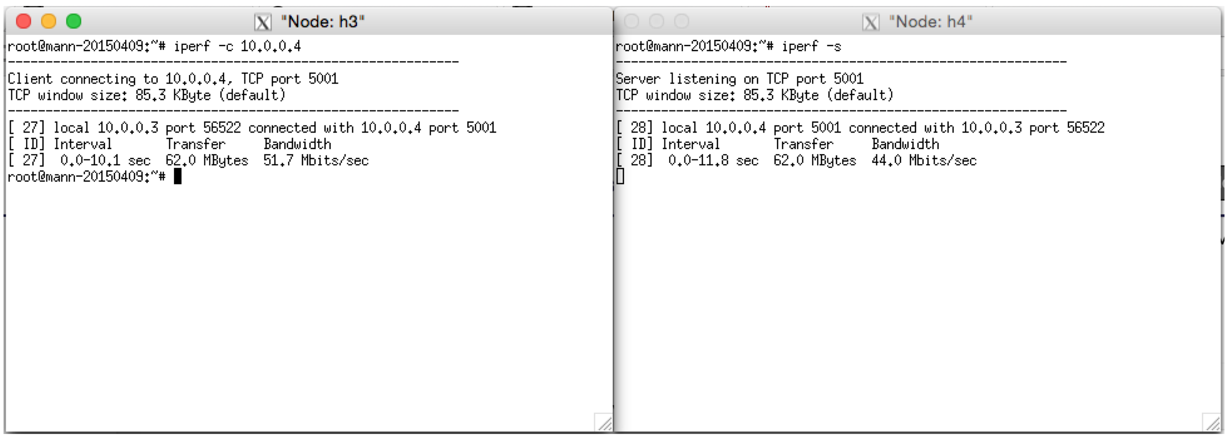
CS 7001-03: Report for GENI Lab 3 - QoS Configuration and Load Balancing using Software-Defined Networking

Chanmann Lim
`cl9p8@mail.mail.missouri.edu`

April 14, 2015

1. Screenshots taken in Step 3.4:



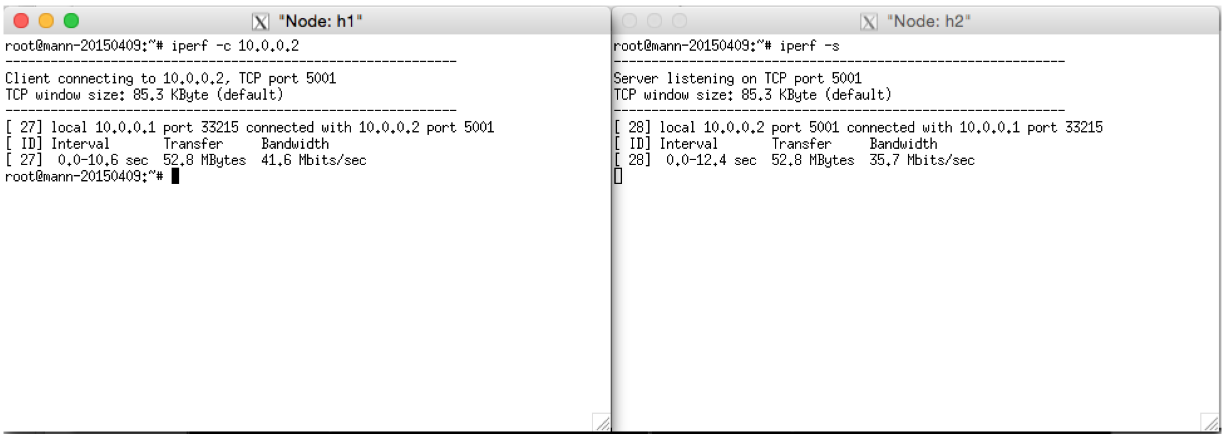
"Node: h3"

```
root@mann-20150409:~# iperf -c 10.0.0.4
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 27] local 10.0.0.3 port 56522 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 27]  0.0-10.1 sec  62.0 MBytes  51.7 Mbits/sec
root@mann-20150409:~# 
```

"Node: h4"

```
root@mann-20150409:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 28] local 10.0.0.4 port 5001 connected with 10.0.0.3 port 56522
[ ID] Interval       Transfer     Bandwidth
[ 28]  0.0-11.8 sec  62.0 MBytes  44.0 Mbits/sec
```

Figure 1: Iperf test from h3 to h4



"Node: h1"

```
root@mann-20150409:~# iperf -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 27] local 10.0.0.1 port 33215 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 27]  0.0-10.6 sec  52.8 MBytes  41.6 Mbits/sec
root@mann-20150409:~# 
```

"Node: h2"

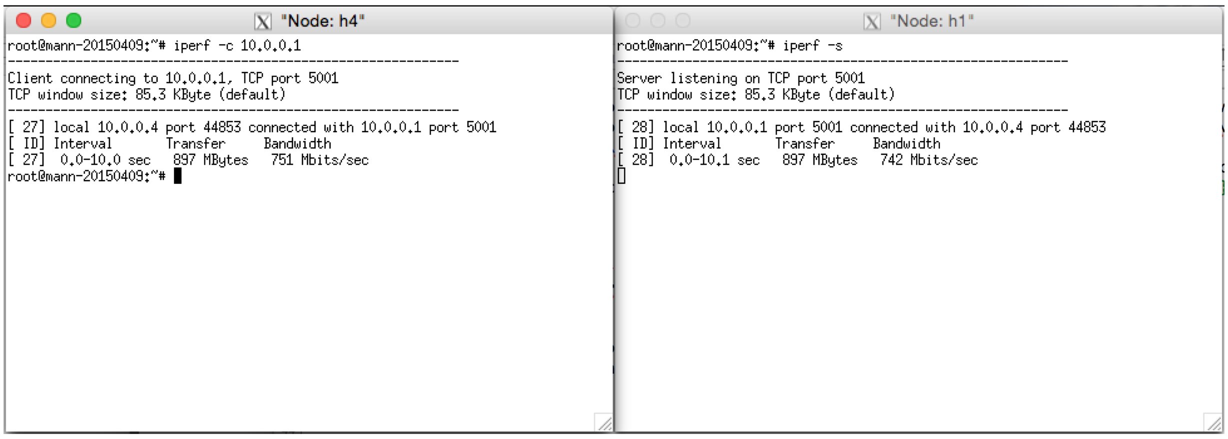
```
root@mann-20150409:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 28] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 33215
[ ID] Interval       Transfer     Bandwidth
[ 28]  0.0-12.4 sec  52.8 MBytes  35.7 Mbits/sec
```

Figure 2: Iperf test from h1 to h2

The packets captured by Wireshark with "lo" interface and "of" filter in **step. 3.4** list only the OpenFlow control packet of the loopback interface, and when clicking on an "of_aggregate_stats_request" packet, we obtain the detail OpenFlow headers in the packet such as *version*, *type*, *length*, *xid*, *stats_type*, *flags*, *of_match*, *table_id*, and *out_port*.

And the *of_match* header contains *wildcards*: Wildcards fields, *in_port*: Input switch port, *eth_src*: Ethernet source address, *eth_dst*: Ethernet destination address, *vlan_vid*: Input VLAN id, *vlan_pcp*: Input VLAN priority, *eth_type*: Ethernet frame type, *id_dscp*: IP ToS (actually DSCP field, 6 bits), *ip_proto*: IP protocol or lower 8 bits of ARP opcode, *ipv4_src*: IP source address, *ipv4_dst*: IP destination address, *tcp_src*: TCP source port, *tcp_dst*: TCP destination port.

2. Screenshot of Iperf test from host4 to host1:



```
root@mann-20150409:~# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 27] local 10.0.0.4 port 44853 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 27]  0.0-10.0 sec   897 MBytes   751 Mbits/sec
root@mann-20150409:~# 
```

```
root@mann-20150409:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 28] local 10.0.0.1 port 5001 connected with 10.0.0.4 port 44853
[ ID] Interval       Transfer     Bandwidth
[ 28]  0.0-10.1 sec   897 MBytes   742 Mbits/sec
```

Figure 3: Iperf test from h4 to h1

By using the topology created in **step 3.3.1** and the QOS configuration in **step 3.3.2**, the traffic flow between host4 and host1 is conducted through queue "0" which had configured with the maximum bandwidth up to 1 Gbps, similarly since the traffic flow from host2 to host1 doesn't match with both queue "1" and queue "2"'s rules "50Mbps-h3-h4" and "40Mbps-h1-h2" respectively it will fallback to use queue "0" in the switches.

3. Assuming that we have the topology with 6 hosts, 7 switches and 1 controller as shown in the Figure 14 of the GENI Lab 3 instruction. To establish a new queue 'q3' of 80 Mbps between host 'h5' as source and host 'h6' as destination, first we need to configure the queue by editing the `mininet-add-queues.py`

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

then change the `queuecmd` variable at line 67 to

queuecmd = "sudo ovs-vsctl %s -- --id=@defaultqos create qos type=linux-htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2,3=@q3 -- --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 -- --id=@q3 create queue other-config:max-rate=80000000 other-config:min-rate=2000000" % config_strings[sw]

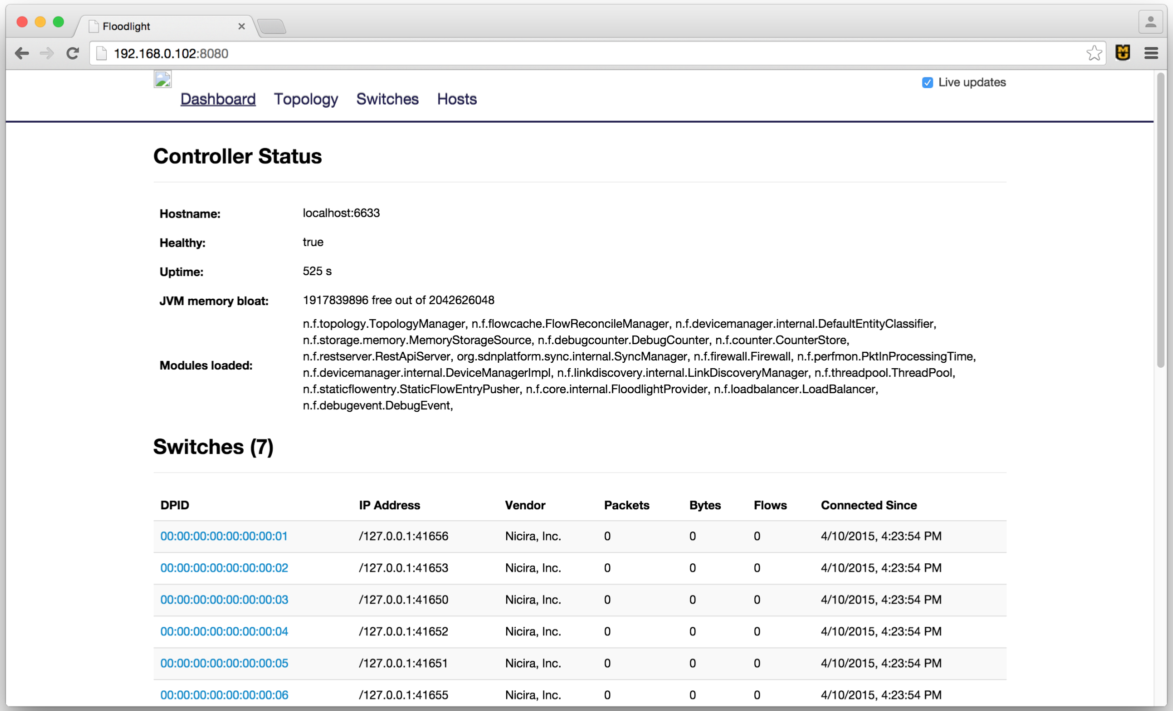
then run:

```
# ./floodlight-qos-beta/apps/qos/mininet-add-queues.py
```

It will configure queue "1", queue "2", and queue "3" with 50 Mbps, 40 Mbps and 80 Mbps respectively for all ports of all switches. Next change directory to `floodlight-qos-beta/apps/qos/` and execute `qospath2.py` command to establish flow between 'h5' and 'h6' using queue "3"

```
# ./qospath2.py -p 8080 -a -N 80Mbps-h5-h6 -J '{"eth-type":"0x0800","queue":"3"}' \
   -S 10.0.0.5 -D 10.0.0.6
```

4. Screenshots taken in Step 3.5:



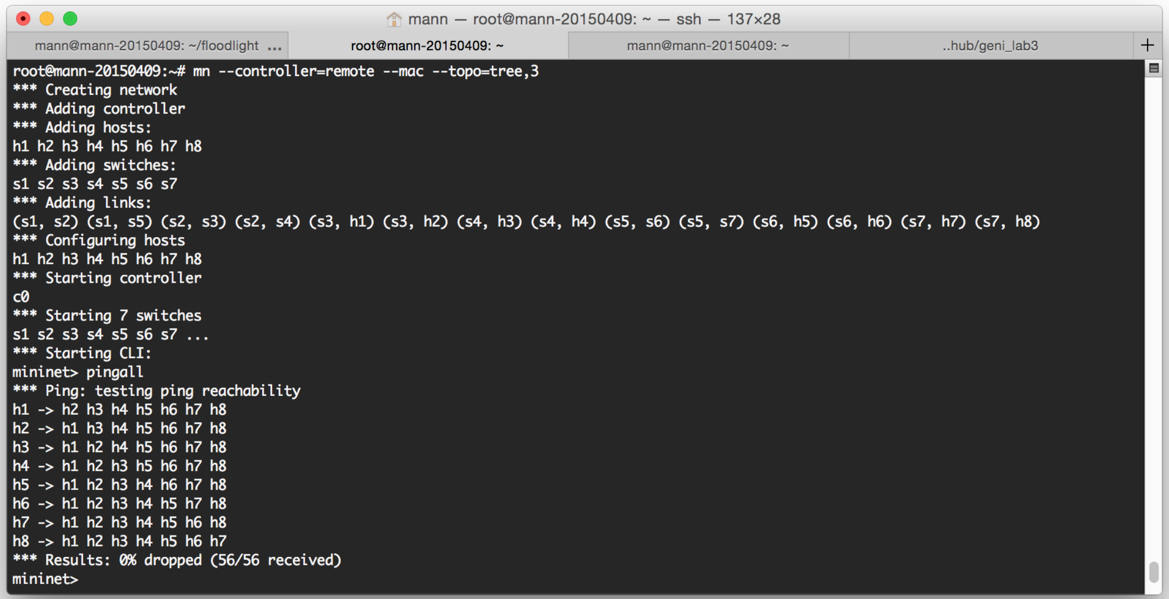
Controller Status

Hostname:	localhost:6633
Healthy:	true
Uptime:	525 s
JVM memory bloat:	1917839896 free out of 2042626048
Modules loaded:	n.f.topology.TopologyManager, n.f.flowcache.FlowReconcileManager, n.f.devicemanager.internal.DefaultEntityClassifier, n.f.storage.memory.MemoryStorageSource, n.f.debugcounter.DebugCounter, n.f.counter.CounterStore, n.f.restserver.RestApiServer, org.sdnplatform.sync.internal.SyncManager, n.f.firewall.Firewall, n.f.perfmon.PktInProcessingTime, n.f.devicemanager.internal.DeviceManagerImpl, n.f.linkdiscovery.internal.LinkDiscoveryManager, n.f.threadpool.ThreadPool, n.f.staticflowentry.StaticFlowEntryPusher, n.f.core.internal.FloodlightProvider, n.f.loadbalancer.LoadBalancer, n.f.debugevent.DebugEvent,

Switches (7)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:00:01	/127.0.0.1:41656	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM
00:00:00:00:00:00:00:02	/127.0.0.1:41653	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM
00:00:00:00:00:00:00:03	/127.0.0.1:41650	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM
00:00:00:00:00:00:00:04	/127.0.0.1:41652	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM
00:00:00:00:00:00:00:05	/127.0.0.1:41651	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM
00:00:00:00:00:00:00:06	/127.0.0.1:41655	Nicira, Inc.	0	0	0	4/10/2015, 4:23:54 PM

Figure 4: Floodlight with LoadBalancer module loaded


mann — root@mann-20150409: ~ — ssh — 137x28
mann@mann-20150409: ~/floodlight ...
root@mann-20150409: ~
mann@mann-20150409: ~
..hub/geni_lab3

```
root@mann-20150409:~# mn --controller=remote --mac --topo=tree,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```

Figure 5: minute pinball in 'tree,3' topology

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: $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.