QoS based Optimal Route Discovery Using OpenFlow

Dhaval Sonawane, Vaibhav Rajadhyaksha, Elson Serrao, Chethan Nannivala Thipperudrappa

North Carolina State University
CSC/ECE-573-002: Internet Protocols
Final Report

2nd December, 2015

1 Introduction

Routing algorithms traditionally route packets using greedy approach to datagram delivery. Algorithms are broadly classified into Distance Vector and Link State algorithms. The primary metrics used by these algorithms to calculate shortest path is number of hops or the link-cost defined based on the bandwidth. Essentially when a new shortest path is identified, all the traffic is routed in that direction. Nowhere in this technique do we take the QoS requirements of the traffic into consideration. There may exist cases where a new path is calculated and the traffic is routed to it, even though the old path could still be satisfying the QoS requirements.

In our project we design a routing scheme based on the QoS requirements of the traffic.
We take into consideration link latency, bandwidth utilization and queue/port drops as the QoS metrics to calculate the shortest path based on the service requirement (ToS/DSCP). These metrics are fed to a cost function for calculating the shortest path based on the service requirements.

2 Software Design

2.1 Major Components

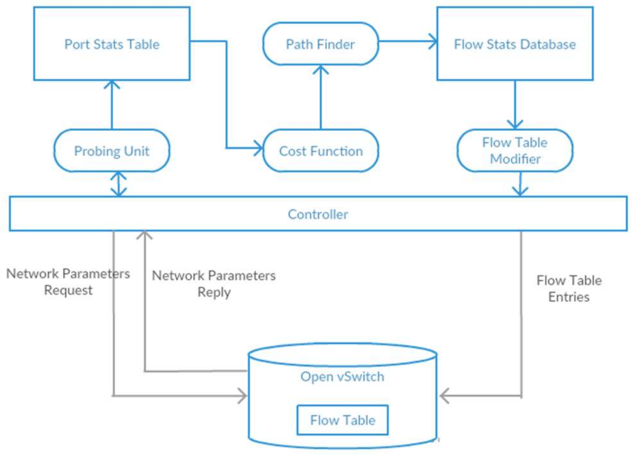

Figure 1: Flow Diagram

2.1.1 Control Units

Probing Unit: This unit is used to query the switches to obtain the network parameters. These parameters are stored as a part of the port stats table. They are then used by the cost function unit to calculate the link costs.

Calculated Network Parameters:

Bandwidth Utilization: The current rate at which the various links in the network are transmitting. Higher bandwidth utilization denotes more traffic on the link.

Packet Drops: The number of packets being dropped from the tail of the queue in the switches. This depicts congestion in the network. Hence it becomes an important parameter to take into consideration while calculating the costs for the links. Delay intolerant traffic can't be sent on such links with high number of packet drops.

Link Latency and Calculation: Network latency is an expression of how much time it takes for a packet of data to get from one designated point to another. In our project we calculate the link latency i.e. delays experienced by packets at the input/output queues of switches.
We are using the method described in the paper 'Monitoring latency with Open-Flow'. Accordingly we send a special probe packet from a controller and back while measuring the amount of time it took to do so. The special packet is identified by (0x07c3) as the Ethernet-type. The payload consists of the switch's mac-address, port it is to be sent out from and time-stamp of its creation.
Initially, when the switches come up the controller writes flow table entries to direct them to forward packets with Ethernet-type 0x07c3 back to the controller. Once the controller receives the packet

back it measures the current time. The latency calculation is done as follows:

$$T_{link(s1,s2)} = T_{total} - [(T_{s1} + T_{s2})/2]$$

Where,
T_{total}- the total time
T_{s1}- the RTT between the controller and s1
T_{s2}- the RTT between the controller and s2

The RTT between the controller and the switches is calculated by sending latency packets to be sent out from controller facing interface of the switches.

QOS Index Calculator:
QOS Index Calculator uses a cost function using the link parameters saved in the port stats table to calculate the QOS index for a particular link. This information is fed to the path finder to determine the optimal path.

*QosIndexCalculator = K1 * Bandwidth utilization + n (K2 Latency + K3 QueueDrops)*

QosIndexCalculator calculates the cost of each link by giving different weights of K1, K2, and K3 for each type of traffic class.

Voice traffic: Voice traffic is extremely sensitive to delay and less tolerant to data loss. K2 has higher weightage and K3 has less weightage.
Video traffic: Video traffic can be classified as medium tolerant when it comes to delay as well as loss. Thus K2 and K3 will have a medium weightage.

Critical traffic: Business traffic is intolerant to delay but highly sensitive to data loss. Thus we have a higher weightage for data loss (K3) and lesser weightage for latency while calculating the cost of the link.

K1= 100, the value for this constant is same for best effort and media traffic.
K2= [0.01, 0.1], Voice=0.1, Video=0.07, Critical=0.01, Best effort=0 (latency is not considered for calculating the cost of the link for best effort traffic).
K3: [0.001, 0.01], Voice= 0.003, Video= 0.007, Critical=0.01, Best effort=0 (Again queue drops are not considered for calculating the cost of the link for best effort traffic).

Path Finder
This module runs the Floyd-Warshall algorithm to find minimal cost from a source to destination. Unlike traditional routing the cost used here is a function of bandwidth, latency and queue-drops. Thus based on the ToS value of the traffic differential treatment is provided for different kinds of traffic. Optimal route is calculated based on the level of tolerance of the traffic towards latency and data loss.

Flow Writer
This component writes the flow table entries in the switches based on the updated values in the path_map. Flows are written incrementally i.e. only when there are updates that happen on the flow entries in the path_map.

2.1.2 Dictionary Tables

We have used dictionaries in python to store statistics and adjacency information.

Switch Adjacency Table:
This table can be considered to be in a form of matrix where the rows are the switches while the ports are the columns. The corresponding switches to which the ports are connected are the values.
Below is the format and example adjacency table.

{switch :{ports:switch}
{1: {2: 2, 3: 3, 4: 4},
2: {2: 1, 3: 3, 4: 4},
3: {2: 1, 3: 2, 4: 4},
4: {2: 1, 3: 2, 4: 3}}

Port Statistics Table:
This dictionary is used to store the link parameter values required for cost calculation. The key is the dpid of the switches while the values are also dictionaries with ports as keys and corresponding parameters as values.
Below is the format and example of the port statistics table.

ports = {'**switch**': { **port** : [latency, bandwidth_util, prev_tx_drops, tx_drops, mac-address]}}

{'**00-00-00-00-00-01**':
{**1:** [0.0, 100, 0, 0, '5e:2d:92:32:01:42'],
2: [0.0, 100, 0, 0, '22:bd:d6:2e:19:01'],
3: [0.0, 100, 0, 0, 'ba:66:fb:bc:d8:03'],
4: [0.0, 100, 0, 0, '6e:59:d1:20:39:c3'],
65534: [0.0, 100, 0, 0, 'f6:88:35:90:eb:43']}

3 Implementation

3.1 Platforms for Project

OpenFlow: It is a Software Defined Networking (SDN) paradigm that provides centralized control functions by decoupling control and data forwarding layers of routing.

Mininet: Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine.

Open vSwitch: Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols.

POX: It is an OpenFlow controller development framework in Python. It provides various OpenFlow APIs that can be used to interact with switches, fetch statistical information from the switches, and write control information to the switches.

Python: Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

D-ITG: Distributed Internet Traffic Generator is a platform capable to produce traffic at packet level.

3.2 Specifications

Hosts	OS - Ubuntu Memory - 3GB D-ITG 2.8.1
Switches	OS - Ubuntu Memory - 3GB Open vSwitch 2.3.2
Controller	OS - Ubuntu Memory - 3GB POX Controller 0.3.0 (dart)

3.3 Installation Instructions

3.3.1 Mininet

1. Run the following commands from Ubuntu terminal

```
cd ~
mkdir mininet
cd mininet
git clone git://github.com/mininet/mininet .
cd util/
./install.sh -a
```

3.3.2 POX Controller

1. Extract the `QoSOptRouting.zip` into a new Folder "`QoSOptRouting`". POX Controller is ready to run.

3.3.2 D-ITG

1. Run the following command from the folder extracted above "`QoSOptRouting`"

```
sudo sh D-ITGInstall.sh
```

4. Testing and Validation

4.1 Test Environment Details

Figure 2 shows the network topology we are using for the test cases.
Table 1 shows the DPID's of the switches
Table 2 shows the hosts and their ip addresses.

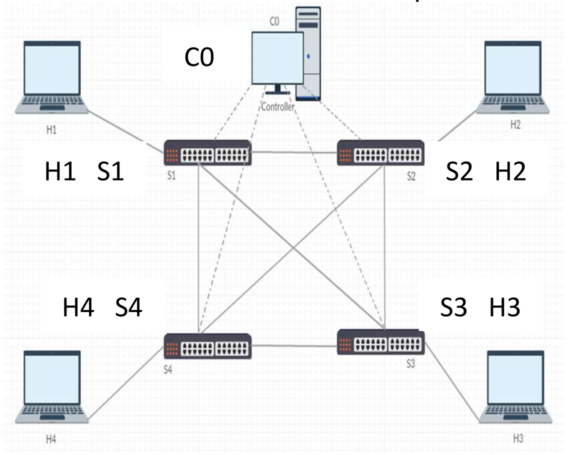

Figure 2: Topology Diagram

Switch	DPID(String)
Switch-1	00-00-00-00-00-01
Switch-2	00-00-00-00-00-02
Switch-3	00-00-00-00-00-03
Switch-4	00-00-00-00-00-04

Table 1: Switches

Hosts	IP Address
Host-1	10.0.0.1
Host-2	10.0.0.2
Host-3	10.0.0.2
Host-4	10.0.0.3

Table 2: Hosts

4.2 Unit Testing

Test Case 4.2.1- Switch Connectivity

1. Start POX controller module on the controller node using the command "`QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command "`sudo mn --custom`

`path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Open Terminal on a Switch using the command “`xterm s2`” on the mininet console.
4. Verify switches are connected to the controller using “`ovs-vsctl show`” on the switch terminal.
5. Verify controller log to verify it’s connected to the switches.

Switch :
# ovs-vsctl show

```
03defa20-a00b-494d-9bda-95804c3ea877
   Bridge "s2"
       Controller "ptcp:6635"
       Controller "tcp:127.0.0.1:6633"
           is_connected: true
       fail_mode: secure
       Port "s2-eth3"
           Interface "s2-eth3"
       Port "s2-eth4"
           Interface "s2-eth4"
       Port "s2"
           Interface "s2"
               type: internal
       Port "s2-eth1"
           Interface "s2-eth1"
       Port "s2-eth2"
        Interface "s2-eth2"
```

Controller:

```
POX 0.3.0 (dart) / Copyright 2011-2014
James McCauley, et al.
QoS Based Optimal Routing using OpenFlow
Connected to switch 00-00-00-00-00-04
Connected to switch 00-00-00-00-00-03
Connected to switch 00-00-00-00-00-02
Connected to switch 00-00-00-00-00-01
```

Hence it is observed that the switch is connected to the controller.

Test Case 4.2.2 - Controller Handling Packets

1. Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Use ping utility to send packets between hosts on the mininet console using the command “`h1 ping h2 -c 4`”.
4. Check the switch logs for successful ICMP replies.

Switch :

```
mininet> h1 ping h2 -c 4
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=2000 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64
time=2000 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64
time=2000 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64
time=2000 ms
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0%
packet loss, time 3019ms
rtt min/avg/max/mdev =
2000.180/2000.326/2000.700/0.216 ms, pipe
2
```

Hence it is observed that the controller is responding to the ICMP packets.

Test Case 4.2.3 - Switch Adjacency Table

1. Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Controller runs discovery on connecting to the switches.
4. Verify that the adjacency matrix is correctly populated.

Adjacency Table Syntax
{Switch : {Port : Switch}}

```
{1: {2: 2, 3: 3, 4: 4},
2: {2: 1, 3: 3, 4: 4},
3: {2: 1, 3: 2, 4: 4},
4: {2: 1, 3: 2, 4: 3}}
```

Hence we can see that the switches are connected to each other as per our topology.

Test Case 4.2.4 – Probing Unit Calculation

1. Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Check that the port_stats table is being populated with the calculated statistics on the controller output.

Statistics Table Syntax

{'**switch**':
 { **port** : [latency, bandwidth_util, prev_tx_drops, tx_drops, mac-address]}}

Initial Statistics Table (For Switch 1)

```
{'00-00-00-00-00-01':
{1: [0.0, 100, 0, 0, 'de:ab:18:79:59:05'],
2: [0.0, 100, 0, 0, '4e:c6:49:a4:9f:86'],
3: [0.0, 100, 0, 0, 'ba:f7:75:c8:0b:37'],
4: [0.0, 100, 0, 0, 'fe:88:5d:76:7f:8c'],
65534:[0.0,100,0, 0,'56:be:9b:82:37:4a']}}
```

After Receiving Statistics:

```
{'00-00-00-00-00-01':
{1: [0.0, 8, 0, 0, 'de:ab:18:79:59:05'],
2:[1061.4784,5,0,0,'4e:c6:49:a4:9f:86'],
3:[48.62, 6, 0, 0, 'ba:f7:75:c8:0b:37'],
4:[48.0624, 7, 0, 0, 'fe:88:5d:76:7f:8c'],
65534:[0.0, 0,77,77,'56:be:9b:82:37:4a']}}
```

Hence it is observed that the probing unit is calculating the various statistics for switches.

Test Case 4.2.5 - Link Down Event

1.Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Run the command “`link s1 s2 down`” on the mininet console to take down the link between s1 and s2.
4. Verify in the controller output that the link is being removed.

Switch

```
mininet> link s1 s2 down
```

Controller

```
Link removed 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
Link removed 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
```

Hence it is observed that the controller is able to handle Link Down events.

Test Case 4.2.6 - Cost Function Calculation Verification

1. Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Verify controller logs to check that the cost function matrix is being printed appropriately.

Cost Matrix Syntax

{**Switch** : {Port : Cost} }

Controller:

```
----- Traffic Received with ToS -----
160  || Traffic Type =  voice

----- Costs for given ToS -----
{1: {2: 102.7531, 3: 3.8454, 4: 3.5638},
2: {1: 103.8211, 3: 3.8072, 4: 3.5666},
3: {1: 3.8379, 2: 3.7490, 4: 2.6284},
4: {1: 3.6821, 2: 3.5927, 3: 3.7510} }
```

Hence it is observed that the Cost Function is calculating costs for all the links based on the ToS value.

Test Case 4.2.7 - Verifying Path Finder module

1. Start POX controller module on the controller node using the command " `QoSOptRouting# ./pox.py log.level --ERROR QoSOptRouting`"
2. Start mininet module using the command “`sudo mn --custom path/to/QoSOptRouting/customtopo.py --topo mytopo --link tc --controller remote`”
3. Start Terminals for Host 1 and Host 2 using “`xterm h1`” and “`xterm h2`” on the mininet console.
4. Set node h2 as receiver by running the command “`ITGRecv`” on the Host 1 terminal.
5. Set node h1 as sender and run Voice traffic using the following commands -
Best Effort: ITGSend -a 10.0.0.2 -rp 10001 -b 0
Voice: ITGSend -a 10.0.0.2 -rp 10001 -b 160
Video: ITGSend -a 10.0.0.2 -rp 10001 -b 120
Business Critical: ITGSend -a 10.0.0.2 -rp 10001 -b 88
6. Verify controller logs to check that the path is getting printed properly.
7. Start Terminal for Switch 1 using “`xterm s1`” on the mininet console.
8. Verify switch dump-flows using “`ovs-ofctl dump-flows s1`” to check that the path has been installed properly.

Path Syntax on Controller

[(Switch, Input port, Output port)]

Controller:

Voice Traffic :-

```
----- Traffic Received with ToS -----
160  || Traffic Type =  voice

----- Costs for given ToS -----
{1: {2: 103.6173, 3: 6.117, 4: 6.0949},
2: {1: 103.7688, 3: 6.0872, 4: 6.0647},
3: {1: 6.0540, 2: 5.8113, 4: 6.0350},
4: {1: 7.1174, 2: 6.8743, 3: 7.1423}}

-----Printing path-----
[(00-00-00-00-00-01, 1, 3),
(00-00-00-00-00-03, 2, 3),
(00-00-00-00-00-02, 3, 1)]
```

Best Effort Traffic:-

```
----- Traffic Received with ToS -----
0  || Traffic Type =  best_effort

----- Costs for given ToS -----
{1: {2: 600.0, 3: 600.0, 4: 500.0},
2: {1: 600.0, 3: 600.0, 4: 600.0},
3: {1: 500.0, 2: 600.0, 4: 600.0},
4: {1: 600.0, 2: 600.0, 3: 600.0}}

-----Printing path-----
[(00-00-00-00-00-01, 1, 2),
(00-00-00-00-00-02, 2, 1)]
```

Switch:
# ovs-ofctl dump-flows s1

```
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=5.518s, table=0,
n_packets=5043, n_bytes=2793822,
idle_timeout=10, idle_age=0,
priority=65535, udp, in_port=1,
vlan_tci=0x0000, dl_src=66:bf:9b:d2:82:55,
dl_dst=16:95:47:d7:4a:d4, nw_src=10.0.0.1,
nw_dst=10.0.0.2, nw_tos=160, tp_src=36177,
tp_dst=10002 actions=output:3

cookie=0x0, duration=11.608s, table=0,
n_packets=9, n_bytes=638, idle_timeout=10,
idle_age=5, priority=65535, tcp,
in_port=1, vlan_tci=0x0000,
dl_src=66:bf:9b:d2:82:55,
dl_dst=16:95:47:d7:4a:d4, nw_src=10.0.0.1,
nw_dst=10.0.0.2, nw_tos=0, tp_src=51684,
tp_dst=9000 actions=output:2
```

Hence it is observed that the Path Finder is calculating paths based on the Costs and installing it in the Switch flow table.

5 References

5.1 Reference URLs

1. OpenFlow: https://www.opennetworking.org/sdn-resources/openflow

2. Mininet: http://mininet.org/walkthrough/

3. D-ITG: http://traffic.comics.unina.it/software/ITG/manual/D-ITG-2.8.1-manual.pdf

4. Pox: http://openflow.stanford.edu/display/ONL/POX+Wiki

5. Python: https://www.python.org/

5.2 References

[1] Perros, Harry. The MPLS and DiffServ Architectures. Networking Services: QoS, Signaling, Processes. CreateSpace Independent Publishing Platform, 2014. 200-205. Print.

[2] Phemius Kevin and Mathieu Bouet. Monitoring Latency with OpenFlow. *Network and Service Management (CNSM), 2013 9th International Conference (2013)*: 122 - 125. Web.

[3] McKeown, Nick et al. OpenFlow: Enabling Innovation in Campus Networks . *SIGCOMM Compute. Commun. Rev*. Mar. 2008. Web.