DEPARTMENT OF COMPUTER SCIENCE | UNIVERSITY OF KENTUCKY

Quality of Service in Software Defined Networks

MASTERS PROJECT REPORT

SUBMITTED BY : Sumaira Shamim

ADVISOR : Dr. Zongming Fei

April 1, 2016

CONTENTS

1 INTRODUCTION

This project focuses on research and proof of concept about how OpenFlow and Software Defined Networks can be used to provide Quality of Service. The main goal of Software defined networks is to help create a smarter network then we have today. The concept of splitting the control plane and the data plane and increasing programmability of the network by taking the intelligence to a central controller allows us to make our networking entities more application aware. The advantages of a smarter network are better utilization of network resources, adaptability, lower cost and better services.

The decoupling of the control plane and data plane allows a high degree of control over flows that pass through the switches via intelligent controller applications. The increasing advent of real time applications also bring with them the need of demanding Quality of Service guarantees and the network protocols need to have the capability of meeting these requirements [1]. The quality of service evaluation of a network is to make certain that the application receives its due set of connection parameters according to the QoS requirements of that application. The parameters can be throughput, End to end delay, Jitter (deviation from average end-to-end delay) and number of packets lost or damaged in the channel [2].

We need to utilize OpenFlow protocol in SDN for providing dynamic QoS guarantees for different classes of flows with different service requirements by making dynamic routing decisions or efficient priority queuing.

2 RELATED WORK FOR IMPLEMENTING QOS IN SDN

This report summarizes the techniques that were researched for literature review on the project.

2.1 RATE LIMITING AND TRAFFIC SHAPING

The rate limiting or traffic shaping technique is achieved by bandwidth control among the OVS ports for rate limiting. Standard OpenFlow protocol specifications like "enqueue" can be used for queuing traffic based on inspection of flows by the SDN controller in order to assign different queuing QoS policies for different flows. The queues inside OVS itself are configured by an administrator of OpenFlow versions less than 3.0 while versions 3.0 and above introduce "OF-Config" that will make this process much easier [3].

Since only bandwidth guaratees and FIFO scheduling is not enough to provide QoS guarantees in SDN, using multiple packet schedulers of Linux kernel e.g., Hierarchical Token Bucket, Randonly Early detection and Stochastic Fairness queuing can be utilized to overcome packet scheduling issues. The strong traffic control system of Linux can be used to provide traffic shaping, queuing and congestion avoidance. This approach would require data path extensions for OpenFlow for kernel space queues [4]

Some QoS extensions to OpenFlow have been developed in [5] that have the capability of taking in high level QoS requirements of applications and automatically modify the QoS parameters on network devices in the form of rate limiters and dynamic priority assignment. This controller removes the need of manually configuring QoS requirements for each device in the network and centrally controls these configurations.

2.2 Differentiated Services Code Point (DSCP)

This technique is established by using the 8 Type of Service bits in existing IP header for implementing differentiating class of service. Standard OpenFlow protocol specifications allow rewriting of the Type of Service field in the IP header using "network ToS". This is achieved by the SDN controller after flow matching and classification according to different ToS polices for different flows [3].

2.3 Multipath routing

The idea of multipath routing is to split and balance a flow among a set of alternative paths using a Multipath agent which splits tcp session into multiple virtual sessions at the end host by starting internal sockets. The controller can intercept the first packet of each new connection and find out the application it belongs to by inspecting the payload and make sure the flows with the same connection take different paths. All the flows are multiplexed together via a multiflow agent at the destination host. A speciating routing module in the controller keeps track of the subflows, QoS policies and dynamic calculation of paths which serve the best QoS to the flows [2].

2.4 Dynamic routing of specific flows

This approach is designed for special QoS flows like multimedia flows where timely delivery is preferred over reliability usually without affecting other types of traffic. The QoS flows can be dynamically routed on different paths while the rest of the data flows remain on the shortest optimal path for them. The traffic can be differentiated by Traffic class header field in MPLS, TOS (Type of Service) field of IPv4 header, Traffic class field in IPv6 header, source IP address of a known multimedia server and TCP port numbers. Collection of current global network state information like delay, bandwidth and rate of packet loss is also required for dynamic routing [6].

Another approach for dynamic routing is by using two level QoS flows for multimedia like MPEG-4 which encodes videos in a base layer and enhancement layers. The controller identifies the level 1 QoS (base layer), level 2 QoS (enhancement layers) and normal traffic flows and updates the switches with calculated dynamic routes for level 1 QoS and level 2 QoS flows after estimating bandwidth and delays on the links [7].

Another technique in [1] tries to find an optimal path between two end points with the

minimum cost (that does not increase the capacity of any link) after ruling out the links that break specified QoS constraints.(i-e delay and packet loss do not increase a threshold).The architecture dynamically updates the network parameters to become aware of the status of the network and calculate the best route according to the QoS specified policies. The model is based on "multi-commodity flow constrained shortest path problem (MCFSP)" .

3 MOTIVATION FOR THE PROJECT

The project aims for setting up an experiment for implementing dynamic Quality of Service in software defined networks using GENI portal as the platform. The project serves to be a proof of concept on how the behavior of individual network flows can be changed from the default shortest forwarding path through Northbound API applications running on top of the SDN controller.

4 PROJECT DESCRIPTION

4.1 EXPERIMENT SETUP

The experiment is set up in GENI portal using Xen virtual machines as hosts and Open vSwitch nodes as virtual switches between them. The switches are configured with default shortest path routing rules for each of the hosts using an SDN controller. The SDN controller used is Floodlight open source controller which exposes a REST API for the northbound applications to use. The northbound QoS application for the experiment is developed in Python. It uses python packages like Networkx and untangle in order to construct the topology using the information provided by the controller REST API and RSpec document provided by the GENI portal.

4.2 THE NETWORK TOPOLOGY AND THE CONTROLLER

The topology constructed in GENI portal is shown in Figure 4.1. The topology is not linear and has multiple paths between a pair of hosts in order to have alternate paths for routing flows based on bandwidth. The nodes labeled as switches run Openvswitch on them configured to use an OVS bridge to connect to the hosts and other OVS nodes.

Each of the OpenFlow switches (OVS) have a Datapath id (DPID) associated with them which is used by the SDN controller to identify and control the switches. Using the web interface of the Floodlight controller, we can see the switches connected to the controller and the various information statistics related to them in Figure 4.2.

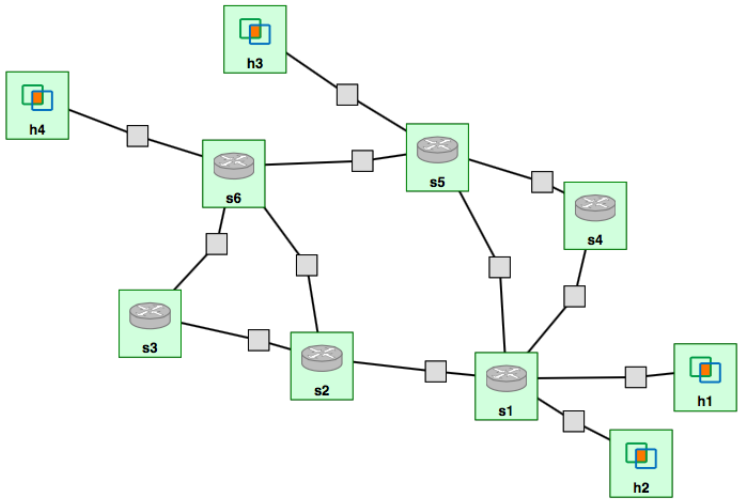

Figure 4.1: GENI topology

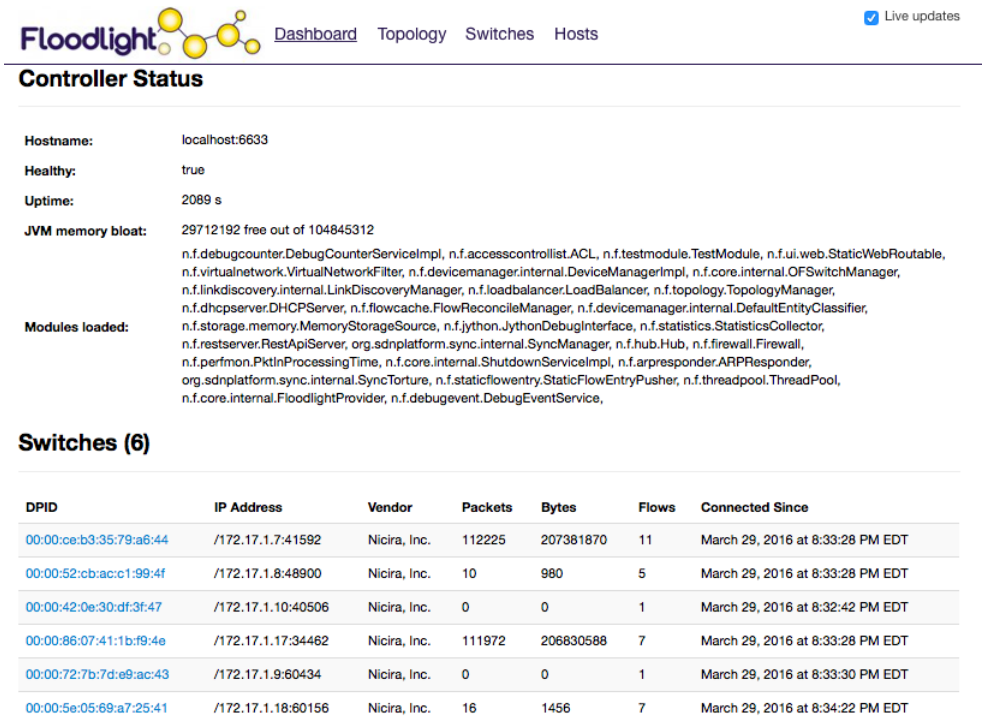

Figure 4.2: Floodlight dashboard

We can also see the OpenFlow rules that a switch is configured with in Figure 4.3. Right

now, the rules shown are default shortest path rules with OpenFlow priority '1' that the floodlight controller pushed into the switches using the static flow pusher REST API.

Flows (5)

Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
45035997150539188	0x0	1	eth_type=0x0x800 ipv4_src=10.10.2.1 ipv4_dst=10.10.4.2	actions:set_eth_src=02:9a:66:b2:1c:cf,set_eth_dst=02:1d:aa:57:2c:c0,output=3	n/a	n/a	n/a	n/a	n/a	n/a	2	196	376	0
45035999210989796	0x0	1	eth_type=0x0x800 ipv4_src=10.10.1.1 ipv4_dst=10.10.4.2	actions:set_eth_src=02:9a:66:b2:1c:cf,set_eth_dst=02:1d:aa:57:2c:c0,output=3	n/a	n/a	n/a	n/a	n/a	n/a	3	294	376	0
45035999199097537	0x0	1	eth_type=0x0x800 ipv4_src=10.10.4.2 ipv4_dst=10.10.1.1	actions:set_eth_src=02:f6:43:92:9f:13,set_eth_dst=02:a5:02:96:8d:89,output=2	n/a	n/a	n/a	n/a	n/a	n/a	3	294	377	0
45035997632372464	0x0	1	eth_type=0x0x800 ipv4_src=10.10.4.2 ipv4_dst=10.10.2.1	actions:set_eth_src=02:f6:43:92:9f:13,set_eth_dst=02:a5:02:96:8d:89,output=2	n/a	n/a	n/a	n/a	n/a	n/a	2	196	377	0
45035996778631187	0x0	2	eth_type=0x0x806	actions:output=controller	n/a	n/a	n/a	n/a	n/a	n/a	0	0	378	0

Figure 4.3: Rule table in OpenFlow switch

4.3 FLOW OF THE APPLICATION

The control flow of the application starts from constructing the network graph, polling the OpenFlow switches using the floodlight REST API to find active flows and then uses the statistics collector module of floodlight to query link statistics. Based on the statistics received, the application chooses a different path for each active flow based on available link bandwidth.

The design decisions taken are the following:

- The topology is constructed using Python Networkx library undirected graph. The information for the nodes and the edges is received from the GENI RSpec document as well as the Floodlight controller REST API calls.
- The inter-switch links have been assigned a total bandwidth of 50 Mbps each and the switch-hosts links have the default bandwidth of 100 Mbps as configured by GENI portal.
- As a default setting of the statistics collector module of Floodlight controller, the links are polled every 10 seconds for updated information. No settings of the module have been changed in this experiment.
- The different simple paths are only on an end-to-end basis for a pair of hosts. A simple path means that no node is repeated in a path between two hosts.
- A switch can only decide to change the shortest default path for a flow if the sender's IP is on the same subnet as the switch's interface.
- The active flows with priority '1' are decided by polling the switches every 10 seconds and comparing the last packets hit count with the new polled information.

- The application selects a different path based on aggregate available bandwidth from a list of unshared / unused simple paths between the source and the destination.
- The application tries to select the next available shortest path between the source and destination whose available bandwidth is greater or equal to atleast 95% of another alternate longer path.
- The aggregate available bandwidth for a path is actually the minimum available link bandwidth on the set of inter-switch links on that path.
- If a better path is not available, the flow is kept on the shortest default path.
- The QoS rule written by the controller in the switches for a particular flow has an OpenFlow priority of '3' which makes sure that it receives precedence to any default shortest path rules for the same flow with priority '1'. Once a different path is selected for a flow, it is not changed again until the application quits.

4.3.1 CONSTRUCTING THE TOPOLOGY

The application reads in the RSpec XML document from GENI portal and parses information about the names and interfaces of the nodes. The Python library used for parsing is "untangle" which converts XML to python objects. The node data items consist of the IP address, MAC address and port number for each of the interfaces connected to that node. The application then queries the controller using REST API commands to get respective OpenFlow Datapath ids (DPIDs) for the switches in the topology. Following command is used to get the DPID list from the controller:

"curl -s http://<controllerIp>/wm/core/controller/switches/json"

To match a DPID received from the controller to the node name in GENI, the following REST API command is used:

"curl -s http://<controllerIp>/wm/core/switch/all/desc/json"

This command returns features of all the switches and the field 'datapathDescription' gives information to match the DPID to the node name from GENI.Similarly, the RSpec does not give information about the switch ports for the interfaces and the information about port numbers has to be queried from the controller using the following command:

"curl -s http://<controllerIp>/wm/core/switch/all/features/json"

For constructing the network graph after parsing the RSpec and querying the controller for the desired information, Python library "Networkx" is used which provides data structures for constructing network graphs. The library creates a Network object to which nodes and edges can be added. The nodes have the data items for interfaces as described above as well as the DPID and port numbers matched to the nodes using the controller information.

Since the RSpec document or the controller do not provide information about the neighbors of the nodes or which interface is connected to which neighbor, the inter-host links are identified by matching the respective IP addresses in order to construct the edges between the nodes for the graph.

4.3.2 IDENTIFYING HOSTS CONNECTED TO THE SWITCHES

As mentioned in the design decisions, A switch can only decide to change the path for a flow if the sender's IP is on the same subnet as the switch's interface. To implement this, a structure 'switchHostsFlows' is constructed which specifies which hosts are on the switch subnet and which are the interfaces that connect to another switch instead of a host. It also specifies the possible source and destination pairs of a switch's traffic after polling all the shortest path flows in the OpenFlow table of that switch. This structure also maintains the flow statistics for each pair of source and destination hosts in the form of the last packet count. This information is later used to determine active flows in the network.

The hosts connected to the switch are identified by comparing the network address of interfaces of the hosts to the /24 network prefix of the interfaces of the particular switch. This structure only references the nodes which play the role of "hosts" on the network and not the ones which play the role of "switches".

4.3.3 ENABLING STATISTICS COLLECTION

In order to take advantage of the Floodlight controller statistics module REST API, it needs to be enabled in all the OpenFlow switches in the network. The command used to enable the statistics collection in the application is:

"curl -X POST -d ' ' http://< controllerIp>/wm/statistics/config/enable/json"

After statistics are enabled in the switches, the application can query the controller using the REST API commands specifying the switch using OpenFlow DPID and the desired port using the following command:

"curl http://<controllerIp>/wm/statistics/bandwidth/<switchDPID>/<port>/json"

The reply to the above command returns the transmission and reception bandwidth per switch per port in bits per second. The default interval when the bandwidth is reassessed by the controller is every 10 seconds [8].

4.3.4 IDENTIFYING INTER-SWITCH LINKS

The application also creates a list of links between the switches and the respective output switch ports for those links called the "interswitchLinks". This is important because the bandwidth calculation for paths is done solely on the basis of available bandwidth on the switch-switch links and not the switch-host links.

4.3.5 DETERMINE ACTIVE FLOWS

As described in section 4.3.2, the structure "switchHostsFlows" keeps track of the last known packet count for a particular flow from a host to a destination. The application queries the controller to poll the switches for a list of flow statistics every 10 seconds using the command:

"curl -s http://<controllerIp>/wm/core/switch/<switchDPID>'/flow/json"

For each shortest default path flow rule returned, the application checks if the source IP is from the same subnet as an interface from the switch DPID being polled by comparing to the list of switch hosts in 'switchHostsFlows'. If determined to be the gateway switch for that flow, the new packet count is compared to the stored last known packet count to determine if the flow is active. If a shortest default path flow rule is found active, the function to calculate bandwidth for all alternate paths is called.

4.3.6 CALCULATE AVAILABLE BANDWIDTH FOR ALTERNATE PATHS

The "calculate bandwidth for paths function" receives two arguments; a source and a target for an active flow on its default shortest path. The function calculates all simple paths between the source and the target using "all_simple_paths" method of the Python Networkx library. A simple path means that no node is repeated in a path between two hosts. If there is only one simple path returned, no alternate path is present between the two hosts i.e. only the shortest default path is available and the program returns to polling other active shortest default path flows.

Using the information from the "interswitchLinks", the function uses the curl command from section 4.3.3 to get the transmission (TX) and reception bandwidth (RX) per switch per inter-switch port in bits per second. The TX and RX values are added to find the bandwidth usage in bits per second and then subtracted from the actual known link bandwidth to get available link bandwidth for each switch port and stored in a structure for using in path selection.

GLOBAL QOS RULES STRUCTURE In order to keep track of all the QoS flows of priority '3' added in the topology, a global list "QoSFlowsList" is maintained. This structure has 4 fields per rule entry; a source, a target, the complete path and the inter-switch links involved. This structure aids in the decision of selecting an unused / unshared path between different nodes in order to achieve maximum throughput when competing traffic is targeted from different sources to the same destination. The function also checks this list to ensure that a QoS flow for a source and destination pair does not already exist. The major functions of this list are:

- If the list is empty at the time application detects the very first active flow, the shortest default path is added as a QoS rule for that flow since all the links have the same available bandwidth. This path is stored in the global list so no other flow will be assigned this path.

- The second purpose is to check if a rule already exists in the global list for a flow so that there is no need to make further path changes for a flow.
- The third role of the list is to check if for a given source-target, a reverse path "target-source" exists in the list. If yes, then the reverse path is added to the list and also sent to the rule generation function.
- The final purpose of this list is to find best possible unused or unshared inter-switch link paths for each incoming active flow based on length of path and aggregate available bandwidth.

For each simple path calculated for a source-target pair, the function decides which switches and inter-switch ports are involved on that path. Using the stored bandwidth statistics, it calculates the available bandwidth for each path by finding the minimum available bandwidth for a link involved in that path.

After calculating available bandwidths on all paths, the function checks if the paths are valid unused or unshared paths by checking the global QoS flows list. Then the function checks if any of the bandwidth values of the valid paths are better than the default shortest path. If yes, then it chooses the next shortest path with the best available value, adds it to QoS flows list and returns the path. If not, then the function returns none since there is no path available that will guarantee good bandwidth without compromising other competing traffic and the flow remains on shortest path.

4.3.7 GENERATE RULE FOR PATH

The "generate rule for path" function is called if the "calculate bandwidth for paths function" returns an alternate path for a pair of hosts. The function takes in three arguments; a path which is returned by the bandwidth function, a source IP and a destination IP. The function first checks the OpenFlow rule table of the switches involved in that path for a match of a priority '3' rule for the same source and destination IP using the following command.

"curl -s http://<controllerIp>/wm/staticflowpusher/list/<switchDPID>/json"

If a rule already exists, the function returns and does not generate a new priority '3' rule. If no rule exists, the function determines the node and the next hop interface for each node in the path, finds the MAC address and switch port for forwarding and constructs a flow rule in JSON format to send via the static flow pusher REST API of Floodlight controller. A sample flow rule is shown in Figure 4.4. The command used by the application to push a generated flow through the controller to the switch is:

"curl -X POST -d '"+<JSON_rule>+"' http://<controllerIp>/wm/staticflowpusher/json"

```
sumairas@pcvm5-23:~$ curl -s http://150.182.135.41:8080/wm/staticflowpusher/list/00:00:4a:fd:b7:b9:b9:47/json | python -m json.tool
{
    "00:00:4a:fd:b7:b9:b9:47": [
        {
            "00974493ffdc04b709032bbe39d06824": {
                "actions": {
                    "actions": "set_eth_src=02:02:3a:c0:fe:78,set_eth_dst=02:83:49:6c:dc:79,output=1"
                },
                "command": "ADD",
                "cookie": "45035996768327880",
                "hardTimeoutSec": "0",
                "idleTimeoutSec": "0",
                "match": {
                    "eth_type": "0x0x800",
                    "ipv4_dst": "10.10.9.1",
                    "ipv4_src": "10.10.11.2"
                },
                "outPort": "any",
                "priority": "1",
                "version": "OF_10"
            }
        },
```

Figure 4.4: A sample flow

5 Results (Preliminary)

To test the effectiveness of the applicaton, we need to test throughput achieved with competing traffic with and without the application running. We will test this by sending traffic from host 1 (h1) to host 3 (h3) and host 2 (h2) to host 3 (h3) at the same time. Figure 5.1 shows the shortest path setup from h1-h3 and h2-h3 without the application running.

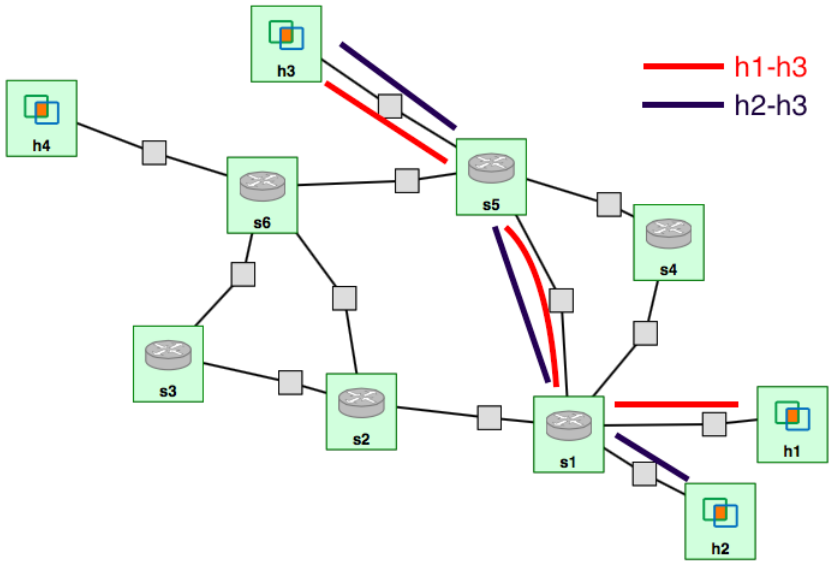

Figure 5.1: Shortest paths from h1,h2 to h3

After running the application, we run "iperf" from h1 to h3 and since this is the first active flow, the shortest path is added as a QoS flow for h1-h3 and also a reverse path for h3-h1 as shown in Figure 5.2. Next, we run "iperf" from h2 to h3. The application discovers the active flow, queries for link bandwidth and chooses the next shortest and unused path with the better bandwidth for h2-h3 as shown in Figure 5.3.

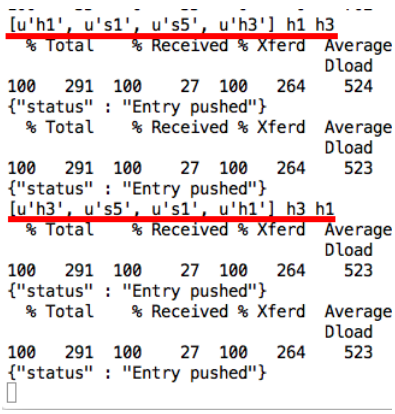

Figure 5.2: Application output for QoS Flow from h1-h3 and h3-h1

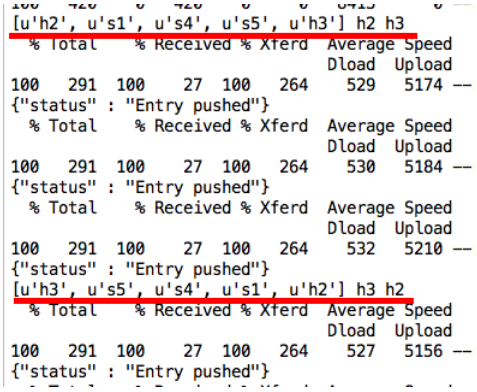

Figure 5.3: Application output for QoS Flow from h2-h3 and h3-h2

Figure 5.4 shows the final set up for the paths for h1-h3 and h2-h3 with the application running.

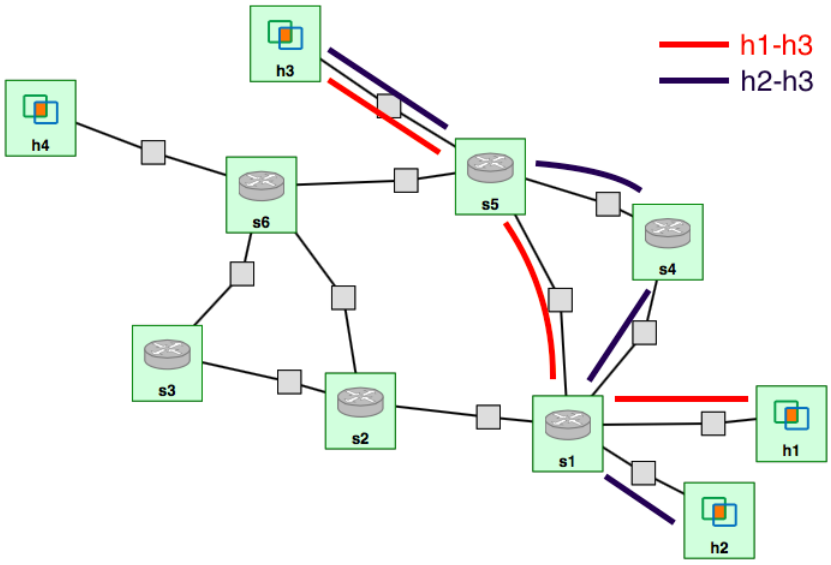

Figure 5.4: New QoS paths from h1,h2 to h3

To test the throughput, we will "iperf" from h1-h3 and h2-h3 at the same time. The maximum link bandwidth that can be achieved is 50 Mbps. Figure 5.5 shows the results achieved without the QoS application running and Figure 5.6 shows the results achieved with the QoS application running using the paths shows in 5.4.

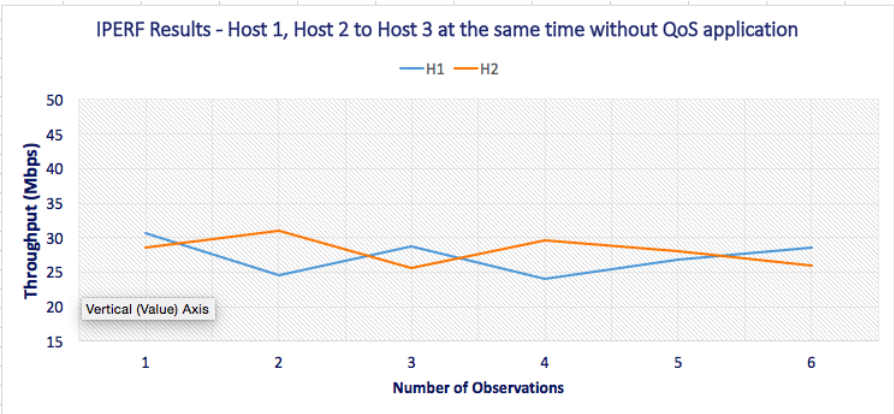

Figure 5.5: Iperf results from h1,h2 to h3 without application

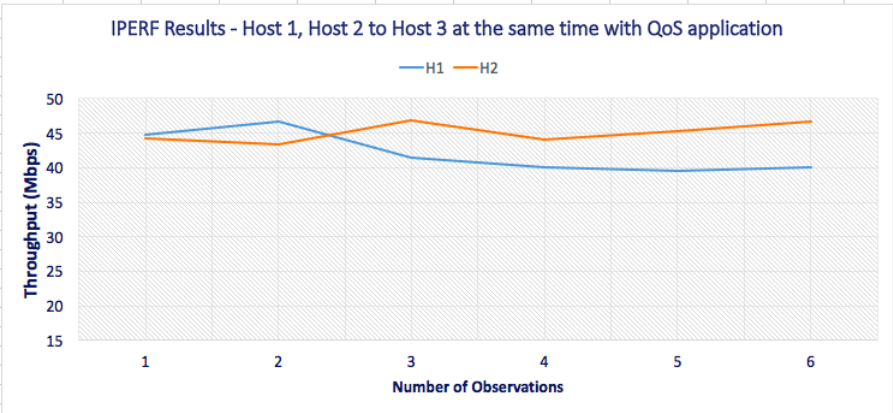

Figure 5.6: Iperf results from h1,h2 to h3 with application

It is clear that the QoS application paths achieve near link bandwidth in case of competing traffic targeted at the same destination.

6 CHALLENGES

There were a few challenges that were faced during the experiment setup.

FLOODLIGHT DEFAULT L2 LEARNING SWITCH: The default modules of Floodlight controller do not have the capability to perform shortest path routing between hosts on different subnets. There is no ARP module that will take care of the ARP requests if we use a simple layer 2 learning switch module in the controller. Therefore, some advanced features of GENI Desktop were used to initialize all the OVS nodes with shortest path rules that modify the Ethernet header after sending all the ARP traffic to the controller. This also requires that the QoS Flow rule action field built by the application must contain clauses which modify the Ethernet header with source and destination MAC address at each hop for each source-destination host IP pair.

BANDWIDTH DESIGN DECISION There is also a design decision involved which is the time interval at which the controller statistics module reassesses the bandwidth at the links. The default time used by the controller is 10 seconds and the application uses the default setting. According to Floodlight, the statistics provide a ballpark figure. OpenFlow does not provide any statistics which have the notion of time and the controller needs to calculate bandwidth using raw byte counters at two instances in time and return the value if requested using the statistics module. The 10 second time serves as a tradeoff between getting less error prone bandwidth calculation (to make control plane latency insignificant) and overburdening the switches with statistics requests by the controller to get real time bandwidth [8].

DYNAMIC PATH CHANGING: The initial experiment setup changed the path for each flow dynamically based on link bandwidth every 10 seconds if a better path was available. This was not scalable since changing the path continuously does incur overhead at both the northbound and southbound interfaces. Also, it mirrored the same problem since the beginning of internet that all flows tend to choose the same better path with greater bandwidth at one instance in time which congests that path and leave the other alternate paths with more capacity but unused. Since the bandwidth stats are reassessed by the controller every 10 seconds, even if we query the stats every time when a flow is active does not guarantee that the seemingly best path in terms of bandwidth was not assigned to another flow in the same 10 second time period.

7 CONCLUSION

This project serves to be a proof of concept of how we can use intelligent controller applications to change the default behavior of the network. For dynamic routing based on bandwidth, more work needs to be done to tackle the challenges mentioned in Section 6. Maybe if OpenFlow protocols introduces some standards for adding byte counters with time notion, the overhead and latency incurred at the controller for calculating bandwidth by keeping track of byte counters at two time instances can be avoided and more real time statistics can be sent through the REST API to applications.

This experiment fixes a path for different flows as they arrive if a guaranteed better bandwidth path can be provided to them. This concept can be used to target better service for a higher class of flows either coming from a priority server distinguished by the server IP address, Multiprotocol Label Switching (MPLS) traffic or multimedia traffic from a known server while other flows can remain on the shortest best effort paths. It is shown in the results that the throughput achieved by using the application was almost at the link capacity when competing traffic was targeted to the same destination as compared to 50% of link capacity achieved by just shortest path.

REFERENCES

[1] F. Ongaro, "Enhancing quality of service in software defined networks," Master's thesis, University Of Bologna, 2014.

[2] E. Chemeritskiy and R. Smelansky, "On qos management in sdn by multipath routing," in *Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International*, pp. 1–6, Oct 2014.

[3] R. Wallner and R. Cannistra, "An sdn approach: quality of service using big switchâĂŹs floodlight open-source controller," *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, pp. 14–19, 2013.

[4] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "Control of multiple packet schedulers for improving qos on openflow/sdn networking," in *Proceedings of the 2013 Second European Workshop on Software Defined Networks*, EWSDN '13, (Washington, DC, USA), pp. 81–86, IEEE Computer Society, 2013.

[5] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2010.

[6] H. Egilmez, S. Dane, K. Bagci, and A. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pp. 1–8, Dec 2012.

[7] H. Egilmez, S. Civanlar, and A. Tekalp, "An optimization framework for qos-enabled adaptive video streaming over openflow networks," *Multimedia, IEEE Transactions on*, vol. 15, pp. 710–715, April 2013.

[8] R. Izard, "How to collect switch statistics (and compute bandwidth)." https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=21856267.