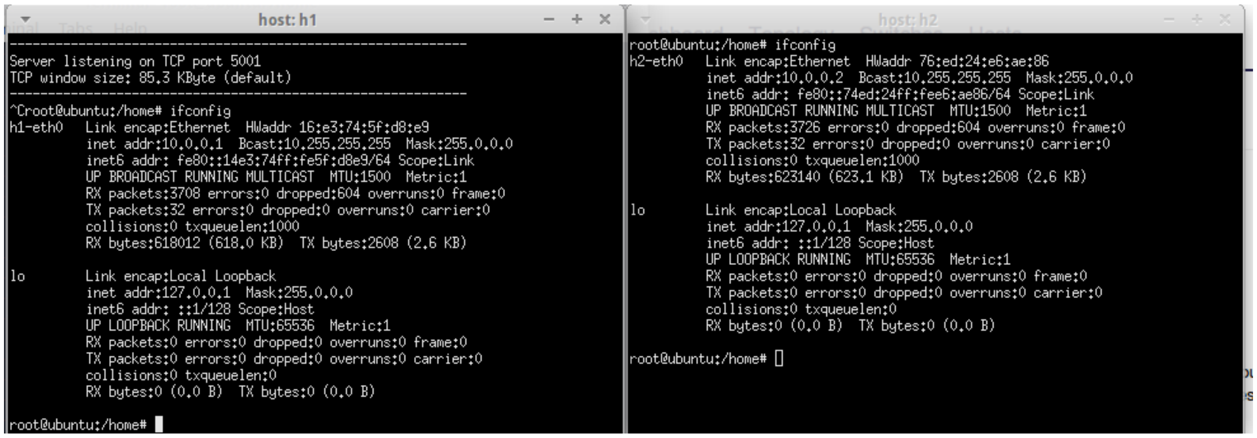


```
Adding Queueing Rule
{
   "ip-src": "10.0.0.3",
   "name": "50Mbps-h3-h4.00:00:00:00:00:00:00:02",
   "ip-dst": "10.0.0.4",
   "sw": "00:00:00:00:00:00:00:02",
   "queue": "1",
   "enqueue-port": "2",
   "eth-type": "0x0800"
}
Connection Successful
Trying to add policy {
   "ip-src": "10.0.0.3",
   "name": "50Mbps-h3-h4.00:00:00:00:00:00:00:02",
   "ip-dst": "10.0.0.4",
   "sw": "00:00:00:00:00:00:00:02",
   "queue": "1",
   "enqueue-port": "2",
   "eth-type": "0x0800"
}
[CONTROLLER]: {"status" : "Adding Policy: 50Mbps-h3-h4.00:00:00:00:00:00:00:02"}
Writing policy to qos.state.json
Closed connection successfully
```

## 3.4 QoS Experimentation using Iperf Tool

Verify IP address of your 4 hosts using xterm terminals.

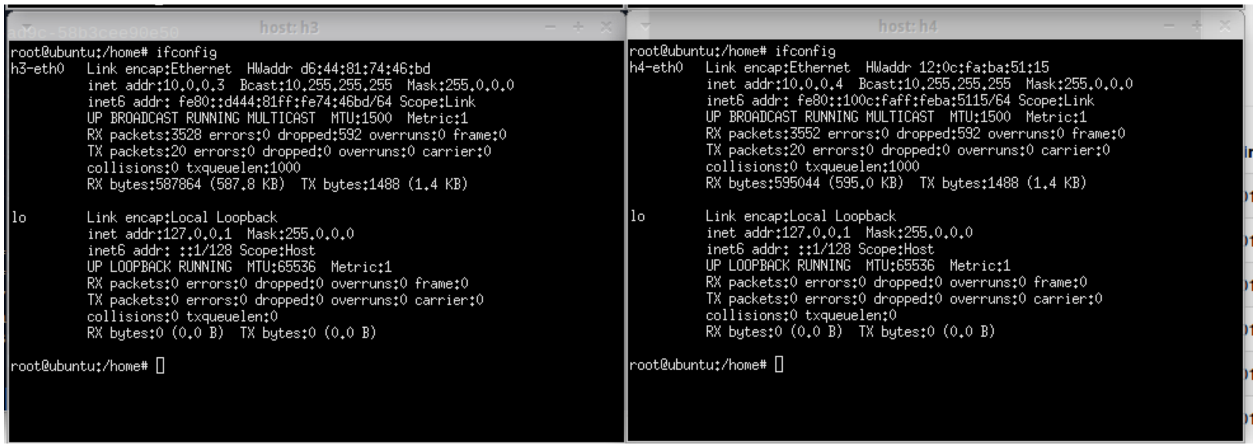
Output for host h1 and host h2 using ifconfig



```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
^Croot@ubuntu:/home# ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 16:e3:74:5f:d8:e9
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::14e3:74ff:fe5f:d8e9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3708 errors:0 dropped:604 overruns:0 frame:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:618012 (618.0 KB)  TX bytes:2608 (2.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home#
```

```
root@ubuntu:/home# ifconfig
h2-eth0   Link encap:Ethernet  HWaddr 76:ed:24:e6:ae:86
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::74ed:24ff:fee6:ae86/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3726 errors:0 dropped:604 overruns:0 frame:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:623140 (623.1 KB)  TX bytes:2608 (2.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home#
```

Output for host h3 and host h4 using ifconfig



```
root@ubuntu:/home# ifconfig
h3-eth0   Link encap:Ethernet  HWaddr d6:44:81:74:46:bd
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::d444:81ff:fe74:46bd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3528 errors:0 dropped:592 overruns:0 frame:0
          TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:587864 (587.8 KB)  TX bytes:1488 (1.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home# 
```

```
root@ubuntu:/home# ifconfig
h4-eth0   Link encap:Ethernet  HWaddr 12:0c:fa:ba:51:15
          inet addr:10.0.0.4  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::100c:faff:feba:5115/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3552 errors:0 dropped:592 overruns:0 frame:0
          TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:595044 (595.0 KB)  TX bytes:1488 (1.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home# 
```

Use Iperf tool to measure and verify the available bandwidth between the hosts.

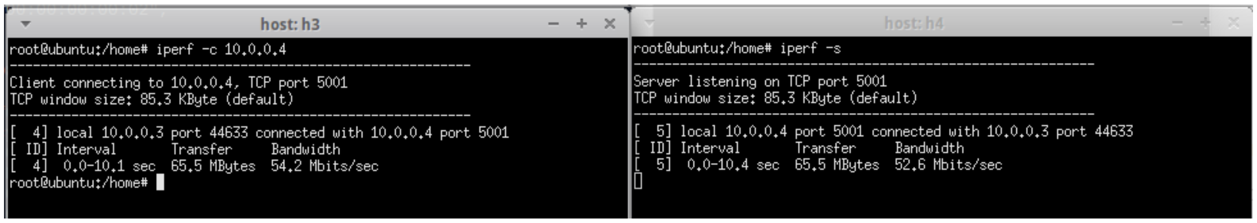
Start the Iperf server on the host h4 terminal using the command:

$iperf –s

Start the Iperf client on host h3 using the command:

$iperf –c 10.0.0.4

As shown in the screenshot below, the packets between the two hosts flow through the queue “q1 “ which is rate limited to 50 Mbps network bandwidth. You should get the available bandwidth measurement value close to the corresponding rate limit setting.



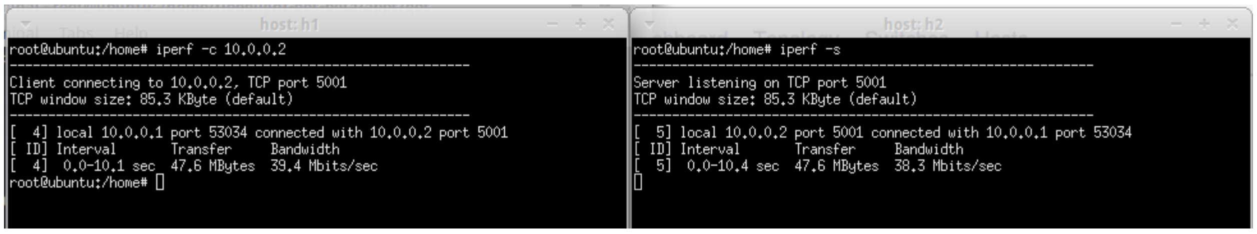
```
root@ubuntu:/home# iperf -c 10.0.0.4
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.3 port 44633 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.1 sec  65.5 MBytes  54.2 Mbits/sec
root@ubuntu:/home# 
```

```
root@ubuntu:/home# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.4 port 5001 connected with 10.0.0.3 port 44633
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.4 sec  65.5 MBytes  52.6 Mbits/sec
```

***Capture above screenshot to submit as part of your report submission for grading***

Similarly, start the Iperf server on host 2 and Iperf client on host 1 as shown.

As shown in the screenshot, the packets between the two hosts flow through the queue “q2 “ which is rate limited to 40 Mbps network bandwidth. You should get the available bandwidth measurement value close to the corresponding rate limit setting.



host: h1

```
root@ubuntu:/home# iperf -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.1 port 53034 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.1 sec  47.6 MBytes  39.4 Mbits/sec
root@ubuntu:/home# 
```

host: h2

```
root@ubuntu:/home# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 53034
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.4 sec  47.6 MBytes  38.3 Mbits/sec
```

***Capture above screenshot to submit as part of your report submission for grading***

## NOTE: DEBUGGING YOUR OPENFLOW CONTROL PLANE

Follow the below steps to understand how to debug your OpenFlow controller functionality using the Wireshark tool that got installed when you setup Mininet. More specifically, you will use Wireshark to know the Ethernet ports that have been used in each switch and capture the OpenFlow traffic between the controller and the Mininet environment.

Start wireshark (Just accept any alert message at the beginning)

$sudo wireshark

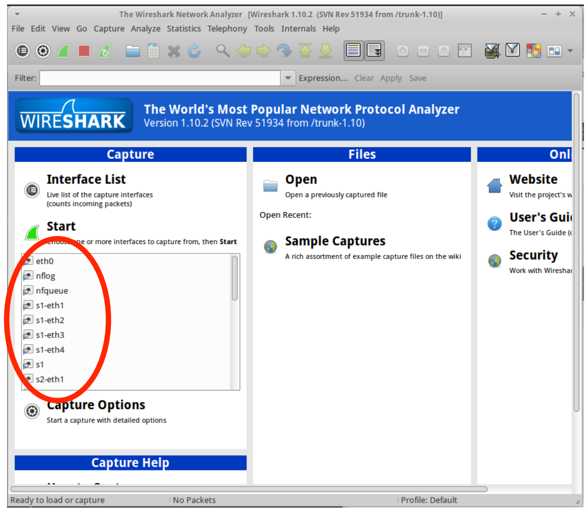
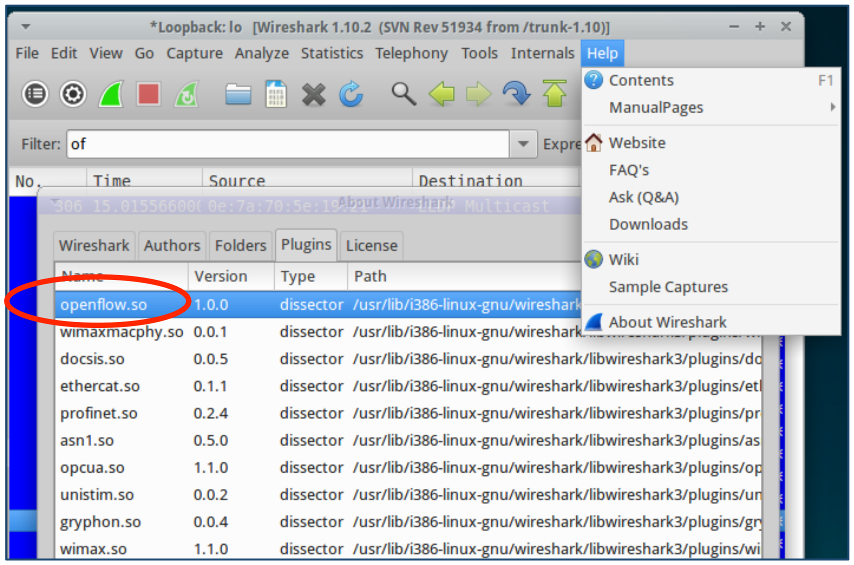

The Wireshark Network Analyzer [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
WIRESHARK
The World's Most Popular Network Protocol Analyzer
Version 1.10.2 (SVN Rev 51934 from /trunk-1.10)
Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
eth0
nflog
nfqueue
s1-eth1
s1-eth2
s1-eth3
s1-eth4
s1
s2-eth1
Capture Options
Start a capture with detailed options
Capture Help
Files
Open
Open a previously captured file
Open Recent:
Sample Captures
A rich assortment of example capture files on the wiki
Onl
Website
User's Gui
Security
Ready to load or capture
No Packets
Profile: Default

**Figure 9: Wireshark showing the virtual Ethernet ports for each switch**

As you will notice in Figure 9, the virtual Ethernet ports for each switch are displayed.

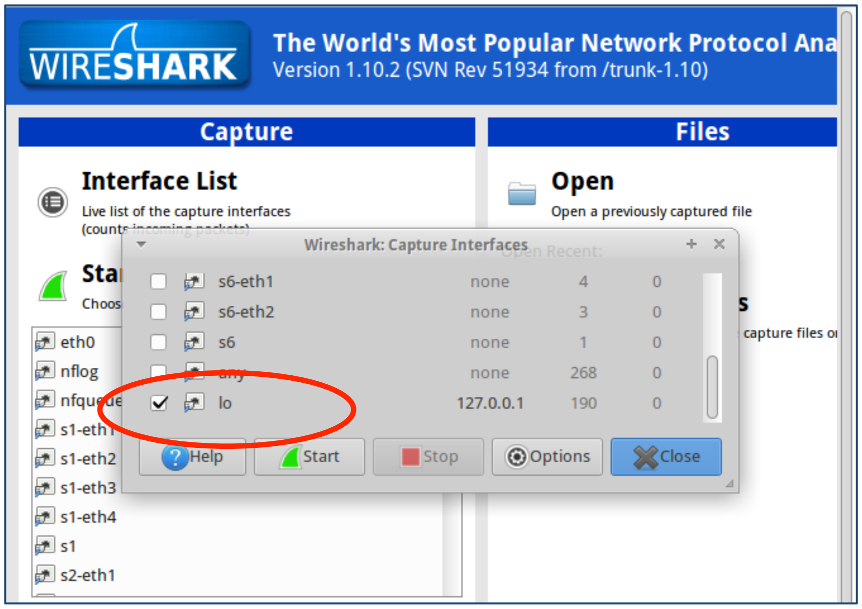
Make sure that the Openflow Plugin (openflow.so) is installed correctly by selecting ‘Help’, ‘About Wireshark’ and ‘Plugins’ tab.


*Loopback: lo [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Contents F1
ManualPages
Website
FAQ's
Ask (Q&A)
Downloads
Wiki
Sample Captures
About Wireshark
Wireshark Authors Folders Plugins License

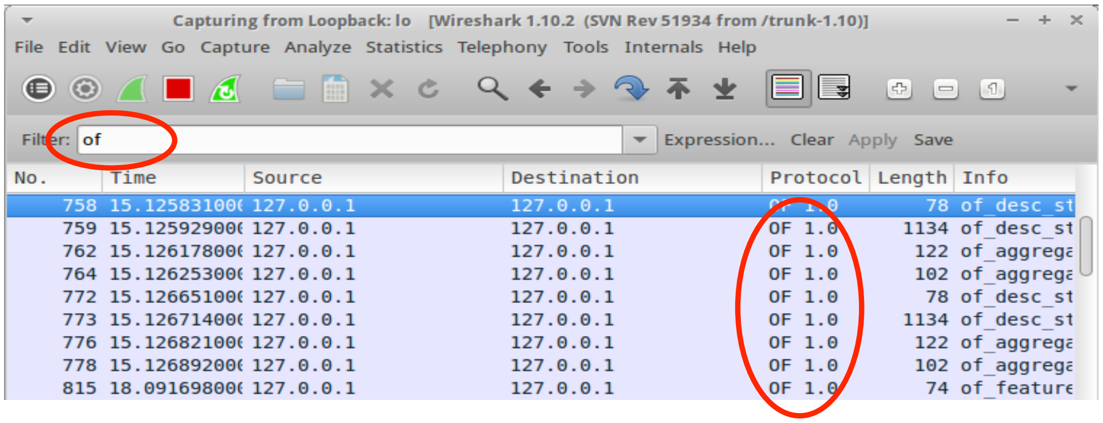
| Name | Version | Type | Path |
|---|---|---|---|
| openflow.so | 1.0.0 | dissector | /usr/lib/i386-linux-gnu/wireshark |
| wimaxmacphy.so | 0.0.1 | dissector | /usr/lib/i386-linux-gnu/wireshark |
| docsis.so | 0.0.5 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/do |
| ethercat.so | 0.1.1 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/et |
| profinet.so | 0.2.4 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/pr |
| asn1.so | 0.5.0 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/as |
| opcua.so | 1.1.0 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/op |
| unistim.so | 0.0.2 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/un |
| gryphon.so | 0.0.4 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/gr |
| wimax.so | 1.1.0 | dissector | /usr/lib/i386-linux-gnu/wireshark/libwireshark3/plugins/wi |

Once this is done, click on 'Interface List' option and check 'lo' for the controller and click on start.


WIRESHARK
The World's Most Popular Network Protocol Ana
Version 1.10.2 (SVN Rev 51934 from /trunk-1.10)
Capture
Files
Interface List
Live list of the capture interfaces
Open
Open a previously captured file
Wireshark: Capture Interfaces
eth0
nflog
s1-eth2
s1-eth3
s1-eth4
s1
s2-eth1
Help Start Stop Options Close

| | Interface | | | |
|---|---|---|---|---|
| ☐ | s6-eth1 | none | 4 | 0 |
| ☐ | s6-eth2 | none | 3 | 0 |
| ☐ | s6 | none | 1 | 0 |
| ☐ | any | none | 268 | 0 |
| ✓ | lo | 127.0.0.1 | 190 | 0 |

Type 'of' filter and apply changes to display only OpenFlow protocol.

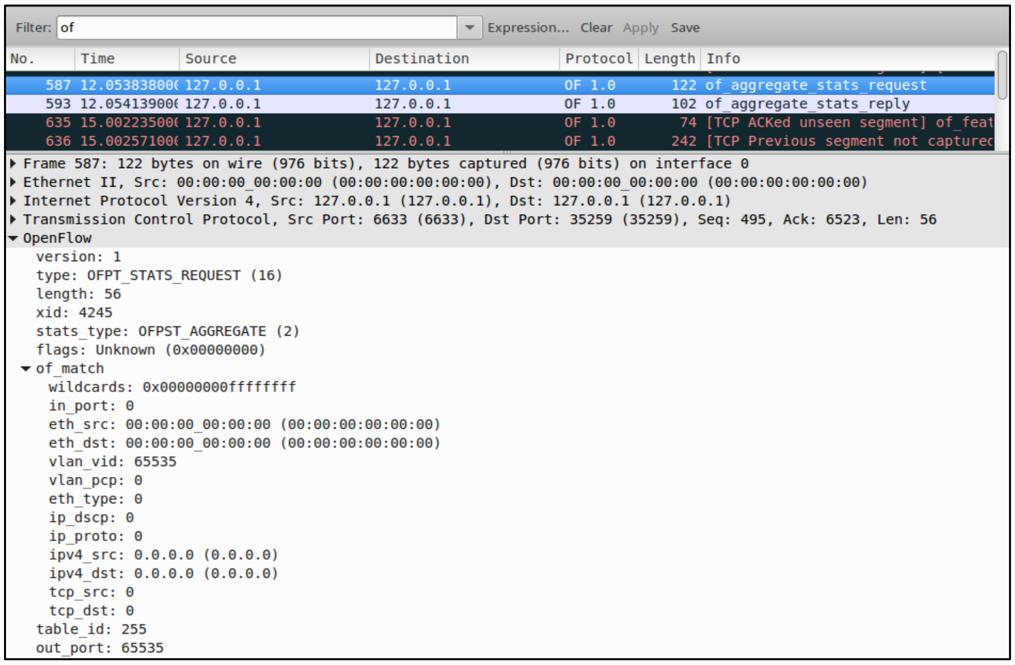

Capturing from Loopback: lo [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save

```
No.  Time            Source      Destination  Protocol Length Info
 758 15.12583100( 127.0.0.1  127.0.0.1   OF 1.0     78 of_desc_st
 759 15.12592900( 127.0.0.1  127.0.0.1   OF 1.0   1134 of_desc_st
 762 15.12617800( 127.0.0.1  127.0.0.1   OF 1.0    122 of_aggrega
 764 15.12625300( 127.0.0.1  127.0.0.1   OF 1.0    102 of_aggrega
 772 15.12665100( 127.0.0.1  127.0.0.1   OF 1.0     78 of_desc_st
 773 15.12671400( 127.0.0.1  127.0.0.1   OF 1.0   1134 of_desc_st
 776 15.12682100( 127.0.0.1  127.0.0.1   OF 1.0    122 of_aggrega
 778 15.12689200( 127.0.0.1  127.0.0.1   OF 1.0    102 of_aggrega
 815 18.09169800( 127.0.0.1  127.0.0.1   OF 1.0     74 of_feature
```

You can stop the collecting process as soon as you get some traffic. Select an OpenFlow control packet as shown in Figure 10 and analyze it.

***Provide a brief description of your Wireshark packet analysis for OpenFlow Protocol "of_aggregate_stats_request" packet as part of your report submission for grading.***



```
No.   Time         Source       Destination   Protocol Length Info
  587 12.05383800( 127.0.0.1    127.0.0.1     OF 1.0      122 of_aggregate_stats_request
  593 12.05413900( 127.0.0.1    127.0.0.1     OF 1.0      102 of_aggregate_stats_reply
  635 15.00223500( 127.0.0.1    127.0.0.1     OF 1.0       74 [TCP ACKed unseen segment] of_feat
  636 15.00257100( 127.0.0.1    127.0.0.1     OF 1.0      242 [TCP Previous segment not capturec
```

```
▸ Frame 587: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▸ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 35259 (35259), Seq: 495, Ack: 6523, Len: 56
▾ OpenFlow
    version: 1
    type: OFPT_STATS_REQUEST (16)
    length: 56
    xid: 4245
    stats_type: OFPST_AGGREGATE (2)
    flags: Unknown (0x00000000)
  ▾ of_match
      wildcards: 0x00000000ffffffff
      in_port: 0
      eth_src: 00:00:00_00:00:00 (00:00:00:00:00:00)
      eth_dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
      vlan_vid: 65535
      vlan_pcp: 0
      eth_type: 0
      ip_dscp: 0
      ip_proto: 0
      ipv4_src: 0.0.0.0 (0.0.0.0)
      ipv4_dst: 0.0.0.0 (0.0.0.0)
      tcp_src: 0
      tcp_dst: 0
    table_id: 255
    out_port: 65535
```

**Figure 10: Wireshark packet analysis window for inspecting OpenFlow Protocol fields**

## 3.5 Load Balancer Configuration in Controller Application

Stop the previous running applications of Floodlight and Mininet. Download the latest version of Floodlight using the below command to obtain the Load Balancing module:

$git clone git://github.com/floodlight/floodlight.git

Enter the floodlight folder and build the application

$cd floodlight

$ sudo git checkout  remotes/origin/v0.91

$sudo ant;

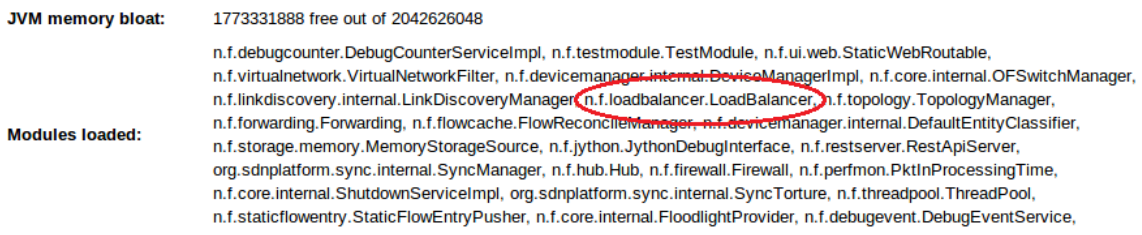
Once the build is successful, edit the floodlight.sh file. Change the server parameter from d64 to d32 as shown below: (similar to steps in Section 3.2)

$vi floodlight.sh

Change -d64 to -d32 and run the application using the command:

```
$./floodlight.sh
```

You should see your new Floodlight controller application running in the browser ([http://localhost:8080/ui/index.html](http://localhost:8080/ui/index.html)) and also the Load Balancer module should be enabled.



| | |
|---|---|
| **JVM memory bloat:** | 1773331888 free out of 2042626048 |
| **Modules loaded:** | n.f.debugcounter.DebugCounterServiceImpl, n.f.testmodule.TestModule, n.f.ui.web.StaticWebRoutable, n.f.virtualnetwork.VirtualNetworkFilter, n.f.devicemanager.internal.DeviceManagerImpl, n.f.core.internal.OFSwitchManager, n.f.linkdiscovery.internal.LinkDiscoveryManager, n.f.loadbalancer.LoadBalancer, n.f.topology.TopologyManager, n.f.forwarding.Forwarding, n.f.flowcache.FlowReconcileManager, n.f.devicemanager.internal.DefaultEntityClassifier, n.f.storage.memory.MemoryStorageSource, n.f.jython.JythonDebugInterface, n.f.restserver.RestApiServer, org.sdnplatform.sync.internal.SyncManager, n.f.hub.Hub, n.f.firewall.Firewall, n.f.perfmon.PktInProcessingTime, n.f.core.internal.ShutdownServiceImpl, org.sdnplatform.sync.internal.SyncTorture, n.f.threadpool.ThreadPool, n.f.staticflowentry.StaticFlowEntryPusher, n.f.core.internal.FloodlightProvider, n.f.debugevent.DebugEventService, |

Reserve the below new topology shown in Figure 11 in the mininet environment using the command:

```
$ sudo mn --controller=remote --mac --topo=tree,3
```

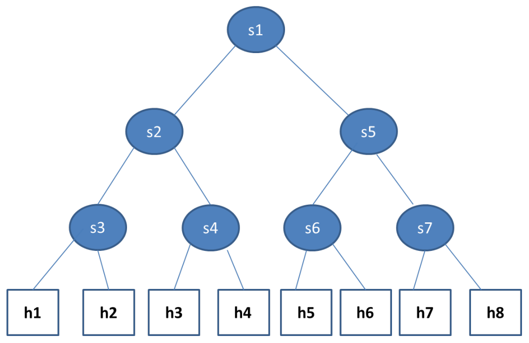


**Figure 11: Load Balancing Experiment Topology**

Test the setup using the command pingall in the mininet CLI.

```
$mininet>pingall
```

**Note: This command is important,** as the switches will learn the attached hosts using ARP to forward the packets accordingly.

### 3.6 Load Balancer Application Experimentation using Ping Tool

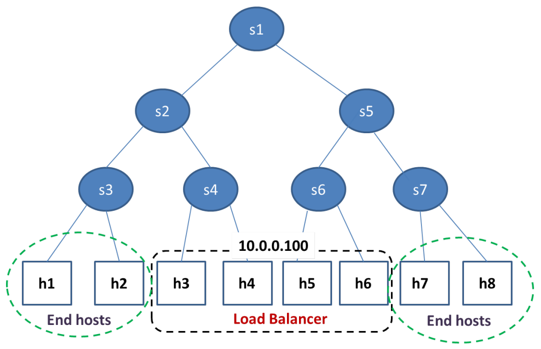


**Figure 12: Load Balancing Experiment Topology with pools annotations**

We will create a load balancer pool for hosts (h3, h4) for handling Ping protocol packets and a virtual IP for the load balancer server using the “load_balancer.sh” script in Blackboard. Place the script in a path in your home directory and run the script.

$./load_balancer.sh

This will install the flow rules required to create the load balancer server 10.0.0.100

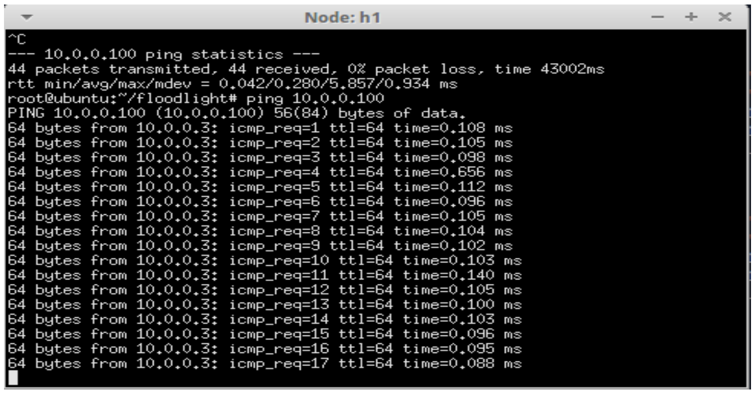
Now, start the xterm terminals on the mininet CLI for hosts h1 and h2 using the command:

$mininet>xterm h1 h2

Ping the load balancer 10.0.0.100 from host h1 terminal

This will route the ping on host h3 10.0.0.3

***Capture below screenshot to submit as part of your report submission for grading***

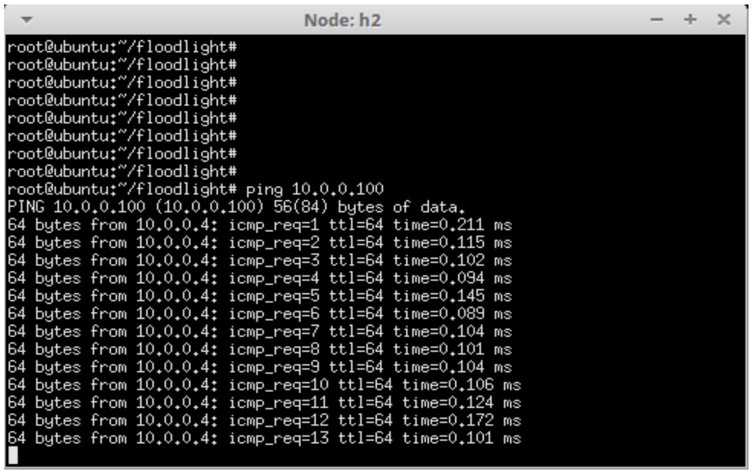


```
Node: h1
^C
--- 10.0.0.100 ping statistics ---
44 packets transmitted, 44 received, 0% packet loss, time 43002ms
rtt min/avg/max/mdev = 0.042/0.280/5.857/0.934 ms
root@ubuntu:~/floodlight# ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.108 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.105 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.098 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=0.656 ms
64 bytes from 10.0.0.3: icmp_req=5 ttl=64 time=0.112 ms
64 bytes from 10.0.0.3: icmp_req=6 ttl=64 time=0.096 ms
64 bytes from 10.0.0.3: icmp_req=7 ttl=64 time=0.105 ms
64 bytes from 10.0.0.3: icmp_req=8 ttl=64 time=0.104 ms
64 bytes from 10.0.0.3: icmp_req=9 ttl=64 time=0.102 ms
64 bytes from 10.0.0.3: icmp_req=10 ttl=64 time=0.103 ms
64 bytes from 10.0.0.3: icmp_req=11 ttl=64 time=0.140 ms
64 bytes from 10.0.0.3: icmp_req=12 ttl=64 time=0.105 ms
64 bytes from 10.0.0.3: icmp_req=13 ttl=64 time=0.100 ms
64 bytes from 10.0.0.3: icmp_req=14 ttl=64 time=0.103 ms
64 bytes from 10.0.0.3: icmp_req=15 ttl=64 time=0.096 ms
64 bytes from 10.0.0.3: icmp_req=16 ttl=64 time=0.095 ms
64 bytes from 10.0.0.3: icmp_req=17 ttl=64 time=0.088 ms
```

Now Ping the load balancer 10.0.0.100 from host h2 terminal

This will route the Ping on host h4 10.0.0.4 accordingly

***Capture below screenshot to submit as part of your report submission for grading***



```
Node: h2
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight#
root@ubuntu:~/floodlight# ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_req=1 ttl=64 time=0.211 ms
64 bytes from 10.0.0.4: icmp_req=2 ttl=64 time=0.115 ms
64 bytes from 10.0.0.4: icmp_req=3 ttl=64 time=0.102 ms
64 bytes from 10.0.0.4: icmp_req=4 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=5 ttl=64 time=0.145 ms
64 bytes from 10.0.0.4: icmp_req=6 ttl=64 time=0.089 ms
64 bytes from 10.0.0.4: icmp_req=7 ttl=64 time=0.104 ms
64 bytes from 10.0.0.4: icmp_req=8 ttl=64 time=0.101 ms
64 bytes from 10.0.0.4: icmp_req=9 ttl=64 time=0.104 ms
64 bytes from 10.0.0.4: icmp_req=10 ttl=64 time=0.106 ms
64 bytes from 10.0.0.4: icmp_req=11 ttl=64 time=0.124 ms
64 bytes from 10.0.0.4: icmp_req=12 ttl=64 time=0.172 ms
64 bytes from 10.0.0.4: icmp_req=13 ttl=64 time=0.101 ms
```

## 4. What to turn in for Grading?

1. Add the **two** screenshots taken in **Step 3.4**. Also, provide a brief description of your Wireshark packet analysis for “Header” and “Stats Request”.

2. From the QoS Experimentation scenario shown in below Figure 13, show a screenshot and describe the results obtained using Iperf server on host1 and Iperf client on host 4. Which queue number in the switches is used for the traffic flow between host1 and host4? Similarly, which queue number in the switches is used when you start Iperf client on host 2 and Iperf server on host 1?

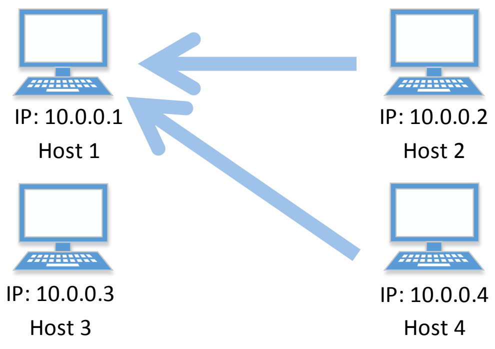


**Figure 13: QoS Experimentation Scenario**

3. Consider the new topology shown in Figure 14 with 6 hosts, 7 switches and 1 controller. Describe the commands to establish a new queue ‘q3’ of 80 Mbps between host ‘h5’ as source and host ‘h6’ as destination (i.e., give the two commands for queue configuration and for adding the flows).

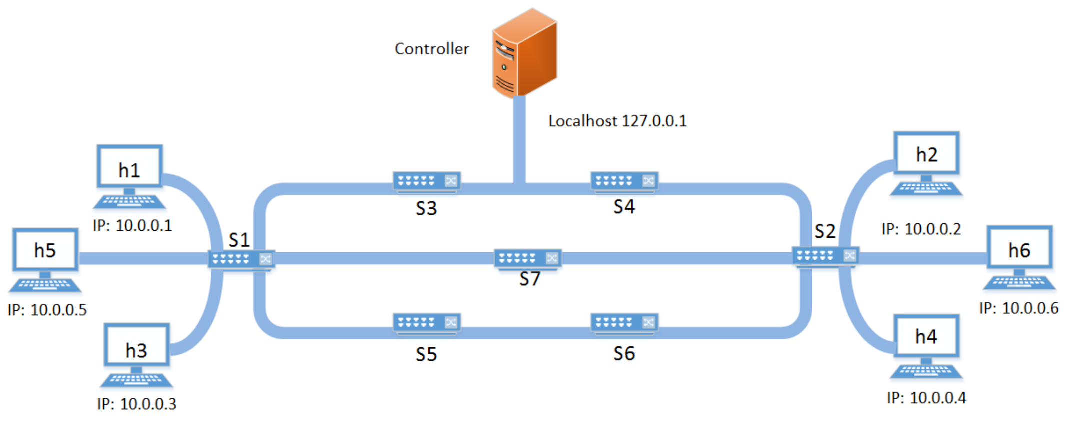


**Figure 14: New QoS Experimentation Topology**

4. Attach the **two** screenshots taken in **Step 3.5**

5. This question requires that you extend the load_balancer.sh script and run new Load Balancer Experimentation using the following steps:

i. Scale the load balancer to handle more requests by adding two new hosts h5 (10.0.0.5) and h6 (10.0.0.6) to the load balancer pool and adding the appropriate entries in the load_balancer.sh script. Run the load_balancer.sh script again to update the new flow rules.

ii. On the mininet CLI, start the xterm terminals for the new end-hosts h7 and h8 by giving the below command:
    a. $mininet>xterm h7 h8

iii. Start the ping command from hosts h1 and h2 terminals to load balancer 10.0.0.100 and allow the ping to run continuously. Simultaneously, start the ping command from hosts h7 and h8 terminals to the load balancer 10.0.0.100.

What happens when you ping from new end-hosts h7 and h8 to the load balancer 10.0.0.100? Which hosts are responding to the new requests and what does this result suggest? Attach a **single** screenshot with all the four ping windows running simultaneously.