2017 25th Iranian Conference on Electrical Engineering (ICEE)

Prioritized Packet Loss and Delay Control in Multimedia Services over Software Defined Network

Sepehr Kazemian
Computer Engineering and Information Technology
Amirkabir University of Technology
Iran, Tehran
Email: Sepehr.kazemian@gmail.com

Siavash Khorsandi
Computer Engineering and Information Technology
Amirkabir University of Technology
Iran, Tehran
Email: khorsandi@aut.ac.ir

Abstract— The importance of the low delay in order to have a constant and continual communication is vital in the state-of-the-art world of multimedia networks. There are solutions for controlling delay and packet loss in a non-SDN environment by implementing IP-QoS. However, due to lack of a general view of network and resources, they cannot achieve real-time reconfigurability and adaptivity. Besides, they oblige intra and inter Autonomous Systems to be QoS-aware. This paper proposes an architecture to improve QoS for multimedia services by introducing a novel way to control delay and packet loss in egress ports' queues over an SDN environment by using a centralized controller. This paper is preliminarily concentrating on preventing burst packet loss in multimedia flows by prioritizing flows based on their importance which also helps to diminish packet delays. Mainly, we use multiple thresholds on our egress queues for decision on rerouting flows to their best path. The process of how to find the best path and how to choose the best threshold in order to reduce overload on the controller by adjusting controller's polling frequency.

Keywords- Software Defined Networks; Multimedia Services; QoS; Packet Prioritizing;

I. INTRODUCTION

The key SLA requirements for an IP-based video transport service can be defined by delay, jitter, and packet loss. Delays induced by a network embody four key elements: switching delay, propagation delay, queuing delay and, serialization delay. On the other hand, packet loss due to packet drops between a defined network ingress and egress points has three central causes: network congestion, physical bit errors and device failures. Bit errors might occur because of noise or attenuation in the transmission medium. In this paper, we are addressing the network congestion problem. By relieving congestion situations, we aim to control packet loss and delay simultaneously at an acceptable level for various multimedia streams.

In most multimedia applications, data is to be delivered in a streamed continuous manner. To prevent flickering in videos, deviations in audios, and blank screens in images, we need to keep delay and jitter within a tolerable boundary. For this, networks should support some levels of Quality of Service for multimedia services. The Internet Engineering Task Force (IETF) has proposed several architectures for IP QoS such as IntServ [1] and Diffserv[2]. However, none of these methods has been globally implemented [3]. They suffer from the paucity of a broad view of overall network resources. As a solution, MPLS tunneling has been proposed [4], but it has the problem of real-time reconfigurability and adaptivity. From another perspective, the Internet is divided to Autonomous Systems (AS), where each of which is owned by a single entity; as a consequence, for enabling end-to-end QoS, all intra and inter ASs should be QoS-aware. In intra-domain QoS routing, QoS extensions of OSPF has been used, however, inter-domain routing protocol BGP4 does not have any attributes to support QoS. A solution proposed for all aforementioned obstacles is somehow a brain which could be implemented easily and has a thorough view of our topology. Moreover, this central controller can provide real-time updates and reconfigurability.

The essential properties as outlined could be found in a proposed architecture by Global Environment for Network Innovations (GENI) which is based on a protocol called OpenFlow. OpenFlow represents the Software Defined Networking (SDN) [5] paradigm which decouples the control plane and data one. All the QoS behavior and routing control functions are going to be done in a centralized unit, called Controller, meanwhile, routers and switches have the duty of forwarding which we could call them Forwarders. All the decisions for these forwarders should be made by Controller via the OpenFlow protocol.

Not all the multimedia packets are the same; hence, we should distinguish them from each other based on their types. On the other hand, we may want to distinguish between our users, too. Therefore, we can use a priority basis on the analysis of packets in the controller.

Several standard and non-standard centralized or distributed control planes have been proposed previously. Dixit [6] proposed a distributed controller architecture, which is elastic, in order to enhance responsiveness of control plane and take advantages of multiple controllers based on traffic conditions. It was completely an ingenious work to distribute traffic on several controllers; however, it proposed no way to reduce the overload over controller inasmuch as it did not concern about traffic on one specific controller when it had several ones. Nonetheless, it disregarded traffic on specific controllers and did not specify a

978-1-5090-5963-8/17/$31.00 ©2017 IEEE

formula to do that. Kim et al. [7] used a DiffServ-based approach to gain QoS efficiently. However, as it was mentioned before, DiffServ has not been globally implemented, because they suffer from the paucity of a broad view of overall resources of a network. Egilmez and Tekalp [8] proposed a novel distributed QoS architecture for multimedia streaming over SDN in order to improve acquisition of network topology and state information and to enhance scalability and security of inter-domain QoS routing. Nevertheless, none of these papers differentiated between users and multimedia packets at all.

II. System Architechture

A. Schema for preventing packet drops

The first goal is prioritizing packets based on their importance. This is axiomatic that not every packet is the same. Some of them could be assumed more important than others based upon network administrator's opinion. In other words, the goal of prioritizing is to assure network's administrator of implementing quality of service (QoS). In order to do so, two steps should be taken. The first step is to assign numbers as priority to packets. The second one is assigning two thresholds on queue of each egress port. The former occurs when first packet of every flow is going to pass through controller as PACKET_IN1. The latter is elaborated later.

First of all, multimedia traffic is discovered among other kinds of network traffics by searching multimedia features in application and network layer (Figure 1). For multimedia usage, users and application layer's protocols could be differentiated. To do the former, the controller can distinguish between users based on their IP and MAC addresses. In this way controller gives them a priority number based upon what network manager prefers (as an instance, high numbers could be used as an identifier for high priority or vice versa). For the latter one, the controller can distinguish between application layer's protocols (such as RTP and SIP) based on their port numbers. It is worthwhile to note that each priority that had been given to the services are due to the very services importance.

The control plane has the duty of appointing priority numbers to flows. We enact some rules on switches when first packet of a flow goes straight to the controller for the very first time. Moreover, first packet of those flows that are not going along with any rules on switches, are going to be processed by the controller. For processing flows, at first, controller processes first packet of a flow and reads every bit of the packet. Network manager can set a pattern or the individual can read only specific fields such as port numbers, and MAC addresses, to name but a handful. At last, controller decides to set a priority number for the packet. As an example, the network manager can assign a priority number to RTP services or one for users in one of the IP address classes.

For preventing burst packet drop, as it is aforementioned before, two thresholds should be implemented on egress ports' queues. One is for upper restriction in order to realize when queues are on the verge of being completely full. The other one is substantial either, since, we do not want our queues' resources being used inefficiently. How to find the optimal numbers for upper and lower thresholds is described later. For understanding status of the queues and comparing them with our chosen thresholds, we use OFP_Queue_Stats which yielded some information (such as, number of transmitted packets and bytes) about the queue. When queue is filled more than its threshold number, the rerouting process should be started. Therefore, the goal of checking the queue is to understand how much time is remained to start rerouting process. In order to diminish the controller's overload, the most optimal periods of time, called T*, should be chosen.

Figure 1: An overview

Input: *PACKET_IN*
Output: *OpenFlow Rules*

```
1: Function PACKET_IN() {
2:    if (Multimedia)
4:        if (application_layer_protocol_distinguishing)
5:            distinguishing_port_numbers(){
6:              prioritizing packets();
7:              Set_rules_on_switch();
8:            }
9:        end if
10:       else if (user_ distinguishing)
11:           distinguishing_Mac_IP_addresses(){
12:             prioritizing packets();
13:             Set_rules_on_switch();
12:           }
13:       end else if
14:   end if
15: }
```

If B (Buffer) is considered as queue size and queue_filling_rate as the rate by which a queue is going to be filled, that is the difference between egress port's output and its input, equation 1 is yielded:

$$(1) \quad T^* = \lfloor \frac{\left(B - (Threshold)\right)}{queue\_filling\_rate} \rfloor$$

The pseudo code of what controller should do in different situations can be seen in Figure (2).

B. Rerouting Process

For rerouting, the Open Shortest Path First (OSPF) routing protocol is used with specific costs. These costs are defined based on link's delay and egress ports' queue size. Per every five milliseconds delay, we add one point to the link's cost. On the other hand, if the queue is filled less than 20% we add one point to its cost. If it is filled between 20% and 60%, per every 10% between these numbers, we add one more point to its cost and if it is filled more than 60% we add 100 points to its cost in order to not eliminate this part from a path, but to worsening the possibility of being chosen as part of the path. As an example, this is possible to have paths with egress ports' queues on its way that are filled more than 60%; hence, they should be brought into our comparing process either. However, we would rather to do

[1] Based on OpenFlow protocol

Figure 2: Overall Behavior of Controller

Input: *queue_stat, Up_Threshold, T**
Output: *rerouting*

```
1: Function Checking_Stats() {
2:    if (queue_stat > Up_Threshold)
3:        reroute_out_low_priorities();
4:        Checking_Stats() after T* seconds;
5:    end if
6:    if (queue_stat < Down_Threshold)
7:        reroute_back_low_priorities();
8:        Checking_Stats() after T* seconds;
9:    end if
10:   else()
11:       Checking_Stats() after T* seconds;
12:    end else
13:}
```

not send flows through these paths since they will be in the verge of rerouting very soon. Of course these numbers could be different in different topologies.

At last, these numbers will be aggregated. By doing this, the overall view of the topology's routes will be acquired. At this point, best possible routes into which flows will be forwarded or rerouted can be chosen.

C. Schema for delay control

For delay control, network manager should consider every egress port queue's buffer size. At first, let assume that rules will be performed on only one switch (the right one in the figure 3). The maximum buffer size (B bits) for every egress port's queue which is for multimedia services with maximum delay of α milliseconds and port's transmit rate of β bit/s should be:

(1) $B = \alpha \times 10^{+3} \times \beta$

This is because B bits will be filled when packets in α seconds (milliseconds $\times 10^{+3}$) at the rate of β bits/second go into the queue without any output rate.

On the other hand, it is obvious that every port has its own output rate which is defined by its correspondent link. Since not only multimedia services are existing in our network and mostly WFQ algorithm is being used to allot turn to ports to transmit their packets, a dynamic threshold for different ports in the switch is needed. By knowing this, a dynamic threshold for every queue based on rate of its related output link should be defined. Furthermore, on a larger scale, data should pass through many queues of many switches, which every one of them have different output and input rate, before delivering to the end point. If Q_i is considered as queue filling rate of i_{th} queue and N as number of queues in a path, the aggregation of output rates will be Y:

(2) $\sum_{i=0}^{N} Q_i = Y$

Thresholds of every queue should be different due to its different output and input rates. Therefore, it is vital to assign a threshold to the i_{th} queue. UP_Threshold$_i$ will be depended on *highest expected delay* (*HED*) for a multimedia service (for example 150ms) and opposite filling rate of queue. The reason for the latter is that, the more queue is filled, the lower the threshold should be defined in that the controller should earn more time (T*) to prevent an unruly situation that causes delay to increase. (output rate of the queue)

(3) $X_i < \frac{UP\_Threshold}{T*}$

(4) $UP_{Threshold} = \frac{(Y - Q_i) \times HED}{(Y \times (N-1))}$

Formula (3) is needed for dynamic changing of the thresholds of queues since input and output rate of every queue in every switch could be different in different network situations. Those queues which their input rates are less than their output rates will dealing with no problem in sending their data instantaneously; therefore, until other queues encounter otherwise, UP_Thresholds' of those queues won't be changed. When a queue, which its input rate is more than its output rate, is found, by changing every queue's threshold of this path, thresholds will be justified and the whole network will be alleviated. The pseudo code of what has been explained is in figure (4). (Z_i = input rate of i_{th} queue)

Situation would be different when packets are started to flow into the network for the very first time since controller is not aware of type of packets and services that is needed; therefore, the controller is not aware of input rate of the switches' queues. At this time, a question of how thresholds of

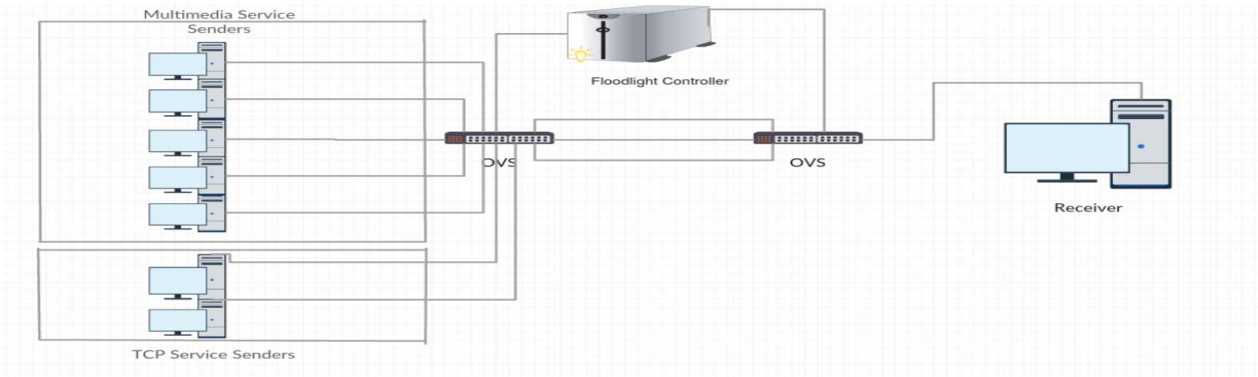

Figure 3- First Topology Outlook

Figure 4: Adjusting Dynamic Thresholds on Queues

Premises: Z_i = input rate of i_{th} queue, X_i = output rate of i_{th} queue
Output: $UP_{Threshold}$

1: **for** (i = 0; i < N; i ++)
2: **if** ($Z_i \geq X_i$)
3: $R_i = \frac{Z_i}{X_i}$
4: $S =+ R_i$
5: **end if**
6: **end for**
7: **for** (j = 0; j < N; j ++)
8: $UP_{Threshold} = \frac{(S - R_j) \times HED}{(S \times 2)}$
9: **end for**

queues should be defined, should be arisen. To overcome this situation (Q_i) should be defined for once as an input rate in formula (2) and (3).

For more precise explanation, we come up with the ongoing example: if we consider a queue in the first switch, which is in the way of flows, has the output rate of 2X bits/s. In addition, if we consider that the other switch's queue, which is in the way of multimedia flows either, has the output rate of 3X bits/s, for delay threshold of 150ms or in other words for aggregated UP_Threshold of 1500Kbits, we allot UP_Threshold of the first one 900Kbits and the second one 600Kbits.

III. Implementation

As a proof of concept, we have made our topology on Mininet [9] which is an emulator and any topology could be implemented on it by a simple python codes. It furnishes a handful features by which we could use programmable switches, get status of these switches, and improve our topology easily, to name but a few.

We had several choices to choose our controller between distributed or centralized. First of all, the controller which is used was a centralized one since if it meets our needs and works efficiently, for sure this could be done by distributed one too. The reason for this view is that our topology or our view is not depended on controller's type (distributed or centralized). As a centralized controller Floodlight which is an up to date controller and completely support OpenFlow 1.3 [10] and up is chosen. Moreover, this controller fully supports OpenVirtualSwitch (OVS) [11], which is used as an programmable switch in this project, and OpenFlowSoftwareSwitch13 (OFSoftSwitch13) [12].

Between programmable switches, we narrowed our choices to OVS and OFSoftSwitch13. Both of them are equally powerful and practical. There are some slight differences between these two switches. As an example, OVS do not support OpenFlow meters. Conversely, OFSoftwareSwitch13 completely supports OpenFlow meters. On the other hand, OVS supports upper versions of OpenFlow; nonetheless, OFSoftwareSwitch13 do not support any version of OpenFlow except 1.3. From this view that we did not need to use OF meters, and we needed OpenFlow1.4 for implementation, it was the best to use OVS.

In order to generate packets and flows in our nodes and send them toward our switches, we needed a multimedia packet generator. We used FFMpeg, a Linux-based packet generator, and we wrote a python code in this order.

For implementing traffic control in this project, we had to use Linux-HTB and TC [13], in addition to features of QoS in OVS. There were some slight problems in working with both Linux-TC and QoS implementation of OVS; nevertheless, by some subtle changes they will work fine besides each other.

IV. Evaluation

In the first phase we ran our project with 7 sender nodes on Mininet by running a python code which causes 5 of these 7 nodes send 2900K, 2600K, 1900K, 1500K, and 1300K bit per second multimedia traffics. Rest of these senders, were sending TCP packets. On the other side of this topology, there was one node which gets all of these packets. As we can see in figure (3), there were two switches in the way of our flows. The first switch in the way had three egress ports, and for each one had three queues and one of these queues was devoted to these multimedia packets, the other ones used for TCP packets. There were three links between switches which their output rate was 100mb/s. In

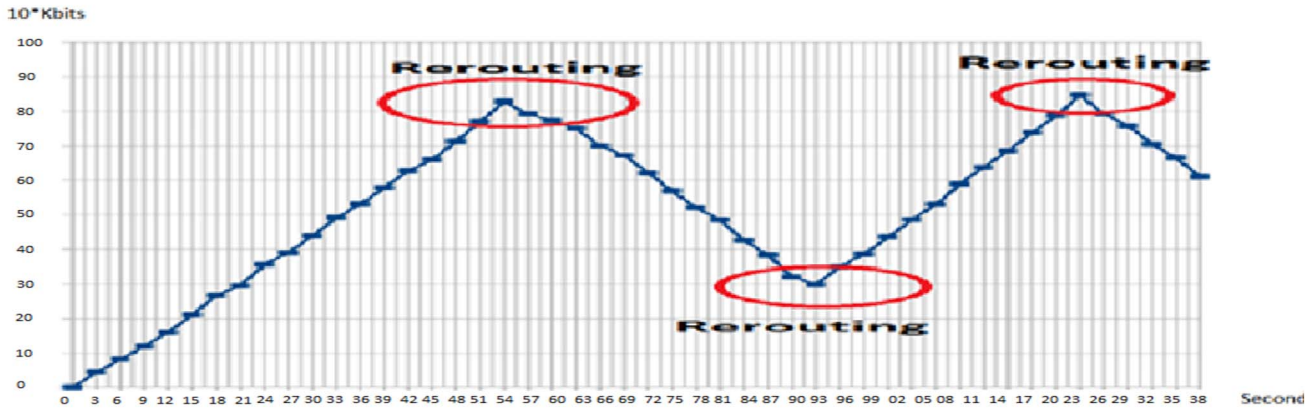

Figure 5- First Simulation without Packet Drops

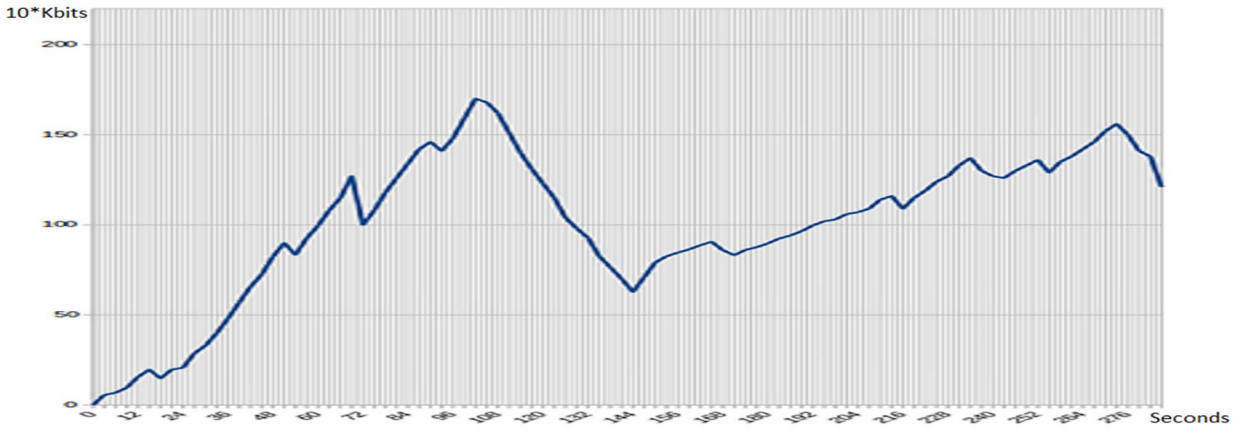

Figure 6- Simulation with Delay Control and Diminished Rerouting times for one Switch

order to simulate a real network schema completely, WFQ algorithm is used for our queues in the first switch. The aggregated queues priority was 10. The priority of our multimedia queue was 1; therefore, its output rate was 10mb/s. At first, our goal was testing the behavior of our topology with only one switch in the way of the multimedia traffics. Of course, the second switch is designed to make the implementation easier; therefore, it was one switch to forward everything to our only receiver. Its output rate was 1gb/s. Here we ran an experiment to show how rerouting hindering packet drop. It can be illustrated from figure (5) that packets are rerouted every time when queue reached to its UP_Threshold as we expected. It is important to consider that the numbers in diagram is average of several assessments with eliminating the outlier ones.

As it is axiomatic, it is not efficient to reroute our flows too many times in that it causes delay to increase. We know that for multimedia usage, it is the best to have a delay less than 150 milliseconds; however, delay less than 200 milliseconds is acceptable too, but not satisfying. Therefore, queues maximum size should be 2000Kbits. By doing this, UP_Threshold will be 1500Kbits. From (1), by assuming that the input rate would be 10200Kbit/s, T* should be yielded 7.499 seconds and we round it to 7 seconds. We round the number to down to diminish the possibility of losing the notification of rerouting. As a proof of a concept, we chose these numbers. Other numbers will have the same result. Based on WRED [14] algorithm we take the *down threshold* 750K bits. Moreover, high number of rerouting times is a problem either. As a consequence, an algorithm based on our topology is used to reduce the number of rerouting times. This algorithm, which should be different for various topologies and usages, calculate the rate of the data, which is flowed toward our switches after the first rerouting, then, those flows which their aggregation is more than 10mb/s (our output rate) and less than 10100Kbit/s flow back to their ordinary route, and other ones flow through the route to which they were rerouting. By doing this, we can enhance our algorithm and reach the figure (6).

By diminishing the rate by which our flows pass through the first queue and based on formula (1), we can easily conclude that we can reduce our queue size in order to use our resources more efficiently or we can increase T* in order to lessen the overload of our controller. Both of them may cause packet drop only before the first rerouting process. We can see it in the figure (7) which is an assessment. However the possibility of this kind of

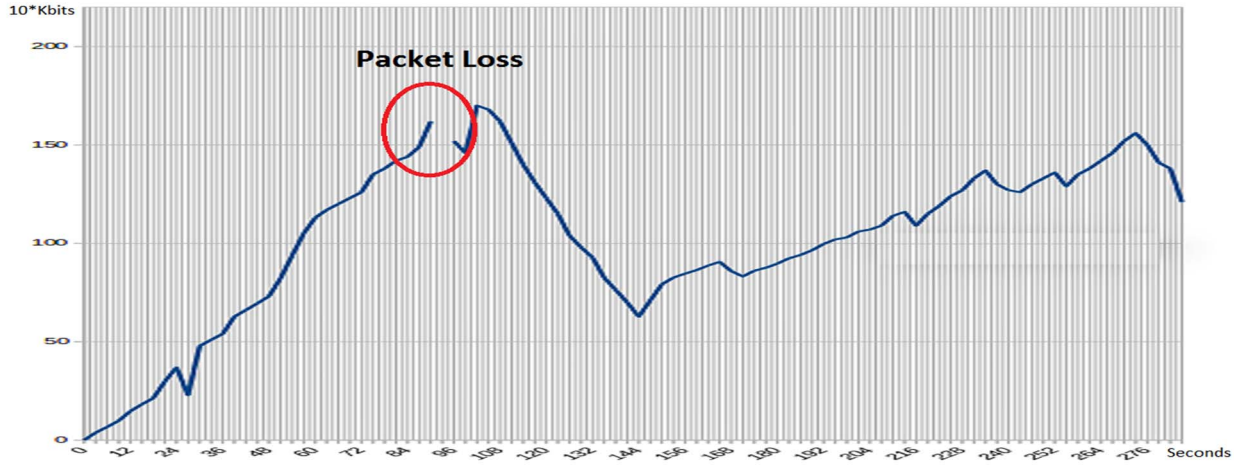

Figure 7 - Simulation with Delay Control and Diminished Rerouting times for one Switch with maximum buffer size of 1750Kbits

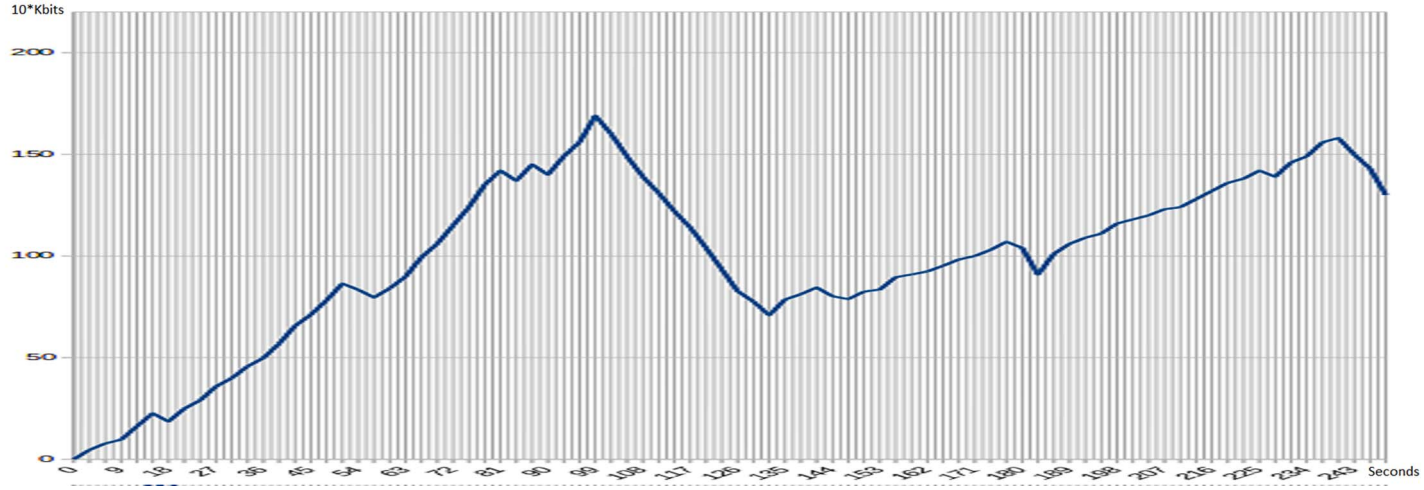

Figure 8 – Final Simulation for 8 Switches

packet loss is not low, it won't affect our system at all. As it is obvious in this figure, there could be some packet drops before the first rerouting. After that, we will have no packet drops since T* will adapt itself with the best situation in order to prevent packet loss in the next turns.

Till here we implemented packet drop and delay control algorithm for one switch in the way of packets. As it is discussed before, our final goal is to generalize project for using more than one switch. In this order, we implemented every queue sizes 2000Kbits inasmuch as we cannot change queue size dynamically. Therefore, we take queue size 2000Kbits which are the maximum size of the queue. The reason of taking 2000Kbits for maximum queue size is that, if we assume we have only one switch, more than this amount of size causes the data to be useless for us in that we only accept data less than 200 milliseconds delay. Therefore, for more than one switch, either we do not need more than 2000Kbits queue size. Based on the previous observation, we can even reduce the size of queues to 1750Kbits. By using formula (2) and figure (4) we reach figure (8) that is average of several assessments.

In the last simulation, we have 15 sender nodes. We have 3 instances of each sender node of the first simulation. The number of switches between senders and receivers are 5 in this simulation. Moreover, the number of egress ports which the multimedia packets distributed between is. We used all the aforementioned improvements in this simulation for 8 switches. The average result of several assessments has been shown in figure (8).

V. Conclusion and Future Work

In this paper, we have presented an approach to implementing QoS for multimedia services. We have explained the overall system architecture of SDN-based QoS for multimedia services. We proposed how to prioritize packets and users, how to find the best path for multimedia packets, how to reduce delay and packet loss, and how to implement a dynamic threshold for egress ports' queues, which are meant to support multimedia services. We ran the very proposal on various topologies, one of which has been shown. In this way, the reduction in delays and packet loss for multimedia services has been shown. As a future work, we will implement SDN-based QoS approach for other services and we will control packet delay and loss on physical links either.

VI. References

[1] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," 1994.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," 1998.

[3] H. Adiseshu, G. Parulkar, and R. Yavatkar, "A state management protocol for IntServ, DiffServ and label switching," in *Proceedings Sixth International Conference on Network Protocols (Cat. No.98TB100256)*, 1998, pp. 272–281.

[4] F. C. Liaw, P. Newman, T. Lyon, E. Hoffman, R. M. Hinden, and G. Minshall, "Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0." [Online]. Available: https://tools.ietf.org/html/rfc1953. [Accessed: 03-Dec-2016].

[5] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[6] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, NY, USA, 2013, pp. 7–12.

[7] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, and S.-J. Lee, "Automated and Scalable QoS Control for Network Convergence.," *INM/WREN*, vol. 10, pp. 1–1, 2008.

[8] H. E. Egilmez and A. M. Tekalp, "Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks," *IEEE Transactions on Multimedia*, vol. 16, no. 6, pp. 1597–1609, Oct. 2014.

[9] "Mininet Walkthrough - Mininet." [Online]. Available: http://mininet.org/walkthrough/. [Accessed: 03-Dec-2016].

[10] ONF, "Openflow switch specification version 1.3. 0 (wire protocol 0x04)." 2012.

[11] A. Unknown, "Open vSwitch, An Open Virtual Switch," 2010. [Online]. Available: http://www.openvswitch.org/. [Accessed: 03-Dec-2016].

[12] F. CPqD, "Openflow 1.3 software switch. Website," 2014. [Online]. Available: https://github.com/CPqD/ofsoftswitch13. [Accessed: 03-Dec-2016].

[13] M. Devera and D. Cohen, "HTB Linux queuing discipline manual-user guide," *Republik Ceko*, vol. 5, 2002.

[14] M. Wurtzler, "Analysis and simulation of weighted random early detection (WRED) queues," University of Kansas, 2002.