2021

Introducing a Test Framework for Quality of Service Mechanisms in the Context of Software-Defined Networking

Josiah Eleazar T. Regencia

William Emmanuel S. Yu
Ateneo de Manila University

Introducing a Test Framework for Quality of Service Mechanisms in the Context of Software-Defined Networking

Josiah Eleazar T. Regencia[1] and William Emmanuel S. Yu, Ph.D[2]

[1] Ateneo de Manila University, Loyola Heights, Quezon City 1108 Philippines
josiah.regencia@obf.ateneo.edu,
[2] Ateneo de Manila University, Loyola Heights, Quezon City 1108 Philippines
wyu@ateneo.edu

Abstract. In traditional non-distributed networking architecture, supporting Quality of Service (QoS) has been challenging due to its centralized nature. Software-Defined Networking (SDN) provides dynamic, flexible and scalable control and management for networks. This study introduces a test framework for testing QoS mechanisms and network topologies inside an SDN environment. Class-Based Queueing QoS mechanisms are tested as an anchor to test the introduced framework. Using a previous study as a benchmark to test the introduced framework, results show that the test framework works accordingly and is capable of producing accurate results. Moreover, results in this study show that the distributed Leaf-enforced QoS mechanisms have 11% lower latency compared to the traditional centralized Core-enforced QoS mechanisms. Leaf-enforced QoS also has approximately 0.22% more raw IP throughput than Core-enforced QoS. The HTTP throughput from the Apache Bench Transfer Rate showed that Leaf-enforced QoS with a 2.4% advantage of Core-enforced QoS. SDN is relatively new and there are many possible QoS strategies that can be applied and tested. These initiatives can benefit from an extensible testing framework.

Keywords: Software-Defined Networks, Quality of Service, Class-based Queueing, Web Traffic, Streaming Traffic

1 Introduction

The Software-Defined Networking (SDN) architecture is a relatively new technology that has been regarded as a potential solution for challenges that traditional networks currently face, a dynamic, flexible and scalable control and management for networks. While traditional networks have a control plane and a data plane inside each device, SDN separates the control plane and data plane in order to improve network management and performance. The SDN architecture has a logically centralized controller, which operates as a single unit whether the number of instances is one or more, located in the control plane to manage the flow control of the whole network. The data plane contains packet-forwarding functions for each device [15]. Dynamic optimization of network flow-management

and resources are both enabled as an effect of the separation of the control plane and the data plane. In addition, this makes it easier and more feasible to implement per-flow Quality of Service (QoS) provisioning in the network which is an advantage when dealing with multimedia flows such as VoIP and video streaming, etc since these require special QoS handling [12].

Chato and Yu explored the use of implementing a distributed QoS for SDN by studying the effects of raw IP throughput using Class-Based Queueing (CBQ) QoS algorithms [2]. Another study by Chato and Yu also explored the distributed QoS effects in terms of latency [3] inside the same environment. Both studies were performed inside a Mininet emulator [5] using a custom Pox OpenFlow Controller. Both studies showed that QoS algorithms applied to the distributed nature of SDN has better performance in terms of bandwidth and decreased latency.

This study introduces a testing framework for QoS algorithms and network topologies. As an anchor to test framework four (4) Class-based Queueing algorithms were tested. Each mechanism has two test cases each. First is the Core-enforced QoS. This represents the traditional networking architecture where QoS is only enforced at a single point, the core switch. Second is the Leaf-enforced QoS which represents the SDN architecture by distributing QoS enforcement across forwading devices closer to the edge, referred to as client leaf switches in this study. These are described in the Theoretical Framework. Intuitively, the Core-enforced QoS should perform better since only a single forwarding device is enforced with QoS. However, given the benefits of SDN shown in Table 1, the goal for the simulations is for Leaf-enforced QoS to have comparable results with Core-enforced QoS. To do this, a student's t-test is performed for every CBQ algorithm comparison of Core-enforced versus Leaf-enforced. If the resulting p-value is greater than $\alpha=0.05$, then the performance advantage of Core-enforced QoS is not significant, hence, comparable, satisfying the goal of the simulations.

As more networking applications rapidly evolve and devices connected to the internet rapidly increase, maintaining Quality of Service (QoS) across the network continues to become more challenging with the traditional networking architecture for reasons such as lack of flexibility and expensive costs. As shown in Table 1, the Software-Defined Networking (SDN) Architecture has the capabilities of addressing the challenges that the traditional networking architecture face. SDN is relatively new and there are many possible QoS strategies that can be applied and tested. These initiatives can benefit from an extensible testing framework.

The rest of this paper is organized as follows: Section 2 shows a brief literature review on SDN, Class-based and Quality of Service in SDN. Section 3 discusses the framework of the research. In particular, Section 3 discusses the architecture of the introduced test framework and also the framework as to how the framework is setup for simulations. Section 4 discusses the methodology of this research and also the tools used for simulations. Section 5 discusses the results from the simulations. Lastly, this paper is concluded in Section 6.

Table 1. SDN Architecture vs Traditional Networking Architecture

Criteria	Software-Defined Networking	Traditional Networking
Networking Approach	Virtualization. Networking devices such as firewalls, routers, load balancers can be virtualized with the use of software. This comes with SDN's programmable nature	Conventional. Special hardware devices are required for specific functionalities
Network Control	Centralized. Only the controller has to be configured in order to update protocols for all forwarding devices	Distributed. Each networking has to be configured individually.
Network Configuration	Supports automatic configuration. This also comes with the programmable nature of SDN. As a result, this saves time.	Also supports automated configuration but is usually configured manually and per device, hence, takes more time
Extensibility	High	Low
Structural Complexity	Low	High
Maintenance Cost	Low	High

2 Related Literature

2.1 Software-Defined Networking and OpenFlow

Software-Defined Networking (SDN) is a networking paradigm that introduces a centralized approach to network management where the control logic of the network is dictated by a controller by decoupling the control plane and the data plane. The controller is located inside the control plane while the data plane contains network devices that have now become simple packet forwarding devices [13]. The decoupling of the control plane and the data plane provides multiple benefits. To mention a few, one benefit is that the centralized controller is less prone to error when modifying network policies through high-level languages and software components. Second, the decoupling of both planes enables SDN to ease into Network Function Virtualization (NFV) which potentially gives major advantages in network flexibility, scalability, and also reduction of infrastructure costs. Thirdly, more fine-grained policies can be developed and deployed for the purposes of traffic engineering, Quality of Service (QoS) provisioning, security and other network management essentials [14].

Multiple research have already been performed in SDN with regards to different fields in computer networking. Peña and Yu [16] explored security possibilities for SDN by implementing a distributed firewall across the network. Instead of placing detection and blocking functions of a network on a central switch, Guevarra et al exploits the SDN architecture by distributing these capabilities

to the host-connected switches thus, enhancing detection and blocking functions of the network [9]. Chato and Yu [2] [3] also explored distributing the enforcement of Quality of Service (QoS) mechanisms to the host-connected switches using SDN. Lastly, Suba et al [18] introduced a testing framework for machine learning algorithms in order to detect and block Distributed Denial of Service (DDOS) attacks in a SDN environnment.

OpenFlow is the protocol that enables network operators to perform fine-grained implementations of network flows inside the SDN architecture [12]. It is widely considered as the standard open communication protocol used for SDN. In fact, most common SDN Controller frameworks such as OpenDaylight, POX, and Ryu use OpenFlow as their communications protocol. OpenFlow enabled switches have three (3) main parts: First is the OpenFlow protocol itself. Second is the Flow Table which contains instructions how a switch is going to process a matched flow. The last main part is a Secure Channel that connects all OpenFlow switches to the controller [15]. OpenFlow is the protocol used in this study.

2.2 Quality of Service in SDN

In traditional non-distributed networking architecture, supporting Quality of Service (QoS) has been challenging due to its centralized nature. Furthermore, the continuing advancement in network applications along with the rapidly rising number of devices connected to the internet, standard QoS policies have become no longer capable of supporting today's complex networks [12]. The flexibility and programmability of SDN has been seen as an answer to the issues traditional networks continue to face when it comes to QoS. Thus, multiple studies have been made with regards to QoS in the context of SDN.

Different types of traffic flows require different QoS mechanisms in order to optimally handle each traffic flow type across the network. Steady network resources and little to no packet drop are required for multimedia from applications such as video conferences of video streaming. In designing frameworks for such flows, prioritization and classification are both key factors [12]. To calculate QoS rich paths for video flows to be routed, Civanlar et al. [4] introduced a lienar programming-based formula while other types of traffic flows are routed using best-effort traffic on shortest paths. OpenQoS [6] uses packet header fields in order to classify incoming flows as either multimedia flows or data flows sending multimedia flows to QoS-rich paths while data flows are routed using best-effort routing.

Queuing is another common QoS mechanism. QosFlow [11] provides flexible control mechanisms by manipulating multiple packet schedulers such as Hierarchical Token Bucket (HTB), Random Early Detection (RED), and Stochastic Fairness Queuing (SFQ) schedulers. Chato and Yu [2] [3] also explored use of queuing QoS mechanisms by using four (4) algorithms Class-Based Queuing (CBQ): Basic CBQ at the Leaf, Basic CBQ at the Core, Source CBQ at the Leaf, and Source CBQ at the Core. Results in this study showed that the distributed nature of SDN had better performance over the traditional centralized networking architecture.

2.3 Class-Based Queueing

Class-Based Queueing (CBQ) is a link-sharing hierarchical resource manager [7]. Basically, what it does it to divide network traffic into classes based on the combination set by network operators. The link-sharing first ensures that link-sharing bandwidth allocation is received by every class within a given time-frame and secondly, distribute excess bandwidth fairly. To accomplish this, CBQ has to estimate limit status of each class and then check whether the queue is satisfied or unsatisfied [8].

3 Framework

3.1 Testing Framework Architecture

In this study, a testing framework is designed based from the methodology in the Chato and Yu [2,3] study. The aim of the framework is so that configurations for new Quality of Service (QoS) algorithms can be added more easily for testing than the current setup. The key component in the framework is the separation of the SDN Controller and the implementation of QoS algorithms. Figure 1 shows the architecture of the introduced testing framework. The use of separate configuration files is also another key component in this framework.

The *Topology Configuration* is used to define the number of client hosts and the number of layers of client leaf switches. This is used by Mininet topology to create virtual network topology. The number of client leaf switches is derived from the number of layers. Where N be the number of layers of client leaf switches, the number of client leaf switches directly connected to client hosts (first layer) will be 2^N. Succeeding layers of switches will then be half of the number of switches in its previous layer. The *Class Profile Configuration File* contains the Class Profile test code, the function name for the Class Profile as defined inside the CBQ Implementations Python File, the description of the QoS profile, and its test case whether QoS is core-enforced or leaf-enforced. This is used by the QoS Implementations Python File to serialize QoS mechanisms defined and used by Test Simulator for results filenames. Both the *Topology Configuration* and the *Class Profile Configuration File* are manually set by the user.

The next configuration file are created by generators using the information from the *Topology Configuration*. The *Load Configuration File* contains details of a single client host's destination server and also the file size or video streaming resolution of target file or video inside the respective server. The *Source Queue Grouping Configuration File* contains client host IP Addresses and the switch each host in connected to along with its queue assignment. This is used for for Source Class-Based Queueing mechanism. The *Hosts Configuration File* contains client hosts IP Address, client names, client leaf switch connected to, port number. Lastly, the *Client Switch Mapping Configuration File* contains client host name and switch connected to. All these configuration files are serialized

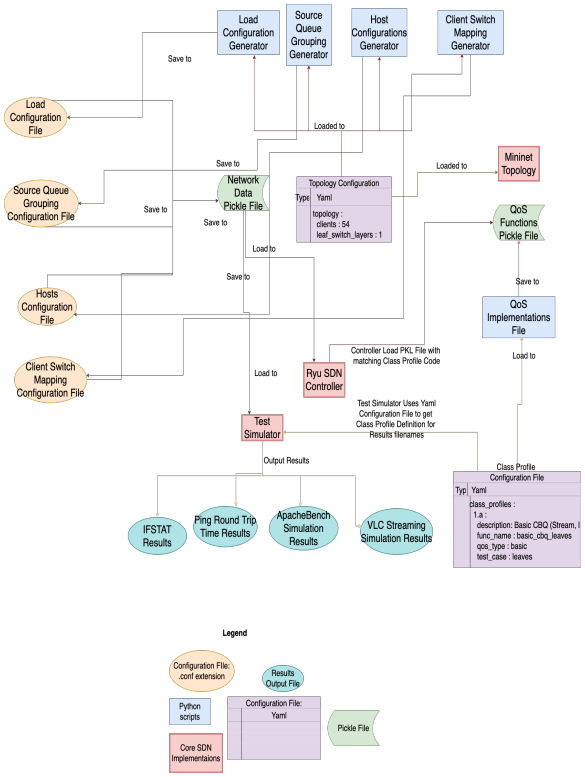

Fig. 1. Test Framework Architecture

into a Python Pickle File which will be deserialized by the test simulator and the SDN controller.

All QoS algorithms in this study are written inside the QoS Implementations File where each QoS algorithm is implemented as a function. The QoS Implementations File performs the serialization of the defined QoS functions into Python pickles. The controller then deserializes the QoS function for the specified test case. After each simulation, the Test Simulator separately saves results from the tools used in this framework: Ifstat, Ping, ApacheBench, and VLC streaming in the case of this study. The SDN Controller is started with a class profile code that should be present in the Class Profile Configuration File. Using the mentioned class profile code, the Controller loads the QoS Function pickle with the matching class profile code and unpickles it in order to implement the desired QoS mechanism. This is only performed during the initialization phase of the controller in order to avoid unnecessary additional cost to CPU resources.

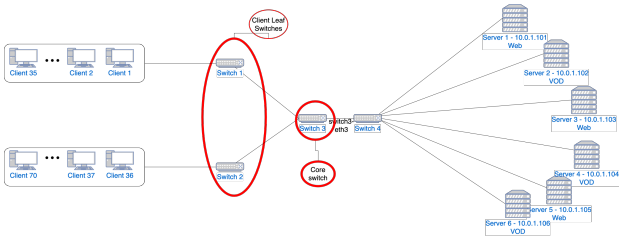

Fig. 2. Virtual Network Topology used

3.2 Conceptual Framework

This study was performed using an Amazon Web Services (AWS) EC2 m5.xlarge Ubuntu 18.04 environment with 40GB storage size. The implementation of the network topology was performed using Mininet version 2.3 [10] along with a custom Ryu OpenFlow 1.3 Controller [1].

The topology setup used in this study is shown in Figure 2. In this study, the implementation of Quality of Service (QoS) in the client leaf switches represent the distributed nature of Software-Defined Networking (SDN).

In addition, all 70 client hosts are requesting HTTP and streaming services with the use of both Apache Bench and VLC in each host. Three (3) of the servers act as HTTP servers running Python3 simple.http and the other three (3) act as Video on Demand (VOD) servers running VLC Streaming Media using VLC version 3.8. All links in the network run at the default 10Gbps of Mininet.

3.3 Theoretical Framework

Clients Configuration As mentioned in the Conceptual Framework 3.2, all 70 client hosts execute both HTTP and VLC streaming requests. Each client host is assigned whether it requests a low or a high configuration file or streaming media. Table 2 shows file size and video resolution for each configuration. The number of client hosts requesting to a specific server host is divided as equally as possible for both HTTP and VLC streaming servers. Each client host will request to a single HTTP server and a single VLC streaming server.

Apache Bench (ab) is used to send HTTP requests. It is configured to send 50,000 HTTP requests over 10 concurrent connections. For the media streaming requests, the VLC media player is used to make streaming video requests. This is done through the use of VLC Remote Control configuration and a telnet session.

Server Configuration For the server hosts, the number of servers divided among the six (6) server hosts is three (3) for each of HTTP and VOD servers. HTTP servers have two (2) jpeg files per server with different file sizes for low and high configuration requests. Python3 http.server is used to host each HTTP server. VOD servers all use VLC as their streaming server using Real Time

Table 2. Servers Setup

Server	**Server IP Address**	**HTTP Request File Size**	
		Low Test Case	**High Test Case**
server1	10.0.1.101	100KB	10MB
server3	10.0.1.103	16MB	100MB
server5	10.0.1.105	100MB	1GB
Server	**Server IP Address**	**Streaming Media Video Resolution**	
		Low Test Case	**High Test Case**
server2	10.0.1.102	360p	480p
server4	10.0.1.104	480p	720p
server6	10.0.1.106	720p	1080p

Table 3. Class Profiles

Class Profile	**CBQ Scheduling Classes**	**Switch QoS**
Basic CBQ at Leaves	Traffic Protocol	Client Leaf Switches
Basic CBQ at Core	Traffic Protocol	Core Switch
Source CBQ at Leaves	Source IP Address Grouping	Client Leaf Switches
Source CBQ at Core	Source IP Address Grouping	Core Switch
Destination CBQ at Leaves	Destination IP Address Grouping	Client Leaf Switches
Destination CBQ at Core	Destination IP Address Grouping	Core Switch
Source-Destination CBQ at Leaves	Source and Destination IP Address Grouping	Client Leaf Switches
Source-Destination CBQ at Core	Source and Destination IP Address Grouping	Core Switch

Streaming Protocol (RTSP). Each VOD server have two (2) videos of the same content but have different video resolution.

Class Profiles Class profiles in this study define the type of Class-Based Queueing (CBQ) algorithm implemented. These are defined in Table 3. In this study, Basic CBQ is classifies traffic as HTTP traffic, Streaming traffic, and lastly, all other remaining traffic types. Prioritization and limiting of behaviour between host groupings is to ensure that high traffic within a source IP group does not adversely affect others [2].

In this study, "at the Leaf" QoS mechanisms are referred to as Leaf-enforced QoS and "at the Core" QoS Mechanisms are referred to as Core-enforced QoS.

Quality of Service (QoS) Configurations Figure 2 shows which switches are the Client Leaf Switch and which switch is the Core Switch. Each switch

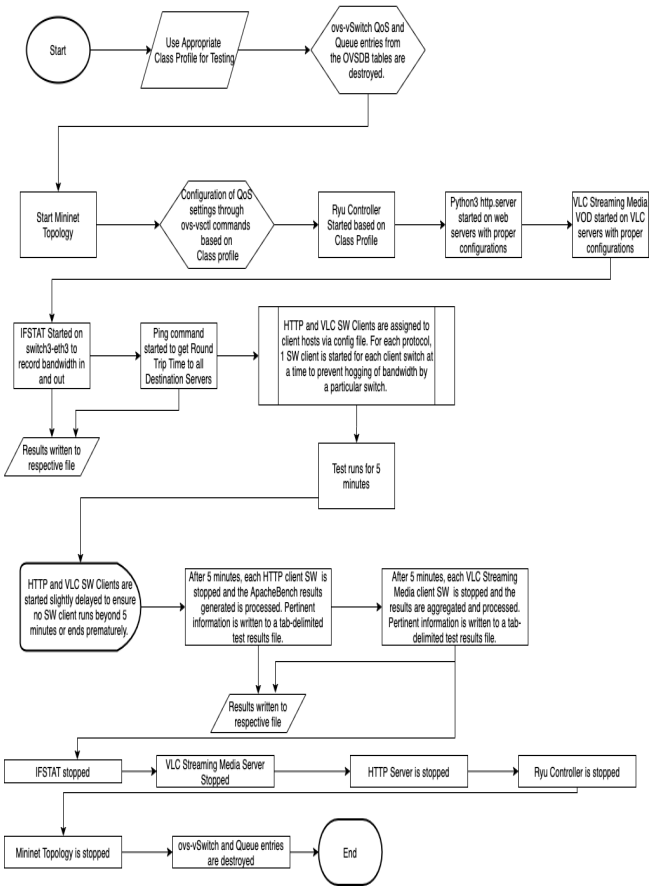

Fig. 3. Methodology Flowchart based on Chato and Yu [2]

that is enforced with QoS is allocated a bandwidth 1 Gigabit queue with three (3) queues inside of it. Each of the three (3) queues is allocated with a minimum and maximum bandwidth of 0.33 Gigabits each.

4 Methodology

The methodology in this study also follows the same methodology from Chato and Yu [2]. Each Class Profile listed in Table 3 is executed one at a time through the process shown in Figure 3.

Each Class Profile in this study was given a 5-minute window to perform test simulations. The following tools were used to perform the simulations and get data results concurrently:

- Ifstat Bandwidth results - This tool is used to get Bandwidth In and Bandwidth Out for every second during the 5-minute window. This was performed and executed at the Core Switch (switch3-eth3) for all Class Profiles
- Ping Round Trip Time - Round Trip Time (RTT) was measured in milliseconds (ms) by sending Ping packets to all destination servers from each of the 70 client hosts. Each destination server has its own measurement but the overall result was calculated by getting the mean results of all destination servers.
- ApacheBench HTTP Results - ApacheBench was used to simulate the HTTP traffic to web servers. The data taken from this tool were the transfer rate and total data transferred. The Theoretical Framework 3.3 shows specifically how the Appache Bench simulation was performed.
- VLC Streaming Media statistics results - VLC Streaming Media software client take both De-multiplexer (Demux) Bytes Read (KB) and Demux Bitrate (in Kbps). The De-multiplexer takes feeds from disparate and separate streams and assembles them into a single coherent media stream for playing.

5 Results and Discussion

Results in this study are shown statistically in Tables 4 - 7. All these data are taken from the tools mentioned in the Methodology 4. All results shown in this study are already processed and simplified to show the mean, standard deviation, minimum value, and maximum value. The experiments and results in this study both serve as an anchor to test the introduced test framework. In general, the results of this study approximately reflect results from the Chato and Yu [2] [3] study.

5.1 IFSTAT Results

Ifstat results for Bandwidth In and Bandwidth Out are shown in Table 4. Outliers in the raw data were removed. This was because the outliers represented the time frame where there were no HTTP and streaming traffic flows from Apache Bench and VLC. Hence, 90 seconds of data were removed from each QoS mechanism simulation.

In general, the throughput results from Table 4 Bandwidth Out show that Leaf-enforced QoS perform better at approximately 0.23% than Core-enforced QoS. Specifically, Destination Class-Based Queueing (CBQ) resulted to highest raw IP throughput by an average of 6% better against all other mechanisms. Although it is noted that Leaf-enforced QoS mechanisms generally resulted to higher raw IP throughput compared to its Core-enforced QoS mechanism counterpart, Source-Destination CBQ at the Core resulted to a higher raw IP throughput than Source-Destination CBQ at the Leaf. A two-sample student's t-test with significant level set to α=*0.05* was performed in order to test the statistical significance of Source-Destination CBQ at the Core against Source-Destination CBQ at the Leaf. The resulting p-value was 0.199 which is greater

Table 4. IFSTAT Results in KB/s

	Bandwidth In		**Bandwidth Out**	
CBQ Algorithm	**mean**	**std. dev.**	**mean**	**std. dev.**
Basic CBQ at the Leaf	857,241.0	57,046.89	2,913.0	231.94
Basic CBQ at the Core	735,776.12	54,882.73	2,430.67	207.28
Src CBQ at the Leaf	832,973.98	58,974.58	2,783.07	213.55
Src CBQ at the Core	847,453.89	60,688.61	2,912.67	221.89
Dst CBQ at the Leaf	867,723.37	58,904.75	2,962.94	225.07
Dst CBQ at the Core	856,943.42	43,341.81	2,942.93	180.69
Src-Dst CBQ at the Leaf	849,536.17	61,722.35	2,855.84	266.45
Src-Dst CBQ at the Core	746,912.39	60,402.67	2,482.74	217.64

than α=*0.05*. Hence the raw IP throughput advantage of Source-Destination CBQ at the Core over Source-Destination CBQ at the Leaf is statistically insignificant. Hence, for IP raw data, Leaf-enforced QoS mechanisms have more throughput — or at least as good as — compared to Core-enforced QoS mechanisms. This is despite having more nodes enforced with QoS for Leaf-enforced QoS mechanism.

In addition, a separate study by the researchers using the introduced test framework with multiple layers in the topology and with more network traffic flow being generate showed results that all Leaf-enforced QoS resulted to significantly more raw IP throughput than Core-enforced QoS [17].

5.2 Apache Bench Results

For the web server simulations, ApacheBench (ab) was used to simulate and gather data. The following data taken in this test were the Total Transferred Data and the Transfer Rate which can be found Table 5.

The HTTP data throughput shown in Table 5 Transfer Rate show that overall, Leaf-enforced QoS perform approximate 2.4% better against the Core-enforced QoS mechanisms. Destination Class-Based Queueing (CBQ) ath the Leaf mechanism also performed best by an average of 5.7% against all other mechanisms. Core-enforced Source-Destination CBQ QoS, however, had higher HTTP throughput than Leaf-enforced Source-Destination CBQ QoS by around -0.26%. Using a two-sample student's t-test with significant level set to α=*0.05*, the p-values resulted to 0.0.9853 for the performance difference between Core-enforced Source-Destination CBQ QoS and Leaf-enforced QoS. Hence the distributed nature of the Software-Defined Networking architecture performs at

Table 5. Apache Bench Results

	Total Transferred (KB)		**Transfer Rate (KBps)**	
CBQ Algorithm	**mean**	**std. dev.**	**mean**	**std. dev.**
Basic CBQ at the Leaf	4,927,743.37	3,830,391.11	15,913.38	12,355.04
Basic CBQ at the Core	4,186,955.07	3,322,636.4	13,593.94	10,779.95
Src CBQ at the Leaf	4,802,749.11	3,511,440.8	15,482.36	11,277.1
Src CBQ at the Core	4,873,786.79	3,551,581.79	15,743.02	11,441.85
Dst CBQ at the Leaf	4,960,232.07	3,637,629.65	16,107.2	11,800.18
Dst CBQ at the Core	4,908,274.32	3,535,940.17	15,888.49	11,421.28
Src-Dst CBQ at the Leaf	4,870,591.95	3,631,200.2	15,764.16	11,733.97
Src-Dst CBQ at the Core	4,245,625.35	3,189,979.13	13,806.68	10,364.87

least as good as or even better compared to the traditional centralized networking architecture. The high standard deviation observed in Table 5 Transfer Rate is expected due to the bursty traffic nature of HTTP.

5.3 VLC Results

VLC Streaming Media results are found in Table 6. The results in this section show that all CBQ algorithms in this study nearly have the same performance in terms of the VLC Streaming Media. Moreover, the streaming media throughput shown in Table 6 Bitrate resulted to Core-enforced QoS performing approximately 0.2% better than Core-enforced QoS. The only Leaf-enforced QoS that had higher bitrate compared to its Core-enforced QoS counterpart was the Leaf-enforced Destination CBQ QoS algorithm with 1% higher bitrate than the Core-enforced Destination CBQ QoS algorithm.

Despite that, the student's t-test have shown that the advantages are statistically insignificant, hence, still comparable and thus, satisfying the goals of the experiments. For Basic CBQ, the Core-enforced Basic CBQ resulted to 0.2% higher bitrate than the Leaf-enforced CBQ but the difference had a p-value of 0.986 which is greater than α=*0.05*. For Source CBQ, the Core-enforced Source CBQ resulted to 0.1.6% higher bitrate than the Leaf-enforced Source CBQ but the difference had a p-value of 0.8552 which is greater than α=*0.05*. For Source-Destination CBQ, the Core-enforced Source-Destination CBQ resulted to 0.08% higher bitrate than the Leaf-enforced Source-Destination CBQ but the difference had a p-value of 0.9931 which is greater than α=*0.05*. Furthermore, the 1% advantage of Leaf-enforced Destination CBQ QoS algorithm over Core-enforced

Table 6. VLC Results

	Total Bytes Read (KB)		**Bitrate (Kbps)**	
CBQ Algorithm	**mean**	**std. dev.**	**mean**	**std. dev.**
Basic CBQ at the Leaf	25,569.0	13,064.05	737.06	390.54
Basic CBQ at the Core	25,574.98	13,071.17	731.58	374.32
Src CBQ at the Leaf	25,180.24	13,498.72	739.39	401.78
Src CBQ at the Core	25,323.76	13,073.63	733.78	384.76
Dst CBQ at the Leaf	25,559.52	13,052.95	741.84	396.46
Dst CBQ at the Core	25,565.41	13,071.55	734.46	380.75
Src-Dst CBQ at the Leaf	25,294.83	13,390.5	735.89	391.87
Src-Dst CBQ at the Core	25,011.5	13,640.5	727.83	380.15

Destination CBQ QoS algorithm resulted with a p-value of 0.9073 which is greater thanα*=0.05*. Meaning, the Core-enforced Destination CBQ is still comparable with Leaf-enforced Destination CBQ. This shows that the throughput of streaming traffic is not affected regardless of whether the enforcement of QoS is centralized or distributed.

5.4 Round Trip Time Results

Table 7 shows results for Round Trip Time (RTT) for the ping simulation. Ping command was executed for all 70 client servers to all 6 destination servers. The results in Table 7 show the overall mean for RTT results of all 6 destination servers. RTT results show that Leaf-enforced QoS generally has lower latency compared to Core-enforced QoS by approximately 11% with Destination Class-Based Queueing (CBQ) at the Leaf having lowest latency by an average of 17% lower than all other CBQ mechanisms. Specifically, Destination Class-Based Queueing (CBQ) at the Leaf had 13.6% less latency against it Core-enforced counterpart, Destination Class-Based Queueing (CBQ) at the Core. Moreover, Basic CBQ at the Leaf has 11% lower latency than Basic CBQ at the Core, Source CBQ at the Leaf has 14.6% lower latency than Source CBQ at the Core, and Source-Destination CBQ at the Leaf has 5.3% lower latency than Source-Destination CBQ at the Core.

6 Conclusion

Due to the increase of network traffic flows and increase in complexity of network infrastructure, Software-Defined Networking (SDN) has been regarded as

Table 7. Ping Results

	Round Trip Time (ms)	
CBQ Algorithm	**mean**	**std. dev.**
Basic CBQ at the Leaf	0.0305	0.01381
Basic CBQ at the Core	0.03986	0.02123
Src CBQ at the Leaf	0.03145	0.01564
Src CBQ at the Core	0.03186	0.016
Dst CBQ at the Leaf	0.03004	0.01414
Dst CBQ at the Core	0.0304	0.01428
Src-Dst CBQ at the Leaf	0.03244	0.01899
Src-Dst CBQ at the Core	0.0391	0.02288

a solution for Quality of Service (QoS). And with the features of SDN such as its programmability and the separation of the control plane and the data plane, more fine-grained QoS mechanisms can be developed in order to ensure successful and efficient packet delivery across the network. As such, this study introduced a test framework for QoS mechanisms and network topologies specifically for the SDN architecture. The experiments performed in this study were based on the experiments performed by Chato and Yu [2] [3]. In this study however, the QoS mechanisms were tested with the use of the introduced test framework. The results in this study were similar to results from the Chato and Yu study. Moreover, the results have also shown that Leaf-enforced QoS generally performed better than Core-enforced QoS. Which shows the advantage of the distributed nature of SDN over the traditional centralized nature of the networking architecture. This is despite having more points in the network enforced with QoS. There are many possible QoS strategies that can be applied and tested by exploiting the advantages of SDN. The use of the introduced test framework can benefit SDN researchers in QoS with the exploration of possible strategies.

References

1. URL https://github.com/faucetsdn/ryu
2. Chato, O., Yu, W.E.S.: An exploration of various qos mechanisms in an openflow and sdn environment. In: Accepted for Presentation in The International Conference on Systems and Informatics (ICSAI-2016) (2016)
3. Chato, O., Yu, W.E.S.: An exploration of various quality of service mechanisms in an openflow and software defined networking environment in terms of latency performance. In: 2016 International Conference on Information Science and Security (ICISS), pp. 1–7. IEEE (2016)
4. Civanlar, S., Parlakisik, M., Tekalp, A.M., Gorkemli, B., Kaytaz, B., Onem, E.: A qos-enabled openflow environment for scalable video streaming. In: 2010 IEEE Globecom Workshops, pp. 351–356. IEEE (2010)
5. De Oliveira, R.L.S., Schweitzer, C.M., Shinoda, A.A., Prete, L.R.: Using mininet for emulation and prototyping software-defined networks. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), pp. 1–6. Ieee (2014)

6. Egilmez, H.E., Dane, S.T., Bagci, K.T., Tekalp, A.M.: Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In: Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference, pp. 1–8. IEEE (2012)
7. Floyd, S., Jacobson, V.: Link-sharing and resource management models for packet networks. IEEE/ACM transactions on Networking **3**(4), 365–386 (1995)
8. Fragouli, C., Sivaraman, V., Srivastava, M.B.: Controlled multimedia wireless link sharing via enhanced class-based queuing with channel-state-dependent packet scheduling. In: Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98, vol. 2, pp. 572–580. IEEE (1998)
9. Guevara, A., Domingo, M., Yu, W.: Enhancing intrusion detection and prevention systems using software defined networking in a distributed topology. In: 17th proceedings on philippine computing science congress. CSP, Quezon City, Philippines, pp. 219–228 (2017)
10. Huang, T.Y., Jeyakumar, V., Lantz, B., Feamster, N., Winstein, K., Sivaraman, A.: Teaching computer networking with mininet. In: ACM SIGCOMM (2014)
11. Ishimori, A., Farias, F., Cerqueira, E., Abelém, A.: Control of multiple packet schedulers for improving qos on openflow/sdn networking. In: 2013 Second European Workshop on Software Defined Networks, pp. 81–86. IEEE (2013)
12. Karakus, M., Durresi, A.: Quality of service (qos) in software defined networking (sdn): A survey. Journal of Network and Computer Applications **80**, 200–218 (2017)
13. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Communications Magazine **51**(2), 114–119 (2013)
14. Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: A comprehensive survey. Proceedings of the IEEE **103**(1), 14–76 (2014)
15. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review **38**(2), 69–74 (2008)
16. Pena, J.G.V., Yu, W.E.: Development of a distributed firewall using software defined networking technology. In: 2014 4th IEEE International Conference on Information Science and Technology, pp. 449–452. IEEE (2014)
17. Regencia, J.E.T., Yu, W.E.S.: Latency and throughput advantage of leaf-enforced quality of service in software-defined networking for large traffic flows. Submitted to: SAI Computing Conference 2021
18. Suba, A.M., Bautista, K.V., Ledesma, J.C.T., Yu, W.E.: Developing a testing framework for intrusion detection algorithms using software defined networking. In: International Conference on Information Science and Applications, pp. 303–313. Springer (2018)