Implementation of measurement-based QoS mechanisms

Alperen Gündogan, Ekin Budak, Mustafa Selman Akinci and Sarfaraz Habib
Fakultät für Elektrotechnik und Informationstechnik, Technische Universität München
München, Germany
Email: gundoganalperen1@gmail.com, budakekin@gmail.com, selman.akinci@outlook.com, sarfarazh4@gmail.com

***Abstract*—The Idea behind Software defined Networking is to separate control plane from data forwarding device using protocols such as OpenFlow. This approach give network a lot of gains and global network awareness is one of such advantages. In this project, we are implementing QoS application using SDN approach. This QoS application enables us to optimize traffic dynamically over a network while sustaining reliable communication. We compared the results of our application with a non-QoS application. Results show that QoS delivers more desired values than non-QoS applications.**

***Index Terms*—Software-Defined Networking, RYU, QoS, Topology discovery**

I. Introduction

The lab report presents the implementation of measurement based QoS mechanism. The sensitive application that requires low latency is rerouted based on the live network stats to satisfy their QoS requirements. We used a self-balancing robot called Homer in order to demonstrate the sensitive application whose path is monitored continuously and re-routing is performed in case of congestion. The report also demonstrates a comparison of results using mininet emulator.

II. Methodology

We are inspired by the article named OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks [1]. This part is the brief explanation of the methods and ideas presented in this paper which we have implemented on our design.

Our project is an SDN [2] controller design that enables QoS for multimedia delivery over OpenFlow networks. In order to support QoS, it categorizes the incoming traffic as data flows and special flows, where the special flows are dynamically routed to support QoS and the data flows routed considering the shortest-path.

The SDN controller has several key concepts:

- Topology management: Includes discovering the topology by using NetworkX [3] and detecting the changes on it.
- Route management: Idea of periodically collecting network state and calculating the congestion parameter on the links for route calculation. SDN controller also decides whether a link is congested or not.
- Route calculation: Depending on the up-to-date topology and congestion on the links, both QoS and shortest path routes must be calculated.

Our design can act differently for different flows so that it can differentiate data and multimedia traffic using different packet header files or values such as traffic class header field in MPLS, TOS (Type of Service) field of IPv4, traffic class field in IPv6, source IP address(if multimedia server is known) or transport source and/or destination port numbers. Using lower layer (L2, L3) packet headers reduce the complexity of packet parsing. For this reason, we have used source and destination mac addresses to differentiate special flows.

In order to characterize the network conditions and support QoS requirements, a cost metric and a constraint is selected. The calculated route should have the minimum of the total cost while keeping total delay under a predetermined value. Thus QoS routing becomes a Constrained Shortest Path (CSP) problem. However, since maximum delay cannot exceed the number of hops, in our design delay constraint doesn't cause many problems. Thus, CSP problem becomes the shortest path problem with cost parameters as weights on the links. Typical QoS indicators are packet loss, bandwidth, delay and delay variation (jitter). Since bandwidth is a more direct indicator of congestion, we have preferred this parameter for our implementation. In case of a change in topology or congestion state of the links, route calculation is ignited and SDN controller updates the switches' flow tables accordingly. Hence, the QoS routes are dynamically set.

III. Algorithm

The flowchart in Figure 1, explains the details of our algorithm. SDN controller sends stats request to the switches in the topology to update all the edge attributes in the NetworkX graph. When OFPT_Packet_IN message is received from the switch, controller calculates a path depending on source and destination MAC addresses of traffic. If it is Homer traffic, then controller uses link load as a metric, otherwise just calculates the shortest path. There is another function that checks congestion on the links which starts to run when the Homer traffic is started. If there is a congested link on the Homer path, then controller basically deletes all the flow entries of the Homer. Therefore, switches have to send OFPT_Packet_IN message again when the Homer traffic has reached to them, then a new path which has less load is calculated with the help of NetworkX and flow entries are installed to the switches on the new calculated path. Also, Homer traffic path will be

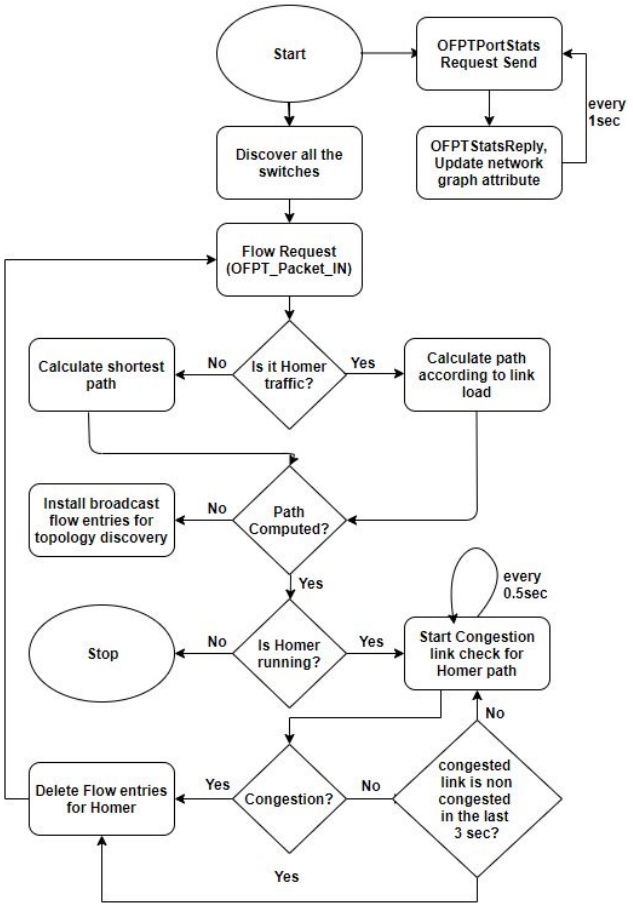

Fig. 1. Flowchart Explaining the Pseudo Algorithm of our Aplication.

changed again when previously congested link becomes non-congested in the last three seconds.

IV. Implementation

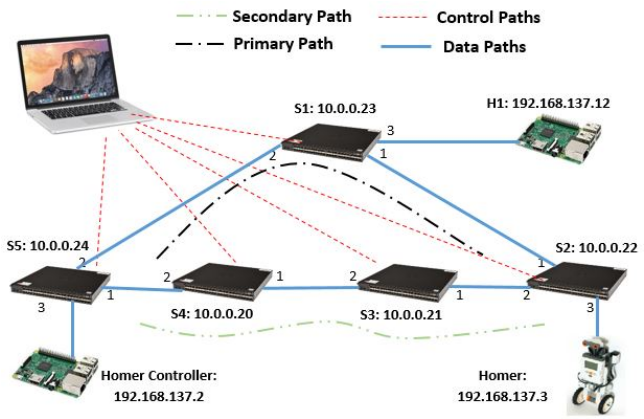

Fig. 2. Topology Setup of Our Application.

We deployed the topology with five OpenFlow enabled Zodiac FX [4] switches as depicted in Figure 2. Our aim was maintaining end-to-end QoS between Homer and Homer Controller despite the introduced load on the link in between. In order to achieve this load, we have used Raspberry Pi as host and created UDP/TCP traffic with IPerf between this host and Homer Controller which should be large enough to cause Homer to fall. Homer has two paths to contact its controller. Initially, when there is no congestion in the network, Homer and Homer Controller takes the route through S1 based on the shortest path. As traffic is created on the link between S1 and S5, this triggers the path between Homer and Homer Controller to change towards the secondary path, on which there is no congestion. After three periods of non-congested state for the previously used links between Homer and Homer Controller, SDN controller decides to delete flow entries related to Homer and Homer Controller. Therefore, new route is calculated and QoS is supplied again using the primary path in this case.

To understand why there are limited number of topologies we can use, one must investigate the ports of the Zodiac switches. One port of the switches must be used to connect to the SDN controller and the other port has to be used to connect to the Pi, Homer or Homer Controller (our hosts). That means that we have two ports to achieve connection between switches, which also means that we can connect one switch to the maximum of two other switches. With this limitation, we have come up with a topology using five switches.

V. Results and Analysis

We have tested our QoS design first in the Mininet environment with the topology as in Figure 2. Round Trip Time(RTT) between Homer and Homer Controller is used to ensure that our QoS mechanism is working.

Required steps for the tests are the following:

- Ping all the hosts for the discovery
- Start pinging between Homer and Homer controller.
- Start UDP traffic with iperf between Pi(client) and Homer Controller(server).

The maximum throughput supported by Zodiac switch is 100Mbps [4].Therefore, we have used approximately 80Mbps UDP traffic to ensure that link congested condition is satisfied. We have also compared different time periods for congested link check in Figure 3 (e.g. 0.5s, 1s). As a result, congestion link check period should be lower than the period of Port Stats Request to detect the congestion link before losing QoS. One can observe that rerouting requires less than 1-2 seconds, to calculate a new route and install new flows into the switches. Homer started to use the primary path for its traffic as in Figure 2. After the UDP traffic between H1 and Homer, the link between them will become congested and the traffic of Homer will be rerouted to the secondary path. We have also tested our software by using Pis and followed the same test procedure as described above. We observed that path has changed and new flows were installed when the link between Homer and Homer Controller was congested.

Figure 4 consists of the comparison of the two different scenarios as the following:

- QoS enabled rerouting, and

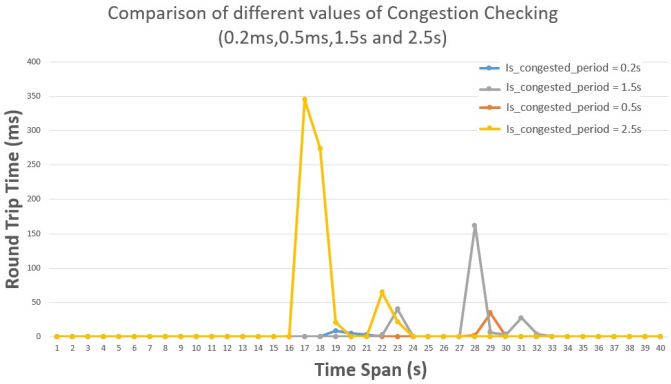

Fig. 3. Comparison of Different Values of Congestion Checking.

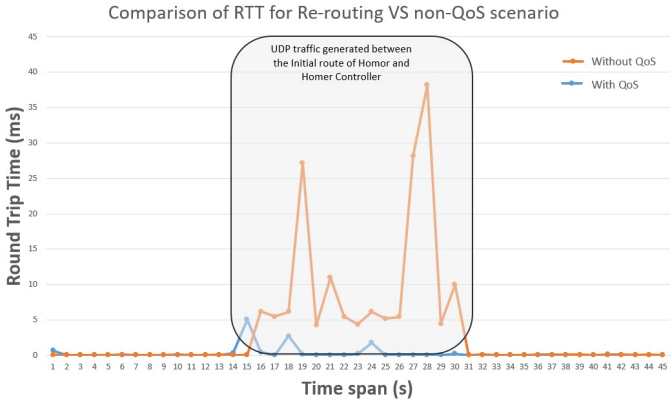

Fig. 4. Comparison of RTT for Rerouting and Non-QoS in Mininet.

- QoS disabled.

It provides the round trip times between Homer and Homer Controller, with the effect of congestion on the primary link that proofs the benefits of the rerouting. Note that these results were derived from the Mininet simulation. One can observe that round trip time is approximately 2-3 ms on average, which is much less than the non-QoS application. Figure 5 shows similar results on hardware implementation, where we have used Pi as hosts.

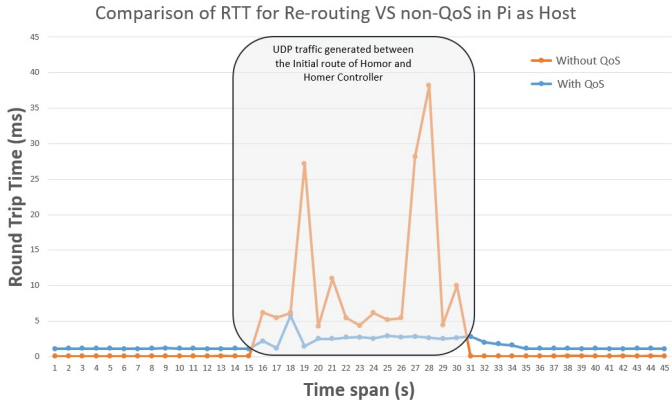

Fig. 5. Comparison of RTT for Rerouting and Non-QoS in Pi as Host.

VI. Conclusion

In this project, we have shown that our implementation satisfies the requirements of QoS application. We have observed that our chosen hosts indeed choose a different path when there is congestion in their path, and return to their original shortest path when there is not any congestion issue. To sum up, our application is an inventive way of communicating over OpenFlow networks with QoS and it has the ability of dynamic QoS routing that achieves end-to-end QoS support.

Acknowledgment

We would like to express our deepest appreciation to all those who provided their valuable input especially to Hilmi E. Egilmez, S. Tahsin Dane, K. Tolga Bagci and A. Murat Tekalp, the authors of the research paper OpenQoS, which provides the basis of this report. We also would like to express a special gratitude to our Lab Supervisors, Amaury Van Bemten, and Johannes Zerwas, whose contributions in stimulating suggestions and providing us with innovative ideas in order to deal with the bottlenecks of the project.

References

[1] Egilmez, Hilmi E., et al. “OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks.” Signal and Information processing association annual summit and conference (APSIPA ASC), 2012 Asia-Pacific, IEEE 2012.

[2] Open Networking Foundation. (2018). Open Networking Foundation is an operator led consortium leveraging SDN, NFV and Cloud technologies to transform operator networks and business models. [online] Available at: http://opennetworking.org

[3] Networkx.github.io. (2018). NetworkX — NetworkX. [online] Available at: https://networkx.github.io/

[4] Networks, N. (2018). Zodiac FX. [online] Northbound Networks. Available at: https://northboundnetworks.com/products/zodiac-fx