

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:

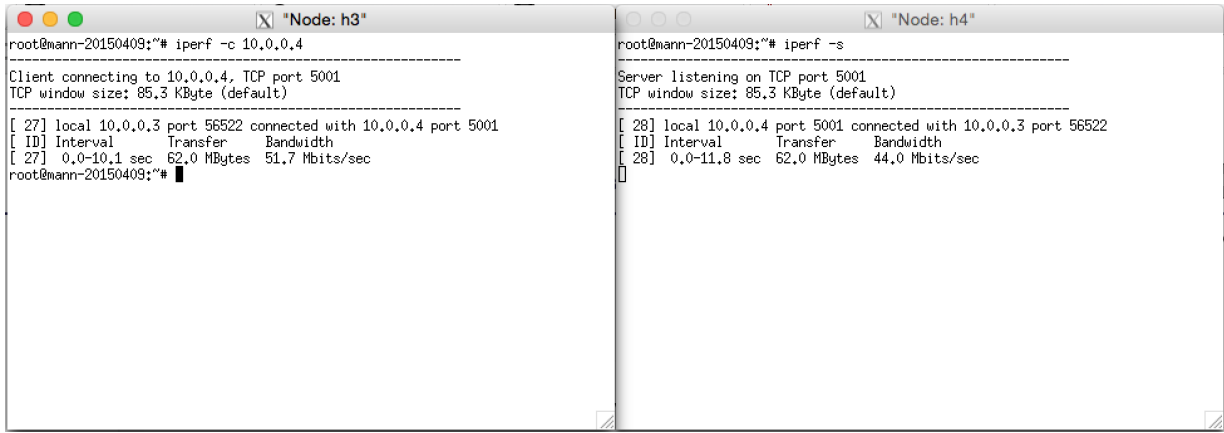


Figure 1: Iperf test from h3 to h4

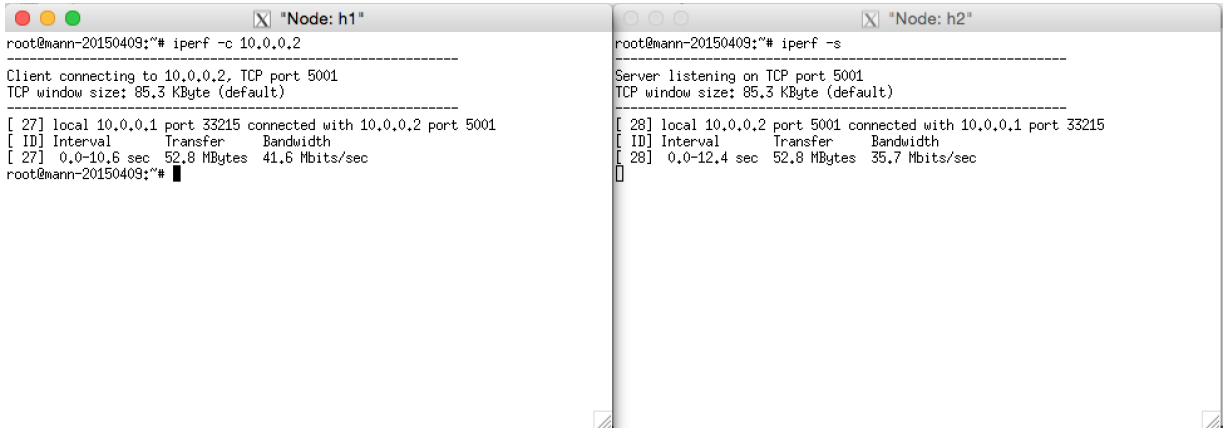


Figure 2: Iperf test from h1 to h2

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:

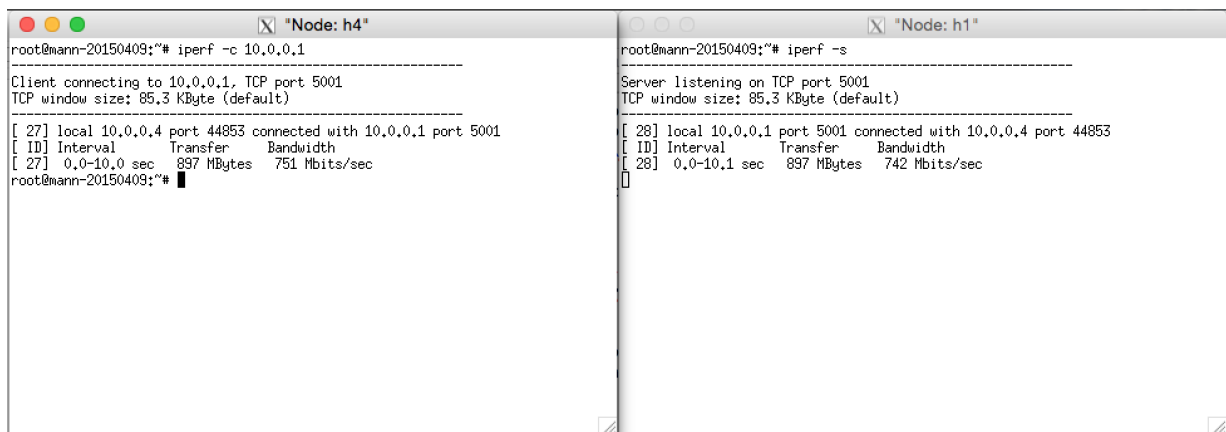


Figure 3: Iperf test from h4 to h1

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=@q0,@q1,@q2,@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" % con-
fig_strings[sw]
```

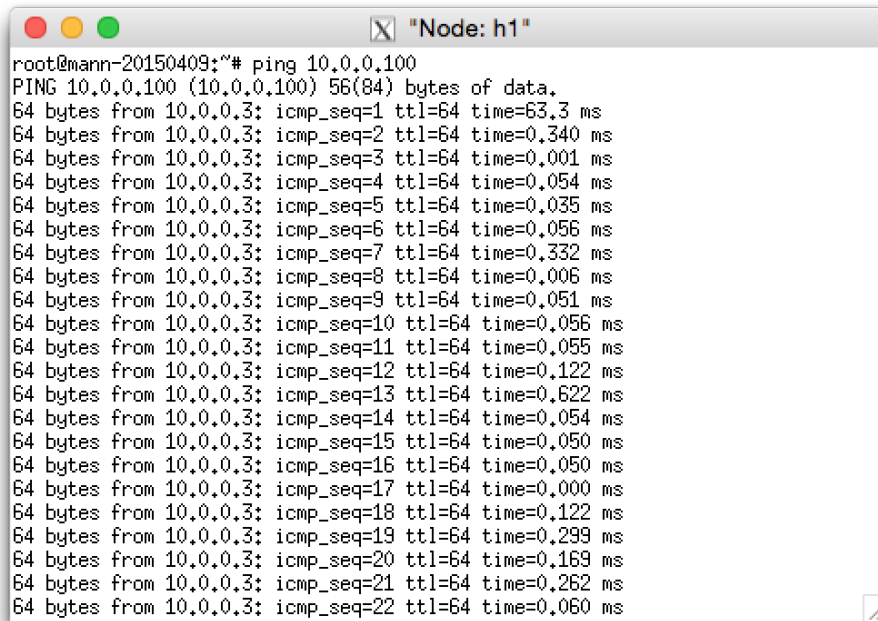
then run:

```
# ./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"

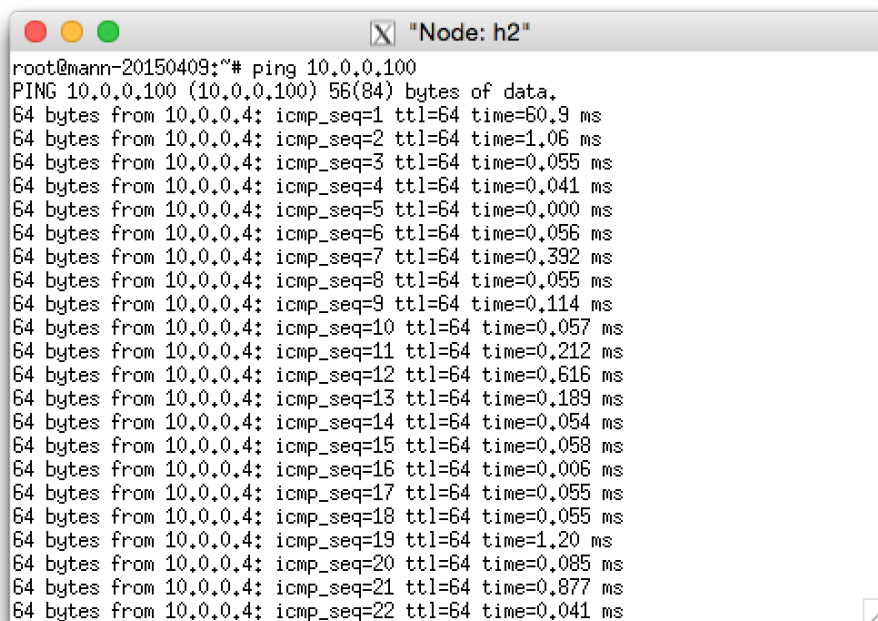
```
# ./qospath2.py -p 8080 -a -N 80Mbps-h5-h6 -J '{"eth-type":"0x0800","queue":"3"}' \
-S 10.0.0.5 -D 10.0.0.6
```

4. Two screenshots taken in Step 3.6:



```
root@mann-20150409:~# 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_seq=1 ttl=64 time=63.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.340 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.001 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.056 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.332 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.006 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.051 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.056 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=0.055 ms
64 bytes from 10.0.0.3: icmp_seq=12 ttl=64 time=0.122 ms
64 bytes from 10.0.0.3: icmp_seq=13 ttl=64 time=0.622 ms
64 bytes from 10.0.0.3: icmp_seq=14 ttl=64 time=0.054 ms
64 bytes from 10.0.0.3: icmp_seq=15 ttl=64 time=0.050 ms
64 bytes from 10.0.0.3: icmp_seq=16 ttl=64 time=0.050 ms
64 bytes from 10.0.0.3: icmp_seq=17 ttl=64 time=0.000 ms
64 bytes from 10.0.0.3: icmp_seq=18 ttl=64 time=0.122 ms
64 bytes from 10.0.0.3: icmp_seq=19 ttl=64 time=0.299 ms
64 bytes from 10.0.0.3: icmp_seq=20 ttl=64 time=0.169 ms
64 bytes from 10.0.0.3: icmp_seq=21 ttl=64 time=0.262 ms
64 bytes from 10.0.0.3: icmp_seq=22 ttl=64 time=0.060 ms
```

Figure 4: Ping LoadBalancer from host1



```
root@mann-20150409:~# 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_seq=1 ttl=64 time=60.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=1.06 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.000 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.056 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.392 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.055 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.114 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.057 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=0.212 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=0.616 ms
64 bytes from 10.0.0.4: icmp_seq=13 ttl=64 time=0.189 ms
64 bytes from 10.0.0.4: icmp_seq=14 ttl=64 time=0.054 ms
64 bytes from 10.0.0.4: icmp_seq=15 ttl=64 time=0.058 ms
64 bytes from 10.0.0.4: icmp_seq=16 ttl=64 time=0.006 ms
64 bytes from 10.0.0.4: icmp_seq=17 ttl=64 time=0.055 ms
64 bytes from 10.0.0.4: icmp_seq=18 ttl=64 time=0.055 ms
64 bytes from 10.0.0.4: icmp_seq=19 ttl=64 time=1.20 ms
64 bytes from 10.0.0.4: icmp_seq=20 ttl=64 time=0.085 ms
64 bytes from 10.0.0.4: icmp_seq=21 ttl=64 time=0.877 ms
64 bytes from 10.0.0.4: icmp_seq=22 ttl=64 time=0.041 ms
```

Figure 5: Ping LoadBalancer from host2

5. After editing the `load_balancer.sh` to add hosts h5 (10.0.0.5) and h6 (10.0.0.6) to the load balancer pool and start ping from h1 and h2 then ping from the new end-hosts h7 and h8 to the load balancer (10.0.0.100), we observed that h7 routed the Ping to h5 and h8 routed the Ping to h6 which indicates that the load balancer serves the requests (Ping in this case) by forwarding and distributing them across all members in load balancer pool.

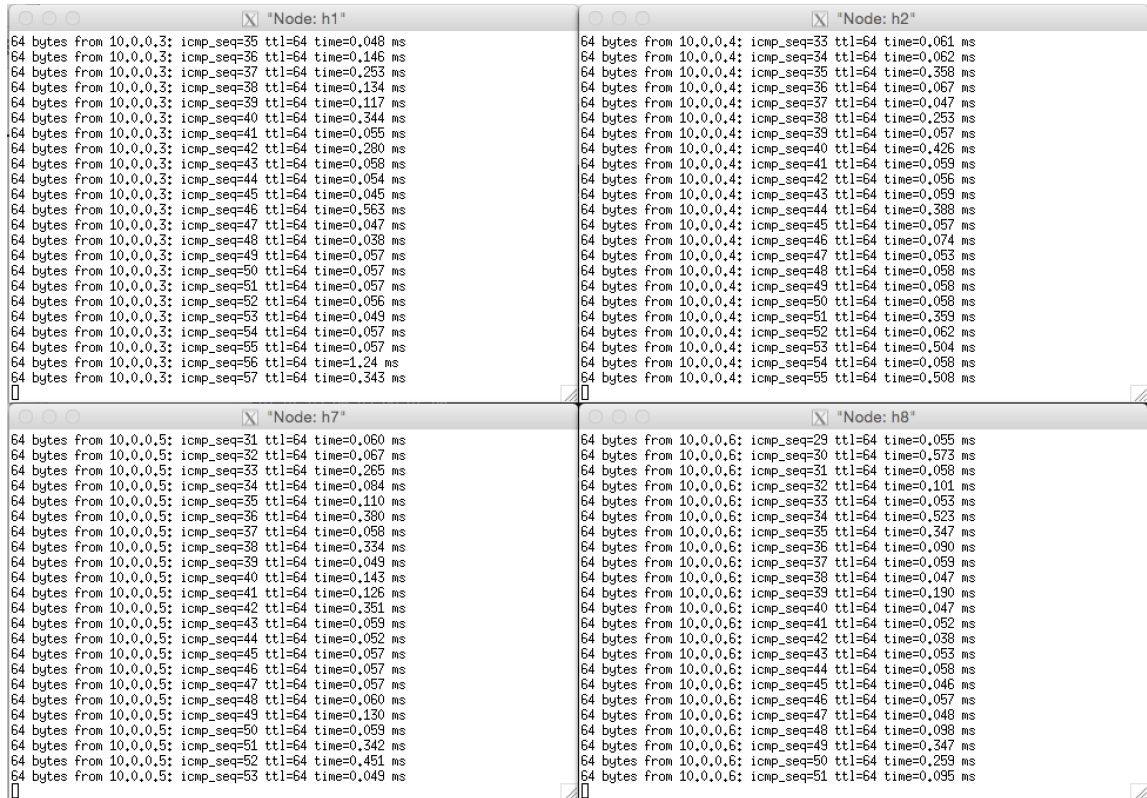


Figure 6: Ping LoadBalancer from host1, host2, host7 and host8