ON DEMAND QOS BANDWIDTH MANAGEMENT APPROACH IN SOFTWARE DEFINED NETWORKING

Master of Science Thesis

MHD. Yassin ALNAHHAS

Eskişehir, 2019

ON DEMAND QOS BANDWIDTH MANAGEMENT APPROACH IN SOFTWARE DEFINED NETWORKING

MHD. Yassin ALNAHHAS

MASTER OF SCIENCE THESIS

Electrical and Electronics Department

Supervisor: Assoc. Prof. Dr. Nuray AT

Eskişehir

Anadolu University

Graduate School of Sciences

December 2019

FINAL APPROVAL FOR THESIS

This thesis titled “On Demand QoS Bandwidth Management Approach in Software Defined Networking” has been prepared and submitted by MHD. Yassin ALNAHHAS in partial fulfillment of the requirements in “Anadolu University Directive on Graduate Education and Examination” for the Degree of Master of Science in Electrical and Electronics Engineering Department has been examined and approved on 16/12/2019

Committee Members	**Title Name Surname**	**Signature**
Member (Supervisor)	: Assoc. Prof. Dr. Nuray AT	.....
Member	: Assoc. Prof. Dr. Tansu FİLİK	.....
Member	: Asst. Prof. Dr. Gökhan DINDIŞ	.....

Prof. Dr. Murat TANIŞLI

Director of Institute of Graduate Programs

ABSTRACT

ON DEMAND QOS BANDWIDTH MANAGEMENT APPROACH IN SOFTWARE DEFINED NETWORKING

MHD. Yassin ALNAHHAS

Department of Electrical and Electronics Engineering

Anadolu University, Institute of Graduate Programs, December 2019

Supervisor: Assoc. Prof. Dr. Nuray AT

The advent of Software-defined networking (SDN) as one of the promising solutions of the future Internet has been characterized by its two distinct features including separation of the control plane from the data plane and granting programming ability for network application development in various areas such as: security, Quality of Service (QoS), traffic engineering (TE), load balancing, etc. The main goal of SDN is to centralize the network control in an intelligent software called the controller. One of the problems that can be solved using the concept of SDN is network bandwidth allocation to provide QoS. This study aims at exploiting and discussing the implementation of QoS mechanism by proposing an SDN-app to provide user defined bandwidth allocation to manage different kinds of traffic such as VOIP, TCP and UDP traffic and evaluate the extent to which these QoS services can be guaranteed on RYU controller. The intelligence of SDN can be utilized to ensure that network bandwidth is allocated to the applications that need it and according to the user demand as well. In addition, a monitoring system has been proposed to monitor the behavior of the queues. The proposed method ensure rate promising and limiting bandwidth for different application requirements. SDN network was built and implemented on Mininet runs on a virtual machine (VM), to test and analyze the effectiveness of the proposed framework.

Keywords: Software-Defined Networking (SDN), Mininet, Quality of Service (QoS), RYU

ÖZET

YAZILIM TANIMLI AĞLARDA TALEP ÜZERİNE SERVİS KALİTESİ BANT GENİŞLİĞİ YÖNETİMİ YAKLAŞIMI

MHD. Yassin ALNAHHAS

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Anadolu Üniversitesi, Lisansüstü Eğitim Enstitüsü, Aralık 2019

Danışman: Doç. Dr. Nuray AT

Geleceğin interneti için umut verici çözümlerden biri olarak görülen yazılım tanımlı ağlar (SDN), kontrol düzlemini veri düzleminden ayırması ve güvenlik, servis kalitesi (QoS), trafik mühendisliği, yük dengeleme gibi farklı alanlarda ağ uygulamaları geliştirilebilmesi için programlama yeteneği sağlaması gibi özellikleriyle karakterize edilirler. SDN'in ana amacı ağ kontrolünü denetleyici adı verilen akıllı bir yazılımla merkezi hale getirmektir. Çözümünde SDN kavramının kullanılabileceği problemlerden birisi de hizmet kalitesini sağlamak için ağ bant genişliği tahsisi sorunudur. Bu çalışma, VOIP, TCP ve UDP gibi farklı trafik türlerini yönetmek üzere kullanıcı tanımlı bant genişliği tahsisi yapılmasını sağlayan bir SDN-uygulaması önermekte ve bu QoS mekanizmasının gerçeklemesini tartışmaktadır; aynı zamanda, hangi QoS servislerinin RYU denetleyicisinde garanti altına alınabileceğini değerlendirmektedir. SDN yetenekleri, uygulamaların ihtiyaçlarına ve kullanıcı taleplerine göre, ağ bant genişliği tahsisinde kullanılabilir. Bu çalışmada ayrıca kuyrukların davranışlarını gözlemlemek üzere bir izleme sistemi önerilmiştir. Önerilen yöntem farklı uygulama gereksinimlerine göre hız vaat etmekte ve bant genişliğini sınırlamaktadır. Önerilen sistemi test etmek ve etkinliğini analiz etmek üzere SDN ağı sanal makinede çalışan Mininet üzerinde kurulmuş ve gerçeklenmiştir.

Anahtar Kelimeler: Yazalım Tanımlı Ağlar (SDN), Mininet, Servis Kalitesi (QoS), RYU

16/12/2019

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Anadolu University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

MHD. Yassin ALNAHHAS

.....

ACKNOWLEDGEMENT

Foremost, I would like to take the opportunity to wholeheartedly thank my supervisor, Assoc. Prof. Dr. Nuray AT for the profound support, motivation, patience and the great listener she was during my research and thesis work. I am gratefully appreciating the support of the Turkish government and the generous scholarship provided by the YTB for providing this opportunity. I would like also to thank all the teachers who mentored me during my thesis courses and the wonderful devotion they show to teach us. Finally, I cannot forget my thank to my family and friends for all the love, support and motivation throughout my bachelor and master studies.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

API	: Application Programming Interface
CIR	: Committed Information Rate
CLI	: Command Line Interface
DiffServ	: Differentiated Services
DSCP	: Differentiated Service Code Point
HFSC	: Hierarchical Token Bucket
HTB	: Hierarchical Token Bucket
IOT	: Internet of Things
ITU	: International Telecommunication Union
IETF	: Internet Engineering Task Force
IntServ	: Integrated Services
LLDP	: Link Layer Discovery Protocol
NBI	: North Bound Interface
NFV	: Network Function Virtualization
OF	: OpenFlow
ONF	: Open Networking Foundation
SBI	: South Bound Interface
NBI	: North Bound Interface
SLA	: Service Level Agreement
TLS	: Transport Layer Security
TC	: Traffic Control
TCP	: Transport Control Protocol
UDP	: User Datagram Protocol
VOIP	: Voice Over Internet Protocol

1. INTRODUCTION

The Internet has extended to new kinds of networking applications and services (e.g., multimedia, video conferencing, Internet of Things (IoT), VOIP, e-commerce etc.) requiring different manipulations for their own flows to make the delivery successful over a network [1]. These applications require higher bandwidth resources. For example, throughput for real-time multimedia services, low latency (delay) for VOIP or online gaming with low jitter. However, today's de facto delivery model, best effort, in the Internet is not capable of serving to all of these services. In addition, proposed QoS solutions have not been successful enough to solve the QoS challenges of legacy networking architectures [2].

The Internet Engineering Task Force (IETF) has proposed several QoS architectures to the Internet, but none has been truly successful and globally applied. The Integrated Services (IntServ) model utilizes the resource reservation protocol (RSVP) to provide the QoS to end users. In IntServ model, resources are reserved at each router along the flow path (end to end path) and hence all routers store network states related to the services [2], [3]. Therefore, this model has a scalability problem as it is based on individual flows. Differentiated Services (DiffServ) model facilitates the network operation and improves the resiliency of resource allocation [4]. However, due to lack of resource reservation, DiffServ cannot provide the reliability of end to end connection. The Multi-Protocol Label Switching (MPLS) provides a partial solution with its traffic engineering capability. However, due to non-resiliency of underlying protocol MPLS is also known to be hard to configure, manage and troubleshoot [5]. Due to these shortcomings, a novel approach is needed to address some of these challenges. Thus, a new architecture called SDN has emerged in recent years. According to the Open Networking Foundation (ONF), SDN architecture decouples the control and data plane, provides logically centralized view of the network intelligence, and the underlying infrastructure is abstracted from the application layer. Practically speaking this architecture has made the network programmable which was infeasible in traditional networks. So, network administrators can easily and instantly configure, manage, and enhance network resources with automated, proprietary-free and dynamic programs. OpenFlow is a protocol responsible for the communication between the controller and the forwarding devices. The OpenFlow protocol specifications are controlled and published

by the ONF which is led by well-known large companies that own and operate some of the largest networks in the world (e.g. Facebook, Google, Yahoo, Microsoft, Verizon) [6]. Some of SDN controllers available today include Open Daylight, Floodlight, RYU [7], NOX, ONOS, etc.

In SDN notion, network applications are not forced to deal with low level configurations for data plane devices. Controllers can overview the entire network states including switch statistics, network resources availability, events and so on, by sampling packets. With the help of this information, control policies and Service Level Agreements (SLAs) can be specified by the operator at a higher abstraction level (e.g. controller) without the necessity of reconfiguring low-level configurations at each forwarding device. The set of policies and also the different flow classes are unconstrained allowing for fine-grained tuning based on the needs of the admin/user. SDN can help network operators to create vigorous and easy to use automated QoS management architectures by means of resource reservation and queue management for their networks [2]. In this work, we will take advantage of SDN architecture to adjust QoS parameters and classify the flows into QoS flows (TCP and VOIP flows) and best effort flows (UDP flows). This means that the administrator will define the required bandwidth the application needs according to the link capacity available, so the bandwidth must be guaranteed for that application. To this effect, we prioritize guaranteed QoS flows over best effort traffic using Open-flow queues (3 queues on each port), and the Open vSwitch Database (OVSDB) protocol for configuration management. Although SDN controller can define data flows, QoS management for end users remains difficult task. This is because SDN controller does not provide QoS policy and management framework to define constraint on end-to-end network. Hence, it is essential to develop and incorporate an effective QoS management system with an SDN controller. The main contributions of this study can be given as follows:

- Proposed QoS framework supports multiple network topologies. Specifically, tree and linear topologies have been considered and tested in this study.
- QoS capabilities are promoted using the RYU controller including rate limiting and resource reservation.
- Different flow types are distinguished and mapped into corresponding queues configured by the OVSDB protocol at the ports of the data plane devices.

- A real time monitoring system is implemented to observe the states of the traffic that measures the desired throughput and bandwidth allocated according to the application needs.
- QoS SDN application has been formed to perform the QoS operation in the SDN controller.

1.1 The Objective of The Study

The main objective of this thesis is to implement a QoS mechanism by proposing an SDN-app to provide user defined bandwidth allocation to enable bandwidth guarantees in OpenFlow networks. The proposed SDN-app should have the following features:

1. Provide bandwidth guarantees for QoS traffic including TCP, UDP and VOIP flows according to specified bandwidth set by the user/admin and according to a link capacity as well.
2. Our framework provide support for multiple network topologies to apply QoS constraints.
3. Designing a monitoring system to observe the states of traffic, i.e., to observe the queues utilization performance.

1.2 Research Questions

According to the objective of the study some research questions can be stated as follows:

1. How to provide bandwidth allocation in OpenFlow networks?
2. Can QoS be provided with RYU SDN controller?
3. Can traffic classifications and rate limiting be provided using queues in OpenFlow?
4. Can the RYU controller be used to build up a monitoring system to observe the utilization of queues?

1.3 Thesis Structure

The thesis is organized as follows. Chapter 2 provides background information related to quality of service and SDN including the architecture model of SDN and the three basic layers the Openflow protocol, OVSDB protocol and comparison between the traditional network and SDN are also given in this chapter. Chapter 3 describes the proposed quality of service model to apply the QoS constraints to achieve the on-demand bandwidth based

on type of application. Chapter 4 presents the test results and the implementation procedures including the network topologies tested, and the tools used to generate and measure the traffic bandwidth and describe the environment used in the simulation process. Chapter 5 provides the conclusion and the suggestions for possible future work.

2. BACKGROUND ON SOFTWARE DEFINED NETWORKING

Software Defined Networking (SDN) is an emerging networking notion that gives hope to change the way of how the current traditional network architecture works. The tide integration between the software and hardware or, in other words, control plane and data plane limit the network scalability. This integration has an impact as each networking manufacture's charge extra for software running on their devices. This leads to the idea of proprietary. SDN concept has been introduced to overcome the problem of proprietary in the traditional networks. The main objective of SDN is to dis-aggregate the control plane and forwarding plane. That means to have a single unit in the network in the control plane often called the controller and multiple SDN enabled devices in the data plane often called the OpenFlow switches. In addition, this dis-aggregation is a breakthrough to achieve flexibility, disassemble the network control into dissolvable pieces, for example, in the traditional networking architecture routing is based on per hop at every router [8], [9].

The idea of programmable and OpenFlow networks was first introduced by a PhD student Martin Casado from Stanford University in 2008. It was first started as an academic experiment in the campus network of Stanford University [10]. After that they have gained considerable attraction in industry nowadays. The SDN momentum was strong enough to drag the attention of some big companies such as Google, Facebook, Microsoft to fund the ONF for the development of SDN architecture.

In traditional networks, if someone wants to implement a new routing protocol, every router has to be reprogrammed or replaced by new hardware because each networking vendor has its own software and hardware specifications. SDN concept has broken these restrictions of proprietary and changed the traditional networks to programmable, centrally managed networks encouraging networks development and innovation by creating a suitable environment to implement, test, create and develop new applications and services for the network [11]. The SDN framework provides centralized control of the forwarding elements freely of the network technology applied to connect the data plane devices that can be manufactured from different vendors. Consequently, the forwarding elements themselves no longer need to understand the underlying processes and protocol standards because of the abstraction provided by the SDN framework but just simply accept commands from the SDN controllers.

2.1 SDN Architecture

The ONF suggested a model for SDN framework to apply the concepts of networking programmability and automation. This model consists of three major layers namely, an application layer, a control layer and an infrastructure layer stacking over each other as demonstrated in Figure 2.1.

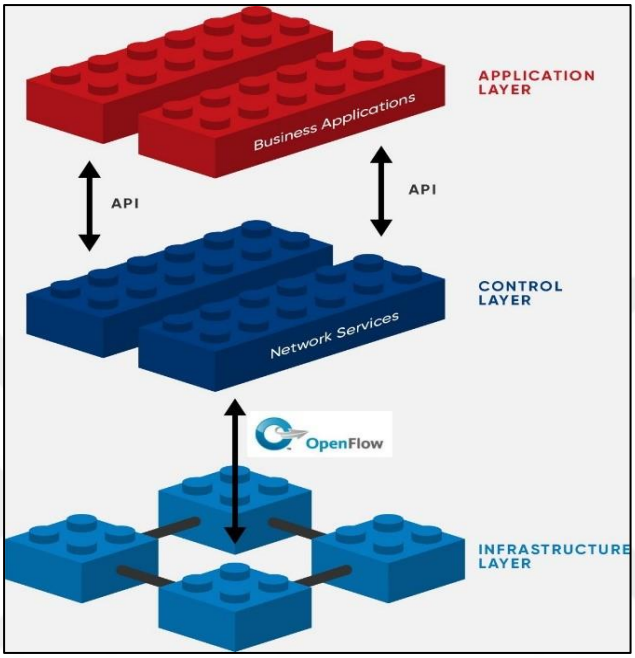

Figure 2.1. *Software defined network architecture* **[12]**

2.1.1 Infrastructure Layer

The infrastructure layer consists of the metal bare or virtual devices (e.g., switches, routers, middlebox, virtual switches, etc.) in the forwarding plane, these devices are mainly responsible for gathering network status and send them to the controller which can monitor and update the performance of the entire network [13]. The network status could include information about the network topology, network resources (e.g., bandwidth), traffic statistics, queue statistics and network usage. Moreover, they are in charge of processing packets based on the rules provided by the controller [14].

The forwarding devices in this layer can be interconnected through different transmission media through wireless radio channels, or copper wires and also optical fibers. The main difference between the traditional devices in traditional network resides in the fact that, the network intelligence is removed from these devices and assigned to centralized control system.

2.1.2 Control Layer

The control layer is the bridge that connects the application layer and the infrastructure layer via two interfaces, the North Bound Interface (NBI) which provides service access points for example, an Application Programming Interface (API) which can be used by the SDN applications to fetch and access network status information from the switches. By using these APIs, we can make system tuning and implement rules and policies to the switching devices. In addition to that, these APIs makes the developer and the system administrator job easier for them to apply and test new protocols. The South Bound Interface (SBI) (such as OpenFlow, NETCONF) feature some functions to the controller such as reporting network status and extracting packet forwarding rules. This layer is composed of one or multiple software controllers, which represent the brain of the network where all the decisions about forwarding, policing, routing are taken. Control layer consists of two main components; the Network Operating System (NOS) and applications, we can consider the NOS as the SDN controller. There are many points that affect the design of the controller since the controller design is a critical task and must be done accurately and efficiently. We can categorize some criteria used for controller design into the following classifications [15]:

- **Scalability:** The scalability issue is considered a critical problem in specific areas. Studies are conducted to improve scalability. DevoFlow is one of the models proposed to enable scalable management architecture for OpenFlow networks [16], specially by alleviating the overhead of the controller so one way to accomplish scalability is to design the controller with a distributed architecture.
- **State Consistency:** Real world networks are too large to be handled by only one controller that will be considered a single point of failure, that's why some kind of redundancy must be satisfied meaning that we can have multiple controllers, and one switch may also be connected to more than one controller. Therefore, the controller itself must maintain consistency across the distributed system because both the controllers and the OpenFlow switches has to hold the same forwarding policy in order to function in the right way.
- **Availability:** Failure recovery and tolerating from link and switch failures are essential requirements of most networks, so SDN controllers and OpenFlow

switches should be robust in different situations hence many studies suggest various ways to achieve high availability. In [17], the controller can install static instructions on the switches to ensure network and topology connectivity and track link failures based on these instructions. In order to discover link and node breakdown the SDN controller use the Link Layer Discovery Protocol (LLDP) messages, which our controller is using to achieve availability.

- **Security and Dependability:** The main concept of SDN is to make the network programmable, so developers now have the power to write applications that run on the controllers and manage the network behavior. This ability came from the nature of the centralized control of SDN on whole network flow and observing behavior of clients and users make SDN possible to detect attacks and prevent them. There are many types of OpenFlow controllers some of them open source and others are commercial. The controller is the brain of the network and since the controller plays major role in SDN environment, it configures and manages the network resources and provides an abstraction of the network resources to applications through interfaces (SBI, NBI) and the relevant information and data models.

Table 2.1 shows a list of some prominent controller's classifications, the language used, developers and overview.

Table 2.1. *Controllers classifications and types*

Controller	Programming Language	Developer	Overview
NOX [18]	C++/Python	Nicira	This controller is the first SDN controller developed. Applications are written in high level programming languages like C++ Python.
POX [19]	Python	Nicira	Open source controller and it is the python version of NOX controller. It supports graphical user interface.
Openday light [20]	Java	Linux Foundation	It was the first open source controller that support non openflow proprietary protocols such as OpFlex by Cisco. In addition to the other protocols (OF, OVSDB, NETCONF, OF-CONFIG, etc.)
Floodlight [21]	Java	Big Switch	One of the most popular opensource controllers, this controller is based on Beacon controller from Stanford University, its source code can be found in GitHub repository.
RYU [7]	Python	NTT	RYU is a component based SDN controller, aims to provide well defined APIs which makes it easy for developers to create new network management and control applications
ONOS [22]	Java	Open Networking LAB	ONOS is one of the production ready controllers, which has variety use of applications and it is based on distributed core platform providing a well global network view using cloud repositories, RAM and manager to implement clusters.

2.1.3 Application Layer

The application layer contains the SDN applications responsible for achieving the user requirements and specify network services or business applications by identifying a service aware behavior of network resources in a programmatic means. These applications have the ability to access and control the forwarding devices in the infrastructure layer. The programming of an SDN applications take the advantage of the abstraction layer provided by the SDN controller exposed via the NBI and SBI [23]. Therefore, the developers do not have to worry about the underlying protocols and devices in the data plane layer when writing networking applications since SDN is vendor agnostic architecture, let alone the abstraction provided by SDN as well. Moreover, it is like an Operating System (OS) which provides rich abstractions for managing hardware level resources so the programmers do not care about hardware when they write their programs because of that abstraction. The function of these applications may include server load balancing, dynamic access control, network virtualization, security and QoS management.

2.2 SDN vs Traditional Network

The difference between the SDN architecture and the traditional network architecture can be realized from Figure 2.2. It shows clearly how the control logic is managed centrally, and the data plane are simply forwarding devices. Management will be simpler and even middleboxes services can be delivered as SDN controller applications. The SDN approach is a solution to simplify networking operations and provide an innovative way to optimize the network management and increase the flexibility in managing the networking devices. In this context we can identify some key problems faced in the management of the traditional network architecture as follow [24]:

- **Complex Network Configurations**: The control plane and the data plane are merged in the same entity and each device is typically configured in vendor specific way. Additionally, the rapid increase of the size of the network, and the changing networking conditions all together have increased the complexity of the configuration process since every time each device has to configured individually which introduce additional configuration errors and management difficulties.
- **Dynamic Network State**: Networks are growing in a fast pace and new trends and technologies are being introduced in networking field such as network virtualization and cloud computing, so the network has to be more dynamic to

change as hosts are continually moving, arriving and departing so we need a new paradigm that can cope with this evolution which is hard to be applied in the current traditional network.

- **Exposed Complexity**: Network management task became much harder in today's large-scale networks this complexity due to the distributed low-level network configuration interfaces and by the coupling between the management, control and forwarding planes where many control and management tasks are implemented in hardware.
- **Heterogenous Network Devices**: Today's networks are composed of a numerous number of bare metal devices and appliances including routers, switches, middleboxes. Each needs to be configured according to specific vendor's protocol, therefore increasing inability in managing the network.

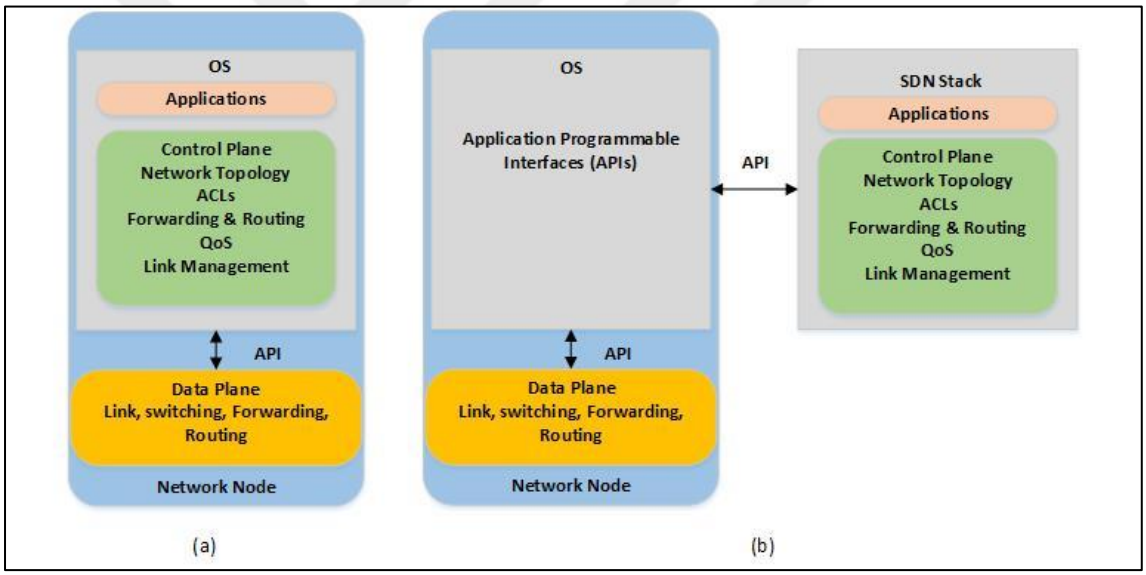

Figure 2.2. *SDN vs traditional network: a) traditional network (each network element has its own control and management plane in one entity); b) SDN network (control plane is extracted from the network node)*

As a solution to achieve easier network management and reduce the tiring workload of device configuration tasks. SDN framework raises the level to simplify and improve network management and offer the opportunity to innovate and customize the behavior of the network easily and efficiently without worrying about the complexity in configuration and control the network according to high level language expressed as centralized programs. Table 2.2 illustrates brief differences between SDN and traditional network properties.

Table 2.2. *SDN vs traditional network*

SDN	Traditional Network
Centralized control and management of the network	Non centralized control (distributed control)
Separation of data plane and control plane	Data plane and control plane merged in the same entity
Wide visibility of the entire network	Non wide visibility of the entire network
Vendor agnostic	Vendor dependent
High level programming languages for configuration (open interfaces)	Low level programming languages for configuration

2.3 Objectives of SDN

SDN have a big pledge in terms of network and operation simplifications with reducing the total cost (CAPEX) and operational costs (OPEX) for the enterprise by introducing the concept of programmable networking services [24]. However, there are several benefits of SDN to realize [25]:

- **Faster Network Business Cycles**: SDN concept provide automation and network programmability of network operations which will help to reduce the response time of business requests to network and service providers. For example, increasing customer satisfaction rate or reduce the payback time of investment.
- **Acceleration of Innovation:** Through the networking flexibility introduced by the SDN in managing the network, this will accelerate business/technical innovation in networking world.
- **Accelerated Adaptation to Customer Demands:** SDN notion through its ability of dynamic negotiation of network service and the dynamic resource control to facilitate customer's connectivity and offer the on demands services requested.
- **Improved Resource Availability and Efficiency:** SDN and network virtualization are tied together nowadays, due to the introduction of automation and service delivery that can be provided using SDN framework this will help to improve network resource availability and efficiency.
- **Customization of Network Resources:** SDN open many doors for networking adjusting and customizing through the APIs to manipulate and include dynamic

implementation of a set of rules and policies such as quality of service (QoS), switching and routing, traffic engineering, security, etc.)

In order for the SDN to fulfil all the aforementioned objectives, some requirements have to be mentioned as the following [26]:

- Support of programmability and automation of network resources.
- Support NFV (Network Functions Virtualization) and services.
- Separation between control and network resources.
- Provides APIs for manipulating and adjusting the behavior of the network resources.
- Support abstractions layers of the underlying network resources by utilizing data models (e.g. YANG) and standard information.
- Support orchestration, management and control of network resources and applications.

2.4 OpenFlow

OpenFlow, is one of the most important South Bound (SB) protocols associated with the SDN architecture. OpenFlow is an open network protocol found to interact with forwarding behaviors of switches from multiple vendors. This protocol provides us a way to control the forwarding devices in the data plane, through this protocol we can manipulate the flow tables dynamically and programmatically in different switches and routers. It is considered the first protocol used and it is a key for many SDN solutions.

Openflow was the result of a six years of research work between Stanford University and California University (Berkeley). Here are the development events related to OpenFlow (OF). Firstly, OF was developed at Stanford University and the first version OF 1.0 was announced in 2009. March 2011 the ONF was initiated. In Oct. 2011, first Open Networking summit was hosted by Stanford University and OF version 1.1 was released. In Feb. 2012 OpenFlow 1.2 was released. In June 2012 OpenFlow version 1.3 was released. However, OF version 1.5 is the latest version available for using in networking industry [10], [27]. One of the most successful and wide deployments of OpenFlow is Google's B4 private backbone network. In this big project, OpenFlow was deployed in 2013 to connect Google's data centers across the globe [28].

2.4.1 The OpenFlow Protocol

OpenFlow protocol standardize the communication language between the controller and the forwarding devices (routers-switches) depending on exchanged messages, these messages composed of instructions of how the switch or router should manage certain packets and collects statistics of these flows. These messages are divided into primary groups; controller to switch messages, asynchronous messages, and symmetric messages. The communication between the controller and switch are done through an encrypted channel, Transport Layer Security (TLS) which connects by default on TCP port number 6633. Table 2.3 gives some description about the controller messages types [28].

Table 2.3. *Description of controller message types*

Controller Message Type	Description
Controller to switch messages	These messages are initiated by the controller and used to handle and manage the switches, and to collect some information from the switches. (e.g., read state, modify state, send packet, barrier request/replies, configuration and features messages.
Asynchronous messages	These messages give the switch the opportunity to talk to the controller without a former permission for communication (e.g., port status, packet in, flow removed and error messages)
Symmetric messages	These are two ways messages sent by the controller or the switch to each other some used at start-up of the communication between the controller and switches (e.g., Hello, Echo and vendor messages).

An OpenFlow switch consist of one or several flow tables used to determine how to direct the incoming flows (packets). These tables are considered as look up tables, so the switch will be informed of how to deal with packets. When a new packet arrives, the switch will look for flow entries that match the incoming packet. If there is no matching for this flow, the packet will be sent to the controller as a '**Packet_In'** message. According to the controller's application it will decide what to do with this particular flow for

example, it might be dropped, forwarded to another flow table or sent back to the sender side. The controller then will install some flow entries to let the switches know about the decision of how to manage the current packet. While waiting for the controller decision. Flows that do not match the flow table's entries will wait in a temporary buffer. If the buffer is full the newly incoming packets will be directed to the controller and if the link to the controller is busy, the packets will be dropped. Flow entry is needed to process and identify packets. Table 4.0 illustrate the components of the flow entry for an OpenFlow version 1.5 (OF 1.5.x).

Table 2.4. *Components of a flow entry*

Match fields	Priority	Counters	Instructions	Timeouts	Cookies	Flags

The match field represent a field against which a packet is matched. A single packet may match various entries in flow tables, for this particular situation, the flow entry with the highest priority will be processed first. As for instructions they describe the OpenFlow processing needed when a flow matches a flow entry. Additionally, it contains the set of actions that will be applied to a flow (e.g., qos set queue action) or modification of the pipeline processing (e.g., direct the packet to another flow table). However, the switch will start the lookup function from "**table 0"** in the pipeline. If there any instruction in the flow entry that say "**go_to"** so the packet will be directed to the next flow table for processing. This multi tables and pipeline processing will increase the level of flexibility in matching and processing the packets [29]. As mentioned before Openflow works on a secured communication channel between the controller and switch, the switch might be attached to one or several controllers for redundancy using various OF connections. The OpenFlow switch could be hardware or software based [24]. Software based such as **CPqD (ofsoftswitch13)**. For some period, the software-based switches have struggled to achieve as good performance as the hardware counterparts.

Open vSwitch (OVS) [30], is one of the most famous software-based switches and it is a multilayer software-based switch, accredited, licensed under the open source Apache 2.0 and is backed by Linux Foundation. The current release supports many features and supports many protocols and technologies including quality of service (QoS) configuration, policing, VLAN, NetFlow, sFlow, etc. Open vSwitch support the OVSDB protocol for configuration management which we will discuss in the next section. As for

the hardware switch it is a dedicated entity with networking abilities. In 2016, HP and IBM have released a dedicated OpenFlow hardware for switching purposes. Other manufacturers like Cisco support OpenFlow protocol in some of their devices.

2.4.2 OVSDB Protocol

This protocol allows us to interact to devices speaking the OVSDB language protocol. First SDN controllers did not support the OVSDB plugins in their framework. OVS present two different interfaces, OpenFlow for control purposes and another protocol was needed for management purposes. Therefore, in 2013, OVSDB was introduced as a management protocol for OpenFlow switches specially OVS hardware and software-based switches along with other protocols such as OF-Config. OVSDB handles switches operations such as configuring QoS policies, dynamic or basic queues configuration, create and delete interfaces, ports, tunnel and collect statistics. To explain more, OpenFlow protocol has the ability to link a flow to a certain queue to ensure some level of quality of service (QoS), but it does not give the ability to create or configure queues. OVSDB also stores information about the physical interfaces of the switch. In addition, it stores the flow table configuration and some monitoring protocols such as sFlow and NetFlow [31]. Open vSwitch consist of three main parts; Open vSwitch switch daemon (ovs-vswitchd), database server (ovsdb-server) and kernel module. The ovsdb-server store the configuration information for the switch and persist states across reboots. The ovs-vswitchd is a core system component, it manages multiple bridges configured on hosts and it is involved in datapath processing for example, monitors the database for modification of information like add, delete, etc. Kernel module responsible for packet processing in kernel. Since the controller communicates with switches through OpenFlow channel, OVSDB server speaks with the OVSDB-Manager. If any changes happened to the database it is also applied to the switch [32], [33]. The configuration in the OVSDB is perpetual meaning that the switch will not lose its configuration in case of switch reboot because of the ovsdb-server. Many SDN controllers have integrated plugins to talk to the OVSDB. OVSDB is a schema-based approach defined in the device or switch, the configuration will be stored in database format [34].

2.5 SDN and 5G Network

The incorporation of the Network Function Virtualization (NFV) and Software Defined Networking (SDN) has raised many approaches and solutions for the next

generation network (5G) in many fields. Quality of Service for example has introduced many characteristics for the next generation mobile networks such as ultra-reliable communication, high data rates, bandwidth on demand and low latency [35], [36]. These attributes can be implemented to the mobile backhaul and the core network of both 5G radio access network or current 4G architecture access network. Many solutions have been proposed to address QoS traffic management, QoS traffic detection and on demand QoS. Taking advantages of Traffic Engineering (TE) techniques provided by the SDN concept, many solid solutions can be implemented in the scope of QoS. 5G networks are developing to integrate the NFV, and to be SDN based networks by the decoupling between Control Plane (C-Plane), and the User Plane (U-Plane) for the network devices. In [35], the author presented QoS architecture implementation for control plane of the mobile network entities using the concept of NFV in centralized manner. While data plane was applied as distributed way through simple elements managed by the SDN controller. The proposed control plane design implements the control element using the NFV concept, and using the concept of SDN adds to the architecture programmability and flexibility in detecting traffic with QoS requirements. To this end, the flexibilities provided by the SDN, NFV assure quick deployment of the network which required in our new era of information explosion, and enable adaptability and integration of new network services. They also reduce the Capital Expenditures (CAPEX), and Operational Expenditures (OPEX), and provide energy efficiency, resource reservation and reconfiguration which represent the main requirements of the current 5G networks.

2.6 Quality of Service

Quality of Service (QoS) concerns about providing end-to-end guarantees to the end user and it is used in networking to ensure best quality performance and quality of experience for some kind of sensitive applications such as VOIP, online gaming, video conferencing, etc. QoS in SDN is an area under research and represent one of the main issues in today's network. It can be measured through different QoS parameters such as bandwidth, delay, jitter and packet loss. The centralized network architecture provided by SDN allows the controller to easily obtain the global view of the network, thus providing flexibility in designing and configuring QoS [37].

The Internet Engineering Task Force (IETF) categorize two main QoS architectures; Integrated Service (IntServ) and Differentiated Service (DiffServ) [38], [4].

Rate guarantees can be classified into two categories: hard guarantees and soft guarantees. Hard guarantees are inflexible or in another word is rigid system, where there is a fraction of the bandwidth that can be used only be a particular flow, so other flow requests will be rejected, hence it is similar to connection-oriented communication like the circuit switched network. Soft guarantees otherwise, is more flexible but does not provide strong guarantees.

2.6.1 Integrated Service Architecture (IntServ)

Integrated services mean that we have guaranteed services, i.e., if all packets coming to the network should not have delay more than 10ms, the network has to comply to that and make sure that will happen. İntegrated service is considered one of the first attempts to deploy the concept of QoS in IP networks. IntServ targets real-time applications to provide hard QoS. The concept of IntServ allocates resources for a specified flow all the way from source to destination. İn other word, it emulates the resource allocation concept of circuit switching. İn this case every individual router in the network should take care about the Service Level Agreement (SLA) through preservation of resources inside the router, for that purpose the routers should coordinate with each other. That why this type of architecture is not scalable because we have millions of routers and that Internet Service Providers (ISPs) have to coordinate with each other. Figure 2.3 shows an illustration about the integrated service architecture in a router and shows the routing protocols and quality of service associated protocols as follow [38]:

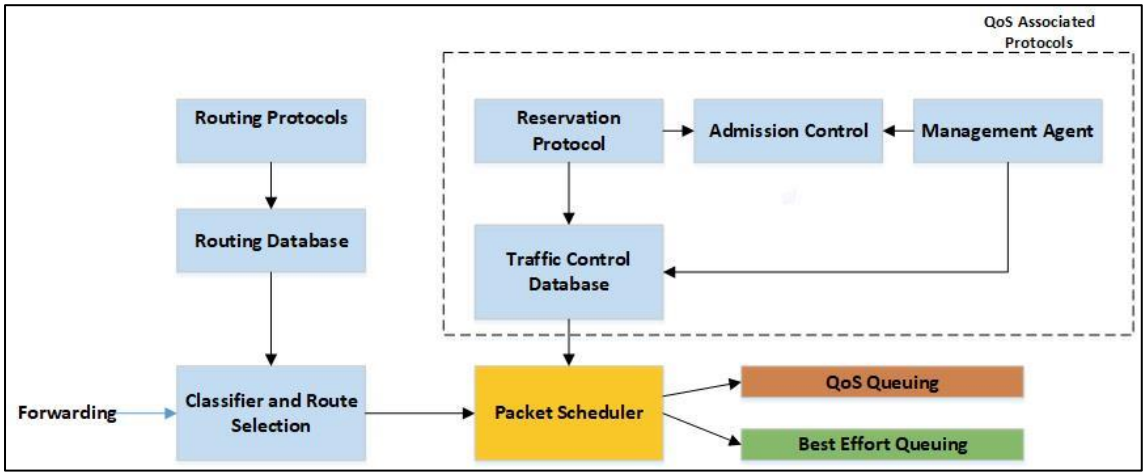

Figure 2.3. *Integrated service architecture in a router*

Admission Control: executes the decision algorithm that a router or host uses to determine whether a flow can acquire the requested QoS without affecting earlier guarantees. It ensures that the packet going inside the network meets the required QoS. For example, when we inject flow in the Internet, we need to make sure the flow gets the required service from the Internet so if we do not get that service the packet will be dropped.

Management Agent: manages different functionalities of QoS like traffic shaping, traffic policing, etc.

Traffic Control Database: it tells how the packets need to be treated by the network.

Classifier and Route Selection: it will classify the packets into one of the available traffic classes, based on that it will select the route by looking to the routing database. In general, it determines the needed QoS.

Packet schedular: it will get the input from the routing information, and traffic control database about how the packets need to be treated and push it to the required queues.

Routing Protocol: it tells which interface the packet need to be routed to.

Resource Reservation Protocol (RSVP): a network control/signaling protocol that allows data receiver to request a special end to end quality of service for its data flows. It is a network control protocol, not a routing protocol i.e., it works with association with routing. RSVP request consists of: flow spec together with a filter spec, this pair is called flow descriptor. Flow spec, specify the desired QoS (i.e., what type of quality of service we are expecting from the end user). Filter spec, defines the set of data packets to determine which type of queuing mechanism we wish to apply on the packets such as priority queuing, fair queuing, weighted fair queuing, custom queuing etc. The flow spec in general used to set parameters in the packet schedulers while the filter spec is used to set parameters in the packet classifier.

The main benefit of Integrated Service Architecture (ISA) is that it provides service classes differentiation, users have the ability to define traffic type and the usage of the service that fits to their applications. The main drawback of ISA is scalability problem. Since IntServ uses RSVP protocol that means all routers have to coordinate with each other and the RSVP requires large end to end signaling overhead to maintain per

flow soft state at every node along the path from source to destination. In addition, IntServ has to store all flows information in routers/switches along the path, which becomes expensive in the network with many flows. Due to all of these reasons this architecture has never been applied on the Internet and only applied to some local enterprise networks.

2.6.2 Differentiated Service Architecture (DiffServ)

Scalability issue was a problem introduced in the IntServ, for that matter a solution had to be provided to overcome the scalability problem issued by IntServ. Here is where DiffServ were introduced. DiffServ is coarse grained, class-based mechanism for traffic management. DiffServ implement the Per Hop Behavior (PHB) policy defines the packet forwarding properties associated with a class of traffic. In other words, DiffServ has the ability to handle different classes of traffic in various ways within the Internet in scalable way. DiffServ depends on aggregate flows in which packets are classified using 6-bit Differentiated Service Code Point (DSCP) in the 8-bit Differentiated Service field (DS field) in the IP header. Packets with the same DSCP bits in the IP header are treated the same no matter to which flow they belong. Unlike IntServ which depends on per-flow state (i.e., end-to-end). DiffServ is not like IntServ which require coordination between routers, here the only coordination between the DS domains and this coordination is done by Bandwidth Broker (BB) [4].

Bandwidth Broker, is an agent that has some knowledge of an organization's priorities and policies, it allocates QoS resources with respect to those policies for example, it allocates a pre-determined amount of bandwidth. The DiffServ architecture composed of set of functional elements [39]:

- **Edge Function (Traffic Conditioning and Packet Classification)**: Consists of a set of control functions that is applied to classified packets streams in order to enforce traffic conditioning agreement, which are made between customers and service providers. It has four components: meter, marker, shapers and droppers. Meter is used to measure the classified traffic stream against traffic profile. The state of meter may then be used to enable remarking, forwarding, shaping or dropping actions. However, meter used to estimate how much quality of service we have already got and what is the level of QoS we have to give for the next hop to the next DS domain.

- **Core Function (Traffic Forwarding)**: When a marked packet arrives at a DiffServ router, the packet is forwarded to the next hop based on PHB that used to specify which kind of behavior that need to be done for a particular packet class at that particular hop. PHB is strictly based on class marking (no other header fields can be used to influence PHB). Big advantage, no state info need to be maintained by the router because whatever marking we want, all done on the edge and when the packet comes into the network or DS domain, the core routers look to the marking happened at the edge routers and based on that mark, the packets will be forwarded appropriately. PHB specified in two categories:
- **Expedited Forwarding (EF)**: Specify that packet departure rate of a class of traffic must exceed or equal a configured rate. This is generally dedicated to low priority, low loss, low latency traffic. However, it maintains a minimum guaranteed rate. It implies isolation i.e., provides guarantee for the EF traffic in which it should not be influenced by other traffic classes. In addition, the traffic is admitted based on peak rate at the edge.
- **Assured Forwarding (AF)**: Divides traffic into four classes with some bandwidth and buffers allocated to them. The intent here is to implement services that differ relative to each other (e.g., bronze traffic, silver traffic and gold traffic, etc.) within each class there are three drop priorities that affect which packets will be dropped first if there is congestion. For example, if we have low priority traffic like bronze traffic it will be the first to get dropped if there is a congestion. Next one to be dropped is silver and then the last one to be dropped is gold traffic.
- **Default PHB**: This is simply best effort services. In other words, if the DSCP field labeled with 000000 the traffic will be treated as best effort traffic (no special treatment).

2.6.2.1 DiffServ vs IntServ

Table 2.5 demonstrate a comparison between the two QoS architectures with respect to many parameters and give overview about the compared items.

Table 2.5. *DiffServ vs IntServ*

Compared Items	DiffServ	IntServ
Manageable Unit	Class	Flow
Router's Capability	Edge and Core	All in One
Defined in the Standard	Forwarding Behavior	Service Type
Guarantee Required	Provisioning	Reservation
Work Region	Domain	End-to-End

The main advantage of DiffServ architecture is scalability, meaning that there is no dynamic change of state, hence the communications between the routers are reduced and it can be deployed for specific domain independently each domain has its own DS mechanism. In addition to that, load on the routers has been divided between edge and core routers in which they are easily adjustable to Service Level Agreements (SLA). On the other hand, there are some disadvantages; firstly, it is not a real end-to-end QoS i.e., it does not provide hard QoS because it depends on Per Hop Behavior (PHB). Secondly it has a limited number of classes so specific flow cannot be isolated. Lastly, there is no dynamic admission control.

One of the major uses of DiffServ in today's networks is a combined technology of DiffServ and Multi-Protocol Label Switching (MPLS), which will help to enable path protection and restoration to create end-to-end specific path, hence the combination between the two technologies will give the ability to give strict E2E (End-to-End) QoS guarantees while optimizing the use of network resource.

2.7 Related Work in QoS in SDN

Many research studies have been carried out in the topic of QoS in the scope of SDN [2], [3], [6], [40]-[49].Many models have been explored, to improve performance and QoS in SDN ranging from monitoring to resource reservation to different routing algorithm. There are no specific architecture and standards like the DiffServ and IntServ in SDN. It can be seen as providing flexibility for network administrators to apply their own QoS algorithms.

In [2], the authors provide a comprehensive picture of QoS related literature in SDN OpenFlow networks. It delivers an inclusive review about the QoS efforts in SDN as demonstrated in Figure 2.4 which summarizes the categories and the studies that has been taken in the QoS domain under the SDN context. Also, it will show our aim in our study which focusing on resource reservation and queue management.

Chenhui Xu et al in [3], presented a novel mechanism for serving QoS by dynamic resource allocation. The authors demonstrated a framework for implementing QoS by classifying flows into different classes and then applying priority queueing to allocate required resources to appropriate flows. In order for this approach to work, the controller must have a good knowledge of the network utilization state including load, delays, and jitter. They created a separate thread to periodically monitor the network utilization then implement a priority queueing technique. Flows are divided into different levels then assigned to particular queues based on their priority, as long as the link of the given queue can satisfy the guarantee requested by the application. They test their mechanism on simulated network on Mininet and also on physical network with real hardware switches and controllers.

Govindarajan et al in [6], proposed a QoS controller named (Q-Ctrl) which involves five main operations including queue configuration, QoS flow deletion, insertion and modification, queue deletion.

In [40], the SDN paradigm is applied to provide bandwidth on demand for inter-datacenter networks. The bandwidth allocation fairness among different traffic classes and the increase in network utilization are evaluated. The results show significant improvements in network throughput, over traditional technologies such as MPLS-TE. The classification of traffic in the data plane is based on priority queueing.

In [41], Cosmin Caba, Jose Soler created an API for dynamic configuration of QoS resources in the network devices by using the aptitudes of the OVSDB, furthermore their work demonstrated the possibility to develop network services with granularity on top of fine granular services exposed by QoS configuration API at the SDN controller. The work show how priority queueing plays good role in dealing with QoS programming in SDN architecture. The results and the tests are done using testbeds and Floodlight SDN controller.

In [42], a novel approach for QoS architecture for home broadband access networks to configure QoS based on applications and devices called FlowQoS. It uses the concepts of SDN to forward traffic through the appropriate rate shapers. In FlowQoS the user should simply specify the maximum allowed bandwidth for specific high-level applications in home network using a web configuration tool.

In [43], the authors proposed HiQoS model that examine the implementation of DiffServ architecture and multipath QoS routing techniques in OpenFlow networks. The system divides traffic into three categories, low delay multimedia applications, video streaming and best effort data stream. The three types of traffic are forwarded via queues with limited rates. The routing paths then calculated using the existing bandwidth utilization as weight, and flows from the same class might use different paths. In general, the system provides bandwidth guarantees per class level rather than flow level guarantees.

In [44], the authors proposed another OpenFlow QoS framework namely, OpenQoS to achieve E2E QoS for multimedia applications. OpenQoS classifies the traffic into QoS flows for multimedia delivery and non-QoS flows. The multimedia flows use dynamic routing algorithm, whereas the other non-QoS flows are following the shortest routing path algorithm. Unlike other OpenFlow-based QoS frameworks, OpenQoS does not employ the queue mechanisms, and other resource reservation techniques. This work extended to a distributed architecture for multimedia streaming which presented in [45].

In [46], the author attempts to attain QoS in OpenFlow-based networks by utilizing QoS routing and rate guarantee. This work, introduces the use of the vendor agnostic interfaces such as the OpenFlow (OF) and the OVSDB to implement the QoS framework proposed. The work proposed in [47], presents an extension to SDN controller (floodlight) in a way to manage the queues in OpenFlow switches and illustrate further possibility of developing groundbreaking on demand resource reservation mechanisms in SDN without adding intolerable overheads. In addition, the architecture is based on a client/server model that comply with the OVSDB protocol, the client will play the role of sending messages according to the OVSDB, whereas the server will play the role of receiving the requested commands and configuration and it is done by OpenFlow enabled switch.

In [48], the authors proposed high level QoS framework extension to Ofelia which is considered one of the large-scale OpenFlow testbeds, the work shows adding QoS capabilities, performance measurements. The framework exploits the queue tables in OpenFlow to apply QoS control in various types of OpenFlow-based switches. The module focused on how to create NBI to manage and work with different hardware switches with non-uniform queue application.

In [49], the authors present a study to show the performance of dynamic queue management in SDN enabled networks. Dynamic queue management considered one of the most widely studies conducted by researchers to implement QoS. The paper shows that change in network resource allocations could be beneficial to some kind of applications. There has been some previous work analyzing the performance, but the most similar to this work uses the Mininet emulator for the study.

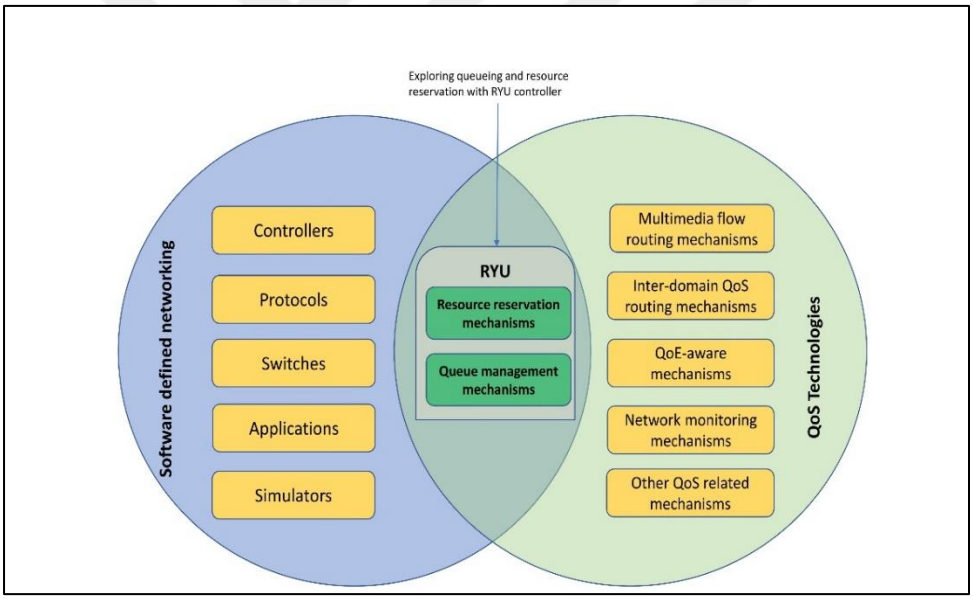

Figure 2.4. *QoS technologies in the context of SDN*

3. PROPOSED QUALITY OF SERVICE ARCHITECTURE

This section provides an introduction to our proposed QoS architecture and the control modules used to provide network intelligence, background on Quality of Service (QoS) support in OpenFlow and queue management, Linux Traffic Control (TC) and the Hierarchical Token Bucket algorithm (HTB). Lastly, we will give an overview for RYU controller architecture and the measurement tools used in this thesis.

3.1 Proposed Architecture

The main goal of the proposed QoS control module is to provide network intelligence, and to keep an updated network state so that the necessary allocation of the user defined resources can be made. To enable the abstraction of the underlying network infrastructure, control plane and forwarding plane have been decoupled to give the flexibility to apply the techniques required for bandwidth allocation for end-to-end quality of service similar to [50].

The QoS control modules consist of two main components: the QoS SDN application and the SDN controller. The QoS control modules implement two interfaces, northbound and southbound interfaces using the OF and OVSDB protocols. These interfaces grant our module to be flexible to enable network configuration based on application's and user's requirements. The core of the study described herein is the QoS control modules which composed of the SDN-app sub-modules and the SDN controller to provide a set of methods for managing the state of the OVS software bridges with respect to QoS configurations. Note that on each port of an OVS switch, a QoS object can be defined. The QoS object specifies the maximum rate that can be shared among the priority queues configured on that port. The possible parameters for a queue in the OVS database are: the minimum guaranteed rate, the maximum rate, the priority of the queue and the DSCP value. The priority fields indicate how the available bandwidth is shared among the priority queues. If configured, the DSCP value is injected into all the packets transmitted through the queue. Advanced and complex QoS mechanisms (e.g., DiffServ) can be implemented through certain configurations of the above-mentioned parameters as in [50]. The QoS control modules can be considered as an SDN app consumed by internal SDN controller. Figure 3.1 shows the proposed QoS control modules architecture in which four different sub-modules (resource reservation, queue management, etc.) can be seen.

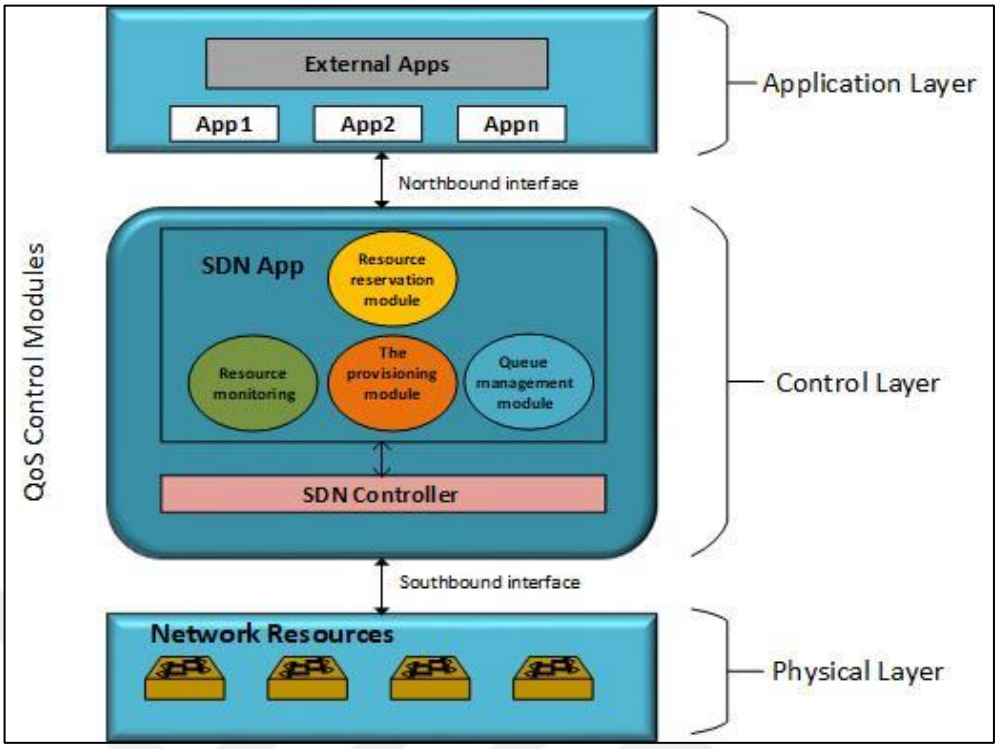

Figure 3.1. *Proposed QoS control modules architecture*

3.2 QoS Control Modules

The proposed QoS control modules architecture in Figure 3.1 consists of a set of generic network sub-modules running on top of an SDN network. They enable to adapt the control plane to fit user requirements in terms of control and management, and to configure the forwarding plane on demand similar to [51]. The QoS control modules implement the following:

a) Resource reservation module: This module provides end-to-end bandwidth guarantees for priority flows by configuring output queues of the network devices similar to [5]. SET_QUEUE action defined within OpenFlow protocol allows packets to be mapped to a specific output queue. Although the controller can send such instructions, queue configuration is itself beyond OpenFlow specifications. OVSDB protocol [34] promises to deliver configuration management functionalities for data plane devices such as configuring queues.

b) The provisioning module: The QoS control modules make use of the provisioning module to allocate the required network resources demanded by the user's applications and guarantees the expected QoS. This module enables the SDN application to customize

QoS to its different types of traffic in order to accommodate them in the network forwarding plane.

c) Queue management module: It maintains the rate limiting of the traffic specified by the user. In case a certain application requests for certain QoS that exceeds Service Level Agreement (SLA), it just drops packets exceeding the certain level defined by the user/admin. In this study, the HTB algorithm is used to achieve this purpose.

d) Resource monitoring: This is responsible for collecting and maintaining information about the current state of the queue's behavior. For example, it observes queue utilization levels for the different types of traffic and bandwidth allocations. In this study, this module is augmented with the function of traffic statistics gathering. The controller is programmed to send special OpenFlow request messages per-port and per-interface statistics every 10 seconds. The key parameter of our QoS module is bandwidth. To measure the utilization of each queue on each port, we need to poll the statistics of the transmitted bytes counter of each queue and keep track of the last polling time since our interval is 10 seconds (i.e., readings for 10 seconds are taken and divided by an interval). The transmitted bytes are stored in matrices and then presented using graphs to show the queue utilization.

Using these sub-modules, the QoS SDN-app builds an updated topology map showing the status of the network resources and the QoS level of the running applications. From this information the QoS control modules configure the forwarding plane by assigning different types of traffic (TCP, UDP, VOIP) to different queues. In other words, the QoS control modules manage the bandwidth allocation to flows on the switches, and the controller manages the mapping across all the flows in a dynamic manner on per switch basis similar to [52].

3.3 Quality of Service in OpenFlow

OpenFlow protocol supports the idea of QoS since the first versions of the protocol. However, in the early stages, the support was limited until new versions has been introduced. For example, OF 1.0 – OF 1.1 supported queues only with minimum rates. OF 1.2 and above started to support queues with both minimum and maximum rates. Then later in OF 1.3, meter tables were introduced to apply more complex QoS operations besides queue tables.

3.3.1 Queues in OpenFlow

Queues were first supported in OF 1.0, with the later versions as mentioned before queues have been supported with minimum and maximum rates. Queues are basically served by scheduling algorithms that enable provisioning of different levels of QoS for different types of traffic flows. The OpenFlow switch consists of two main parts: the first part contains the queues, the frame transmitters and receivers with the flow tables matched therewith, the second part controls the communication with the controller [52]. The first part contains all the essential elements for the transport operations of the packets into the node, and the flow tables are used to direct the traffic into the required and right egress queues. Queues are intended to provide rate limit guarantees on the rate of flow of packets placed in a queue [53]**.**

Queue configuration tasks such as creation, modification and deletion are not managed using the OF protocol, instead they are handled by other switch configuration protocols such as OVSDB and OF-Config. OpenFlow can only help with fetching the queues statistics from the switches. Using the following command in Linux Command Line Interface (CLI) for OVS, the queue statistics can be fetched as shown in Figure 3.2.

~$ sudo ovs-ofctl -O openflow13 queue-stats s2

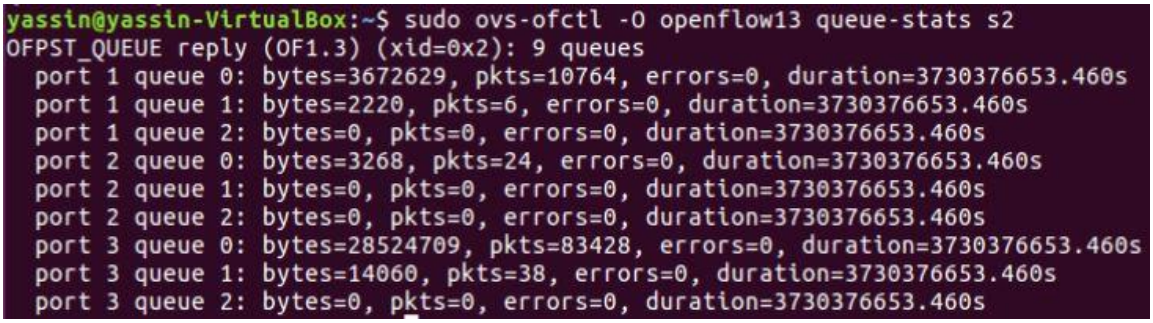

Figure 3.2. *Queue statistics fetched from switch 2*

The QoS constraints are applied to ports. Hence, port tables in OVSDB are related to interface table and QoS table as shown in Figure 3.3. The relation explains that all ports should have an interface. The switch port may or may not have QoS rule attached to it. In general, the port can have at most one QoS rule configuration, while QoS rule can have various queues attached to it.

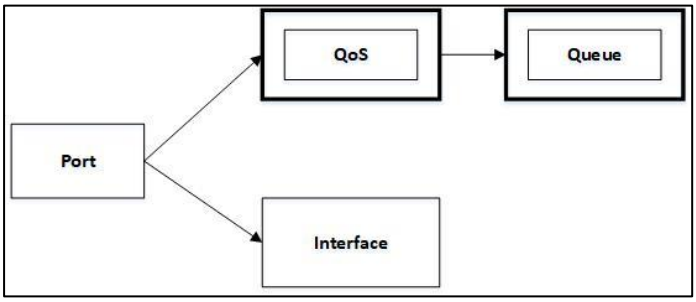

Figure 3.3. *OVSDB database schema in OVS*

However, the proposed framework applies three different queues attached to a single port for every interface of the switch. Once the QoS rules and queues have been installed on the switch, flows can be directed and mapped to their specified queues using the OpenFlow instruction ***set_queue*** action, in RYU controller this is invoked using the function ***OFPActionSetQueue()*** which sets the queue ID for a packet, then it directs packets that match the flow entry via that queue. One queue can be used to process many flows at the same time. Our approach depends on using high level programming in configuring the QoS rules and queues. On the contrary of some old configurations which depend on CLI programming of OVS. In this study, the RYU OVS bridge library has been used to provide the flexibility to configure QoS rules and to direct flows to the required queues efficiently as shown in the Python code in Figure 3.4.

```
names = ovs_bridge.get_port_name_list()
for name in names:
    port_id = ovs_bridge.get_ofport(name)
    self.logger.info("queues %s", self.queues)
    result = ovs_bridge.set_qos(name, type='linux-htb',
                                max_rate=str(self.PORT_SPEED),
                                queues=self.queues)
```

Figure 3.4. *QoS rules configuration using Python.*

The code above discovers the switches in the topology then assigns the QoS rules to the interfaces and create the queues. The ONF specifies the followings properties regarding queues in OpenFlow switches [28], defining the minimum data rate for a queue. If the minimum_rate is configured then the queue will be prioritized by the switch and any packet forwarded through this queue will struggle to achieve the specified minimum rate at the cost of other flows rates. In case of multiple queues assigned to one port with total minimum rate higher than the capacity of the link, the rates of the queues and the capacity will be shared in proportion to the queues minimum_rate. On the other hand, maximum_rate of the queue represents maximum data rate allowed for a queue. When the

actual rate of flows using the queue is higher than the defined maximum rate, the switch will then delay or drop packets to maintain the maximum rate level. To implement queues on the OVS switches Linux Traffic Control (TC) is used to achieve the implementation of queues and manage the traffic among the queues as described in the following sub-section.

3.3.1.1 Linux Traffic Control in OpenFlow

Traffic control is used to make full use of the network connections by using sets of queuing systems and mechanisms to transmit and receive packets on a router. This involves determining at which rate and whether to accept packets on the input interface. Since the simplest form of traffic control consists of a single queue combined with other mechanisms for collecting and dequeuing packets. Queues (buffers) are used in traffic control. In general, queues are used to manage the outbound data traffic, rate limit and prioritize traffic. In other words, queue is a way to organize and rearrange pending tasks or data flows waiting to be transmitted by the hardware. The simplest queuing discipline (qdisc) for Linux is "pfifo_fast" which considered more sophisticated than FIFO. Some of the traffic control solutions are: a) limit bandwidth to a specified rate, b) reserve bandwidth for a particular user/application, c) ensure that a particular flow is dropped. Linux Traffic Control (TC) is a utility program used to configure traffic control and apply scheduling in Linux kernel. TC consists of set of controlling components such as shaping, scheduling, classifying, marking, dropping and policing. Queuing discipline is the key stone of Linux traffic control. There are two kinds of queuing discipline: classful and classless queuing disciplines. In this study, Hierarchical Token Bucket (HTB) classful queuing discipline is used in OVS. HTB is a queuing discipline which uses Linux traffic control components such as filters and classes to do more sophisticated queuing discipline another classful qdisc type is Hierarchical Fair Service Curve (HFSC) [54]**.** The libraries in RYU controller support TC operations, particularly for Open vSwitch programming.

3.3.1.2 Hierarchical Token Bucket (HTB)

Tokens and buckets are widely used in the concept of HTB in which HTB can perform various traffic control techniques. Token term is related to shaping or rate limiting. One of the most uses of HTB is to control and shape the transmitted traffic to specific rate (i.e., control the outbound bandwidth). In HTB the generation of tokens are done at a fixed rate then these tokens are collected in a bucket. HTB is widely used and

one of the most important techniques in traffic control for bandwidth sharing. The main difference between HTB and HFSC, the latter is concern about delay it balances delay for sensitive traffic against other traffic. Since in our tests we only concern about bandwidth and not delay, HTB will be used. HTB is used in this study to limit the rate of the traffic according to the link capacity. Since the HTB uses hierarchical model we have our main link and three leaf classes represent our traffic that we need to limit. One can ask the following questions: why we need to limit some traffic in general? Suppose we have a 10GB/s link coming from the Internet and is entering a LAN network of link capacity of 10MB/s. In order to accommodate to the link capacity, we need to shape the traffic. This is where the HTB comes into the play. To get the most utilization of our 10MB/s link, the bandwidth must be shared among various applications and user's needs. In HTB, shaping occurs at leaf classes only. In this study, we have three kinds of classes and the shaping occurs at the level of these classes only (i.e., no parent classes and borrowing). The HTB classes have some class parameters: a) rate which represents the minimum desired rate guaranteed for a class and its children, it is equivalent to Committed Information Rate (CIR), b) ceil rate which is the maximum speed to which class limit the transmitted traffic (this is used in borrowing model of HTB), c) burst, represents the size of the bucket. HTB will release burst bytes before waiting more tokens to be generated [55].

3.4 RYU Controller Architecture

RYU is an opensource component based SDN framework, which is fully written in Python. It was first used by NTT (Nippon Telegraph and Telephone) a well-known Japanese telecommunication company. All of the codes are freely available under the Apache 2.0 license. RYU means flow in Japanese. RYU introduces software components with well-defined APIs for developers to innovate and create network management and control applications. RYU supports many southbound and northbound protocols such as NETCONF, OpenFlow, OVSDB and OF-CONFIG protocols. The SDN application written in this thesis uses OF version 1.3 and the OVSDB protocol. RYU controller provides a wide variety of built in functionalities and applications such as topology discovery, snort, GRE tunnel, VLAN abstraction, etc. Topology discovery is used as the built-in application in this study to learn and discover links between the switches and to report to the controller. This functionality uses LLDP (Link Layer Discovery Protocol) in RYU by polling the observe link flags in the controller. Figure 3.5 demonstrates the architecture, APIs, and applications supported by the RYU controller.

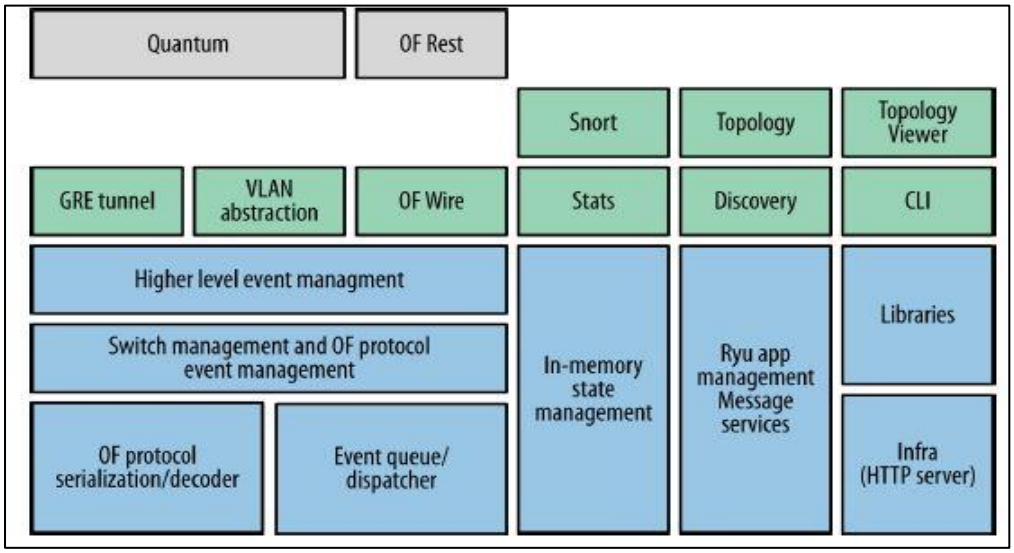

Figure 3.5. *RYU architecture, applications and APIs* **[56]**

As mentioned before, RYU supports many functionalities and applications that are called while applying some functions like topology discovery which discovers the switches and links and the topology structure of the network. However, this is done by calling events. Events are messages exchanged between applications. Each built-in RYU application has a receiving queue for the events. This queue is simply a FIFO queue which preserves the order of the events. The OpenFlow controller part of RYU automatically decodes OpenFlow messages received from the switches and sends those events to RYU application. For example, "EventOFPacketin" is used to inform the controller that packet-in message has been sent from the switch to the controller to make a decision about undefined flow entry in the OpenFlow tables. In addition, it learns about the MAC addresses of hosts and to which ports are connected. Figure 3.6 how the prompt window of RYU controller during the processing time.

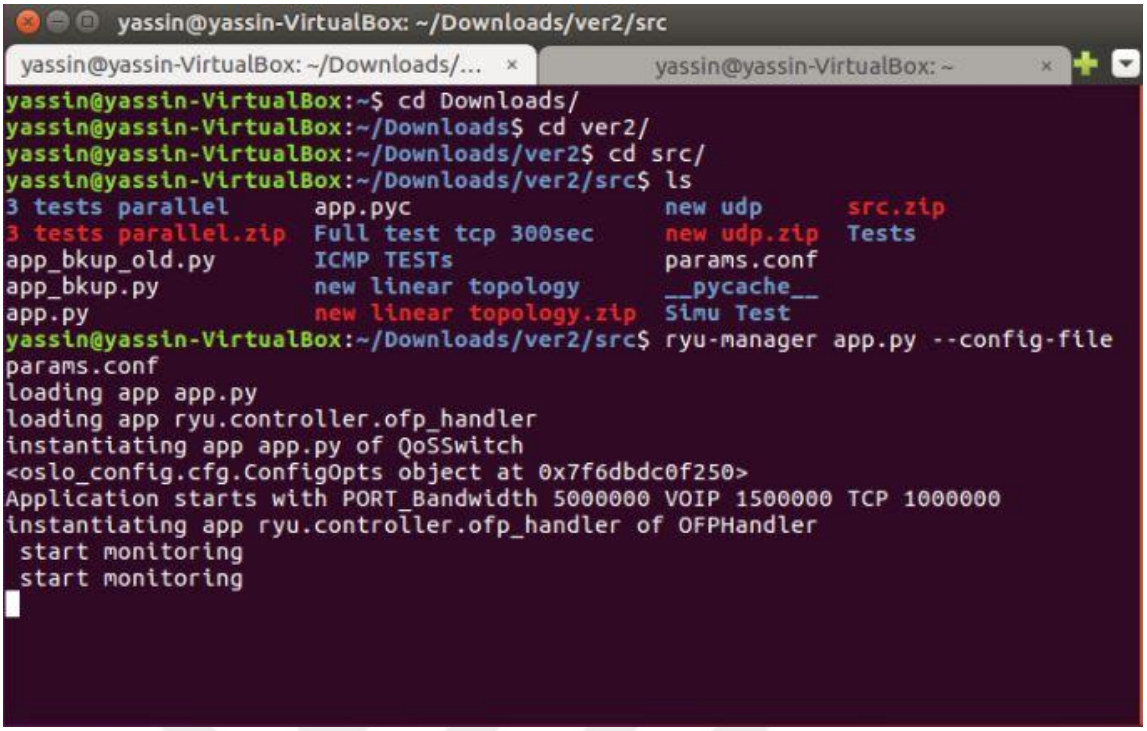

Figure 3.6. *RYU prompt window during processing*

The controller is the main governing entity in the whole SDN ecosystem. Network intelligence is centralized in the controller which maintains the logical view of the entire network. Physically, the controller is a software that runs on a regular computer and connects through dedicated logical interfaces to the routers or switches [57].

Figure 3.7 illustrates a simplified view of how the proposed RYU QoS SDN-app interacts with the modules to achieve resource reservation, resource monitoring, queue management and provisioning.

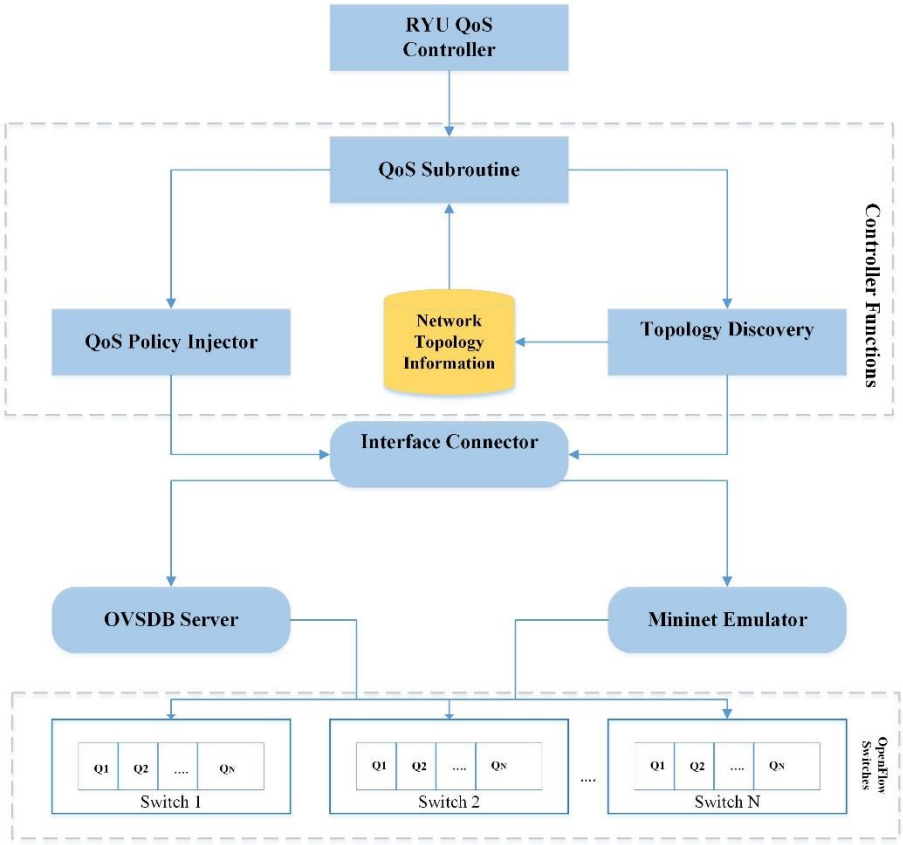

Figure 3.7 *Proposed controller subroutine functions.*

The QoS subroutine receives the network topology information (i.e., switches, links, hosts, etc.). The topology discovery component interacts with the network topology information repository. The topology discovery holds the entire network information. The QoS subroutine also invokes the QoS policy injector which schedules the user defined QoS constraints (i.e., the desired bandwidth) and injects the required QoS flows in the OpenFlow enabled networking devices. The interface connector is the entity responsible for integrating the controller to the OVSDB server and Mininet emulator. Our proposed framework uses RYU as the SDN controller. This controller is able to control an OpenFlow network and satisfies the user's application network requirements, and the user defined QoS constraints for traffic shaping. Note that primary duties of the controller are [6] : a) find and uncover the topology information; b) discover switches' aptitudes: supported OF communication protocol, ports, queues, etc.; c) manage RYU modules such as threads, event-handlers, memory and switches; d) retrieve network statistics: provide an interface to fetch network statistics via OpenFlow protocol; e) procure the rule to map into the switches since the SDN app sends OpenFlow rules to the controller; f) deliver the acquired statistics to the monitoring system.

The proposed QoS framework can be applied to small to mid-size topologies required by the admin/user, let alone, the proposed architecture comes with a designed real time monitoring system to track the resource usage (queue utilization) and reports back to the controller to generate the graphs for monitoring all data flows to observe the queue performance.

4. IMPLEMENTATION AND TEST RESULTS

This section is divided into following subsections: 1. General program workflow gives a general view of the QoS subroutine program, how it is processed and performed, 2. SDN network setup and emulation process describes how to implement the network topologies and what tools are needed for the network emulation. In addition, it will provide a brief view of how RYU installation and the OVSDB configuration are done, 3. Measurement tools will give a view about the test tools used to generate the traffic and measure the bandwidth, 4. Implementation is subdivided into experiment setup and (experiment & result) which provide an overview of the tests performed and the results obtained from this study.

4.1 General Program Workflow

The workflow of the proposed approach consists of several steps: First, best effort flows and QoS flows have been distinguished based on port number and protocol type criteria. A multimedia application can be identified and distinguished from others based on its common characteristics such as protocol type TCP/UDP, and well-known source and destination port numbers. In this study, a combination of those characteristics is used to distinguish among the three types of traffic (TCP, UDP and VOIP). Second, the controller installs rate limiter in the flow's switches and configures priority queues at the switches then directs the traffic to its destined queues. In other words, we will consider UDP flows as best effort traffic, TCP and VOIP flows as QoS traffic. The controller has to direct the traffic to their specific queues. Figure 4.1 shows the complete flowchart of our framework showing the queue mapping process to direct traffic to its specified queues.

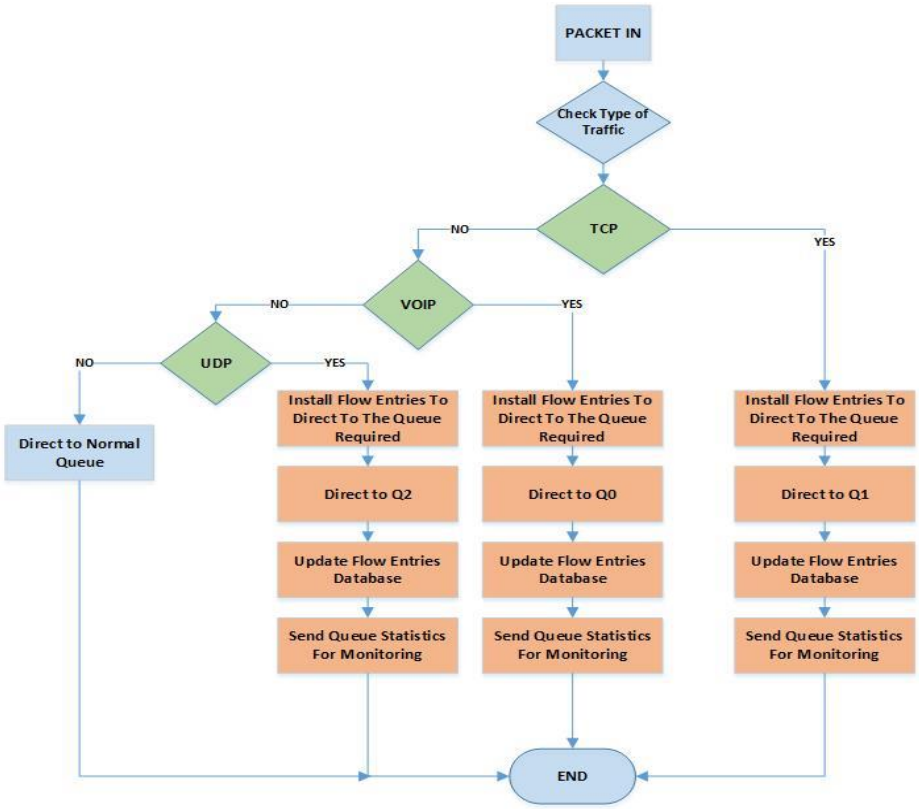

Figure 4.1 *General overview of the workflow of the proposed architecture.*

When a new packet arrives at the switch, the first packet is sent to the controller to check which type of traffic is entering to the switch by checking the port number and the protocol type. If the flows are VOIP traffic, it is directed to queue (q0), while if it is TCP traffic the flows will be directed to (q1), and if the traffic is UDP the flows will be sent to (q2). If none of these flows are injected so the traffic will be sent to a normal queue with no QoS guarantees. For the direction of traffic, the necessary flow entries will be installed on the switch and the database of the flow table will be updated accordingly. After the generation of the traffic, some statistics are sent to the controller for the monitoring process. These statistics include the transmitted bytes coming from the hosts through switches then to the controller. These are fetched and directed to matrices we created to monitor the queues and observe the utilization of these queues using graphs. The monitoring thread will first send **QueueStatsRequest** to the controller to fetch the transmitted bytes, then an interval of 10 seconds (sleeping time) is taken for the monitoring process (i.e., the monitoring thread will be triggered every 10 seconds) and every 50 seconds we present the results via graphs. The average of transmitted bytes on an interval is taken and stored in matrices and then the results are shown using the help of Matplot library in Python.

4.2 SDN Network Setup and Emulation Process

To set up and implement SDN networks, some set of tools must be used. First of all, a network emulator program is needed to create network topologies. To evaluate proposed RYU's QoS architecture a testbed is needed to implement the QoS policies on a network. Working on a real physical network is beyond the scope of our study. Mininet emulator program has been used in this study to replicate the behavior and set up the SDN network. The SDN network topology is emulated on Mininet by creating virtual switches, hosts, links, and a controller.

4.2.1 Mininet

Mininet is an open-source network emulator for deploying large networks on the limited resources of a single computer or VM. It provides convenience and realism at very low cost [58], [59]. Mininet uses Linux process-based virtualization like network namespace and Linux container architectures to emulate many hosts and create complex networks on a single operating kernel. Mininet is also compatible with many virtual software switches like OVS and can be installed easily on virtual machines software (e.g., Virtual Box) as a pre-packed image that can run on multiple platforms such as Windows, Linux and MAC operating systems with OpenFlow tools integrated with it.

4.2.2 Mininet and RYU Setup

The following system specifications are used for the installation process of Mininet and RYU controller in this study as shown in Table 4.1:

Table 4.1. *The used testbed specification*

Category	Specification
Laptop	ASUS K55V
Processor (CPU)	Intel® Core (TM) i5-3210M CPU @ 2.50GHz (4CPU)
Host Operating System (OS)	Windows 10 Pro 64-bit
RAM of Host OS	8 GB (7.5 usable)
Guest Operating System	Ubuntu 16.04 LTS 64-bit
RAM of Guest OS	2048 MB
Mininet Emulator	Version 2.2.2
Virtual Machine (VM)	Virtual Box 5.2.16
Controller	RYU version 4.30

Mininet can be installed directly using the command line in Linux CLI:

```
$ git clone https://github.com/mininet/mininet
```

after this, the required files will be downloaded from the Mininet servers. All the libraries used will be included. Then the OVS version 5.2 on the Mininet will be installed using the following CLI command:

```
$ git clone https://github.com/openvswitch/ovs.git
```

For RYU installation, firstly some python libraries must be installed using the following command applied on Linux CLI:

```
$ sudo apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev
```

then we call the following command also on Linux CLI:

```
$ sudo python -m pip install ryu
```

after these two commands, the RYU controller will be installed on our Ubuntu system. After installing the OVS, we set the configuration for the OVSDB, so some commands need to be typed through the Linux CLI to set the manager address in order to let the controller and the virtual switch (OVS) communicate with each other. To do this, the following commands have been used:

```
$ sudo ovs-vsctl get-manager

$ sudo ovs-vsctl set-manager "tcp:127.0.0.1:6640"
```

The "127.0.0.1" represents the controller IP address, and "6640" the port number to open the communication between RYU and OVS for the OVSDB configuration. These commands are only set for once. Figure 4.2 shows the Mininet prompt window while creating our network topology.

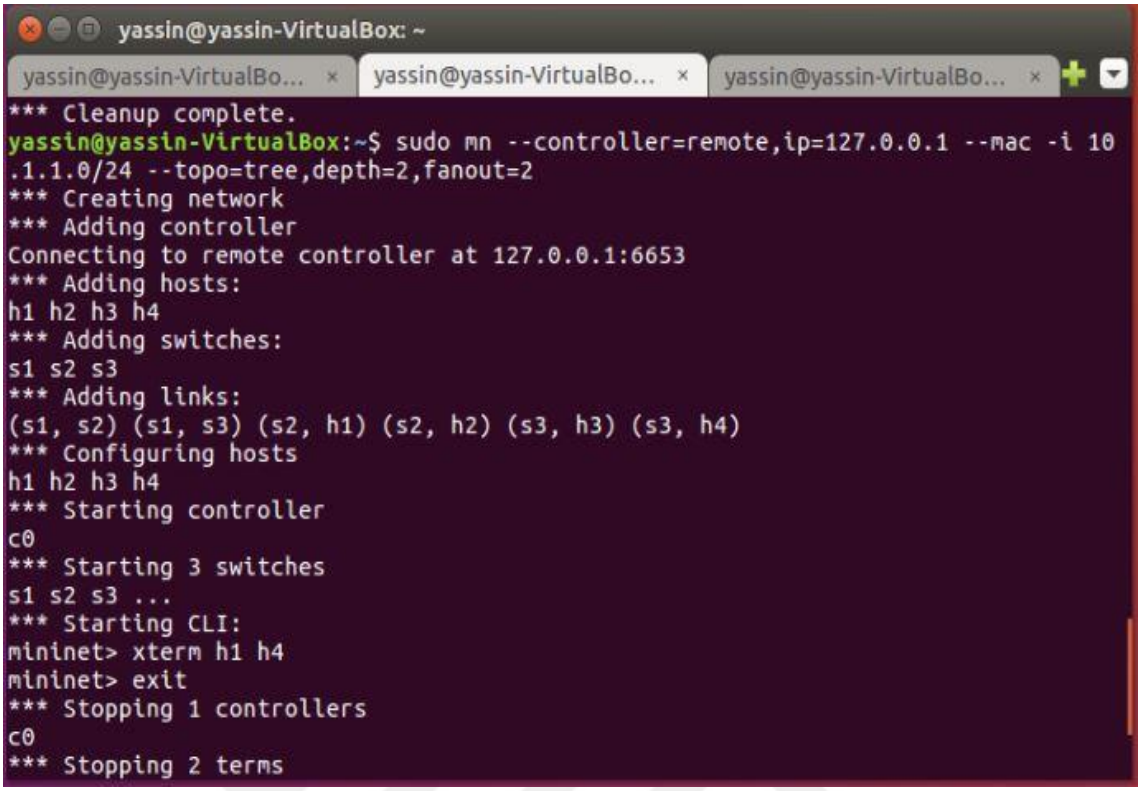

Figure 4.2 *Mininet prompt window.*

4.2.3 Measurement Tools

For this study, precision in measurement is not a compulsory objective. However, it should be accurate enough to show the general behavior of the network flows. Therefore, the following tools are selected to perform the required measurements of the acquired bandwidth and analysis of the traffic: IPerf and Wireshark.

4.2.3.1 IPerf

IPerf is a tool used to measure and generate traffic in IP networks. The main usage for the tool is to determine the maximum throughput between two end hosts in one or both directions. This tool collects statistics based on the number of transmitted packets within the transmission time and interval [60]. This tool is used on a client and a server side. The client side generates the traffic to the server side according to some specifications acquired from the user/admin such as duration of the test, interval, window size, multiple connections or single connection, etc. In this study, this tool is used to generate TCP and UDP traffic. VOIP traffic is basically UDP traffic and IPerf provides the protocol to simulate and generate VOIP traffic, so we took advantage of this feature in IPerf to initiate the VOIP connections. The following commands have been used to generate the required traffic in the performed experiments:

TCP traffic setup (server side): ***# iperf -s -w 100k -i 15***
UDP traffic setup (server side): ***# iperf -s -u -p 6000 -i 15 -t 120***
VOIP traffic setup (server side): # ***iperf -s -u –len 300 –tos 184 -fk -i 15***
TCP traffic generation (client side): # ***iperf -c 10.1.1.4 -w 100k -t 120 &***
UDP traffic generation (client side): # ***iperf -c 10.1.1.4 -u -p 6000 -t 120 &***
VOIP traffic generation (client side): ***# iperf -c 10.1.1.4 -u --len 300 --bandwidth 67000 --dualtest –tradeoff --tos 184 -fk --listenport 5002 -t 170 -P 15 -i 15***

4.2.3.2 Wireshark

Wireshark is one of the world's most commonly used network analyzers. It is used for packets and protocols inspection. This program allows to capture live packets from the network interfaces and get deep inspection about the protocol information, and saves all the captured packets for offline analysis in case needed for further studies [61]. This program is used here to capture the OpenFlow messages exchanged between the controller and the virtual switches. The captured packets can be analyzed under different constraints and can be filtered according to the protocols (e.g., OpenFlow), TCP/UDP ports, and more. The primary goal is to inspect the message called OFPT_FLOW_MOD to investigate that the queue mapping process works in a true manner and it was proved achievable.

4.3 Implementation

In this section, we tested two network topologies to prove our proposed framework efficiency. This section consists of the following two subsections, experiment setup explains the emulated topology and the queue configuration used in the experiments. Experiments and results section illustrate the experiments results through graphs and statistics taken from the measurement tools.

4.3.1 Experiment Setup

The first emulated network consists of seven switches, eight hosts and fourteen links in a tree topology as shown in Figure 4.3. The second emulated network consists of four switches each connected to one host and seven links in a linear topology as shown in Figure 4.4**.** For both experiments we set the link capacity to 5 Mbps. Table 4.2 shows the

generated flows corresponding to queues numbers, traffic types and queue maximum rates.

Table 4.2. *Generated traffic and the corresponding queues.*

Queue Number	Traffic Type	Queue Maximum Rate
q_0	VOIP	1.5 Mbps
q_1	TCP	1 Mbps
q_2	UDP	0.5 Mbps

q_0 is used to forward the VOIP traffic, q_1 is used for the TCP flows and q_2 is used for UDP flows to be forwarded. The maximum rates of the of these queues are equal to the value specified by the user/admin. Here we set the maximum rate of the queues to 1.5 Mbps for VOIP, 1 Mbps for TCP, and 0.5 Mbps for UDP. However, the maximum link capacity is 5 Mbps. In Mininet, we can create the network topology directly using the Linux command line inside the Mininet prompt window as the following which creates tree topology:

$ sudo mn --controller=remote, ip=127.0.0.1--mac -i 10.1.1.0/24 --topo=tree, depth=3, fanout=2

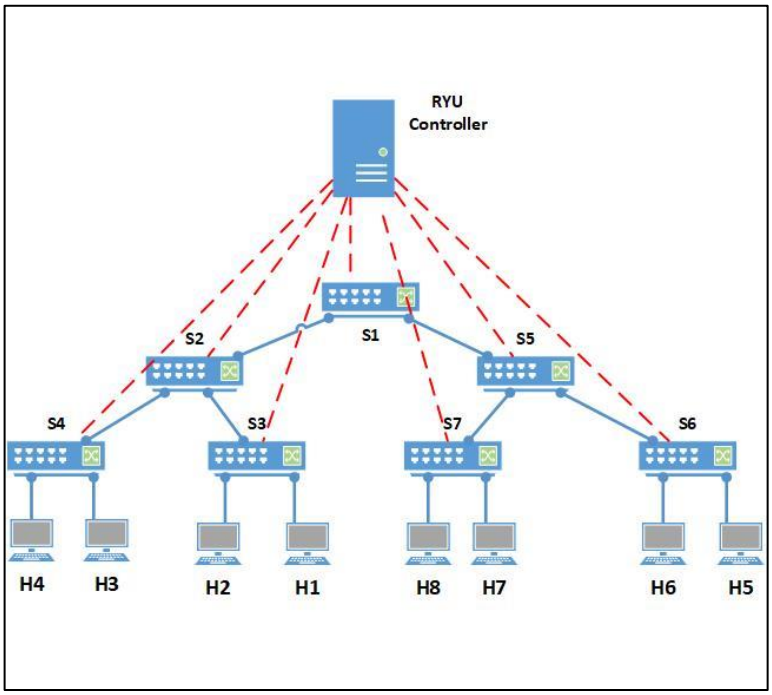

Figure 4.3 *The first emulated network (tree topology)*

For linear topology the following command is used:

$ sudo mn --controller=remote, ip=127.0.0.1--mac -i 10.1.1.0/24 --topo=linear,4

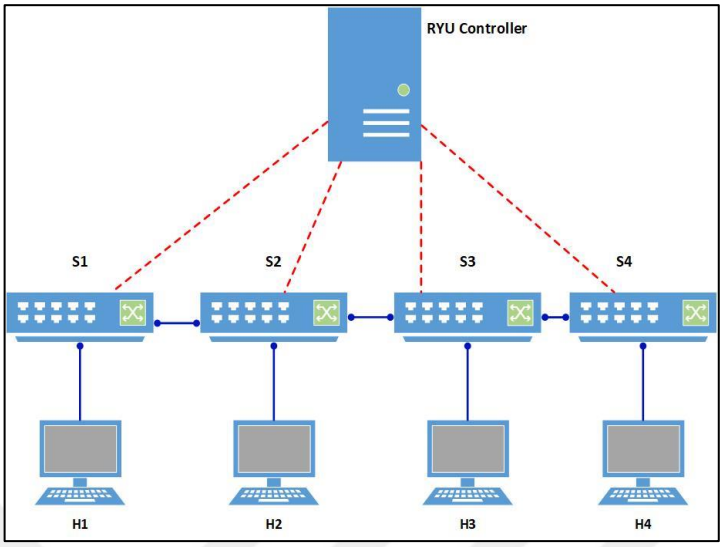

Figure 4.4 *The second emulated network (linear topology)*

The user/admin can specify the required bandwidth for the several types of traffic using the configuration file passed to the controller. This file is used to create the configuration of the queues so the controller will rate limit the flows at the switches accordingly. Our aim here is to utilize the link capacity to share among flows that use the same link. Iperf and Wireshark analyzer are selected as the measurement tools to measure and analyze the traffic under different QoS configurations. The experiments are mainly focused on the followings: First, test the ability of the application layer of RYU to configure queues on the switches and direct the traffic to corresponding queues according to queue mapping process as seen in Figure 4.1. Second, show the flexibility to install QoS entries in the SDN concept using RYU. Third, limit the bandwidth according to the admin/user specification for the desired application usage on the fly. Lastly, show the ability to design a monitoring system embedded in the controller to observe the behavior of the queues.

4.3.2 Experiments and Results

First, we will show the queue mapping process to direct traffic to its specific queues and present the results using the Wireshark analyzer demonstrated in the queue mapping section 4.3.2.1. As mentioned before two experiments are carried out on two network topologies. The first topology (tree topology) is tested and traffic is injected for the three types of traffic; graphs are obtained and the bandwidth results are recorded. The

second topology (linear topology) is tested in the same scenario as the first topology; graphs are obtained and bandwidth results are recorded. The result section 4.3.2.2 will explain test results.

4.3.2.1 Queue mapping

After the generation of traffic, the Wireshark program is used to capture the packets between the controller and the switches. Our aim is to check the specific message called "OFP_FLOW_MOD" which is responsible for the flow entries update and modification. Figure 4.5, 4.6 and 4.7 show the queue mapping process to direct the flows to their specified queues TCP, UDP and VOIP respectively. From the figures, we can notice each type of the chosen traffic was directed to the right destined queue, so it was proved achievable. In other words, TCP flows are directed to q_1, VOIP flows are directed to q_0 and UDP flows are directed to q_2.

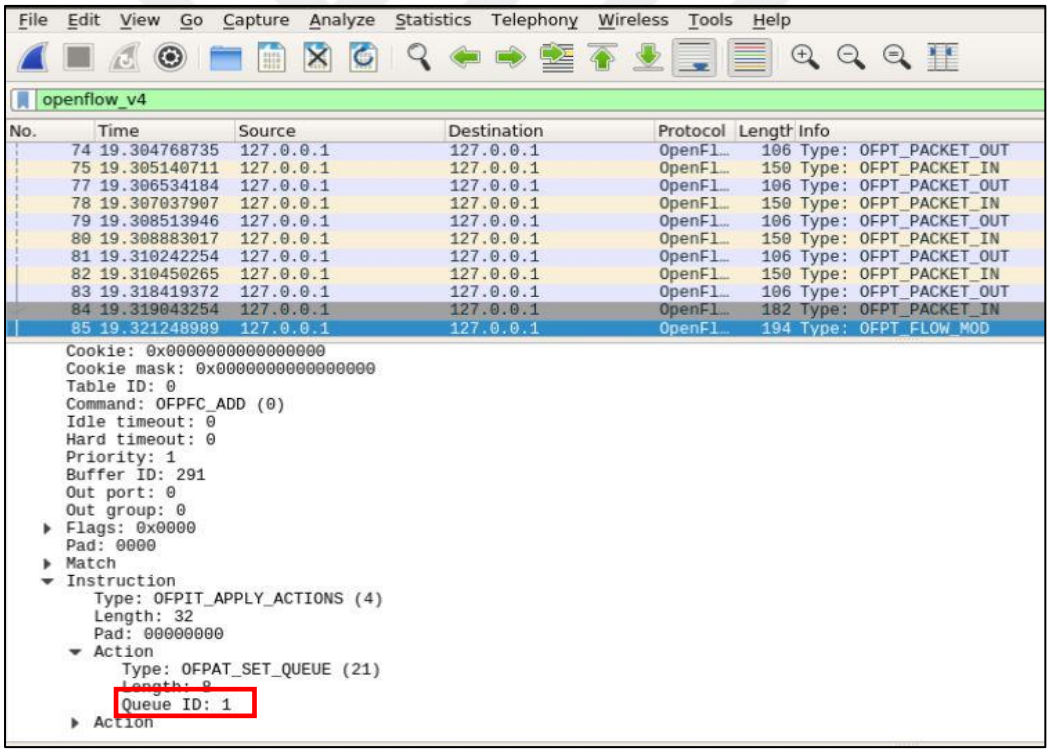

Figure 4.5. *TCP queue mapping.*

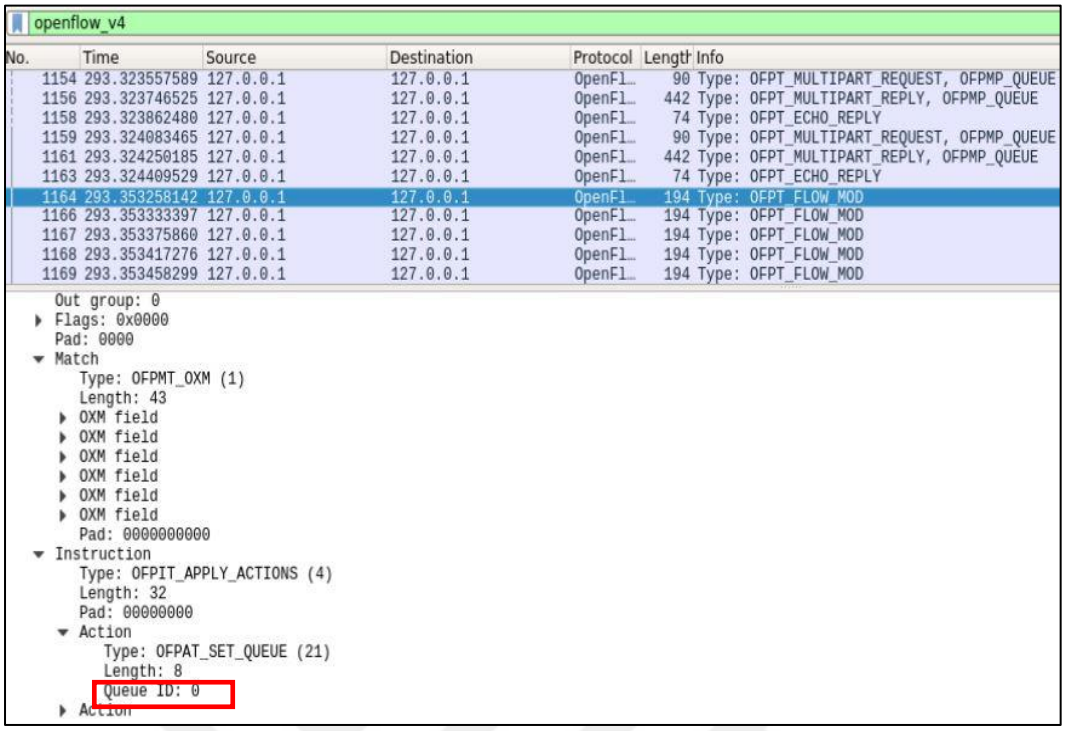

Figure 4.6. *VOIP queue mapping.*

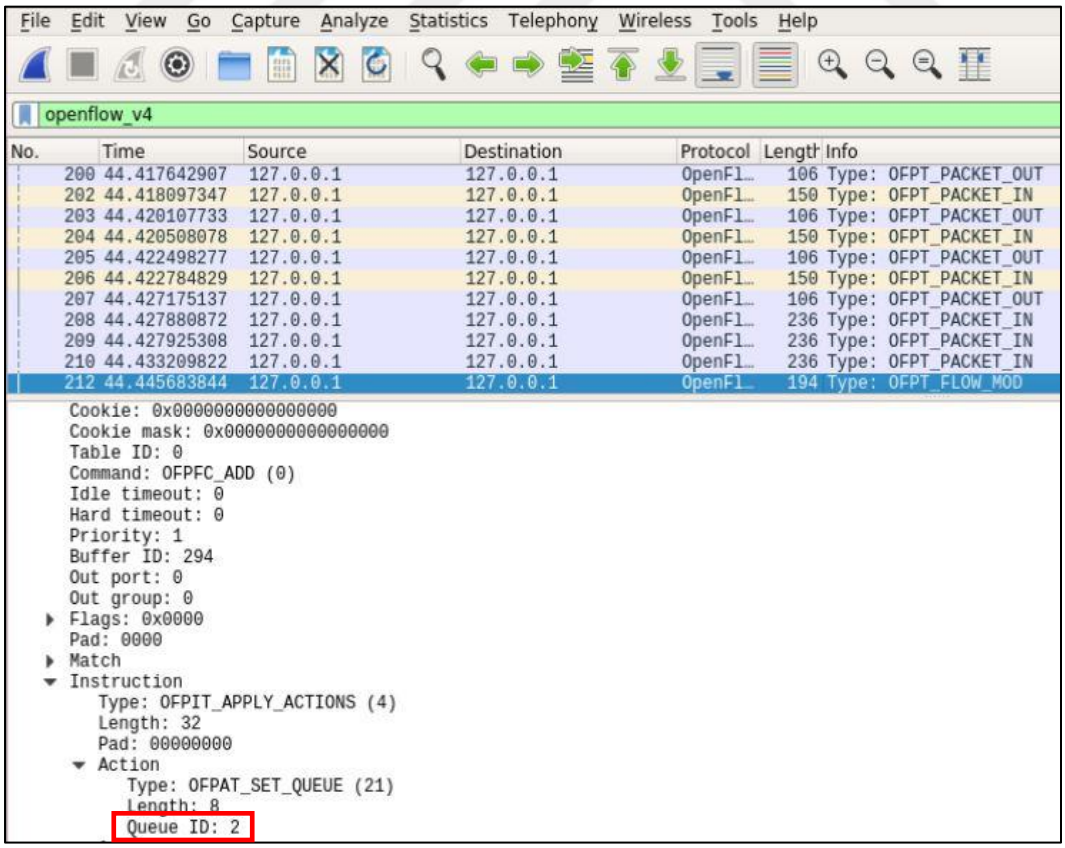

Figure 4.7. *UDP queue mapping.*

4.3.2.2 Results

Two experiments have been conducted**. Case 1:** in this experiment, three types of traffic flows are generated from two hosts [H1, H8] in parallel connected to switches [S3, S7] in a tree topology as shown in Figure 4.3 before. Traffic is directed to its corresponding queues. Iperf testing tool is used to generate traffic and measure the bandwidth allocated for TCP, UDP and VOIP traffic. For VOIP traffic, 15 and 25 calls were initiated. Bandwidth are observed and results are taken from the server side [H8]. The duration of the test is approximately 120 seconds the interval taken to show the results every 15 seconds. Figure 4.8, 4.9 and 4.10 represent TCP, UDP and VOIP bandwidth allocations respectively.

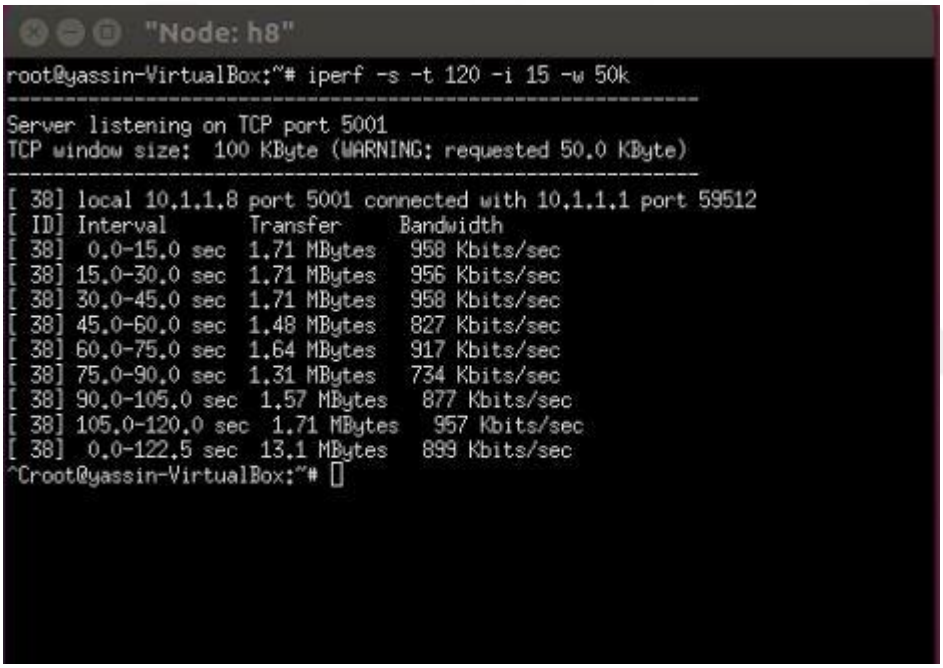

Figure 4.8. *Case 1 TCP bandwidth allocation.*

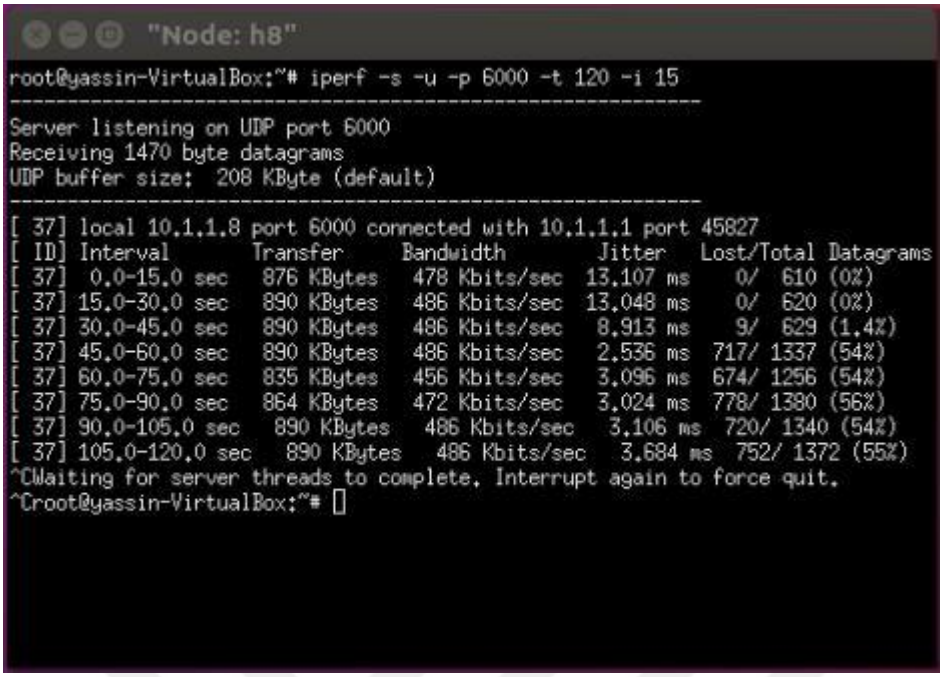

Figure 4.9. *Case 1 UDP bandwidth allocation.*

Figure 4.8 and 4.9, show the results illustrating how the bandwidth is complying to the specified bandwidth set by the user/admin respectively.

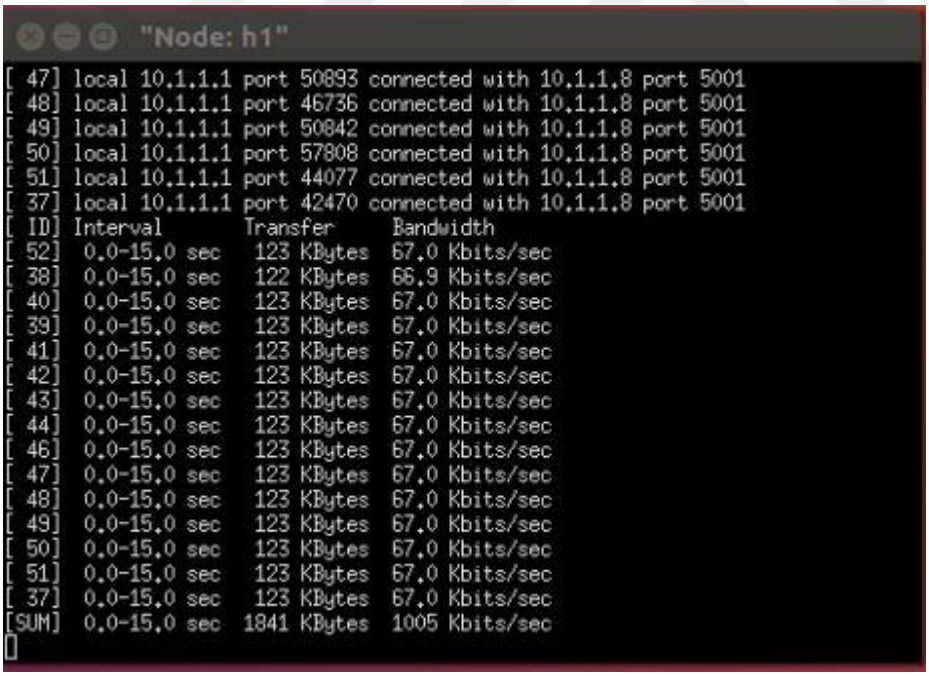

Figure 4.10. *Case 1 VOIP bandwidth allocation. for 15 calls (1005 Kbps).*

For the first VOIP test, we established 15 VOIP calls, for each call 67Kbps of bandwidth has been reserved, so in total 1005Kbps bandwidth is allocated as shown in Figure 4.10. Queue statistics have been fetched as shown in Figure 4.11.

```
yassin@yassin-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 queue-stats s3
[sudo] password for yassin:
OFPST_QUEUE reply (OF1.3) (xid=0x2): 9 queues
  port 1 queue 0: bytes=1217, pkts=13, errors=0, duration=129470322.099s
  port 1 queue 1: bytes=370, pkts=1, errors=0, duration=129470322.099s
  port 1 queue 2: bytes=0, pkts=0, errors=0, duration=129470322.099s
  port 2 queue 0: bytes=1068, pkts=11, errors=0, duration=129470322.099s
  port 2 queue 1: bytes=0, pkts=0, errors=0, duration=129470322.099s
  port 2 queue 2: bytes=0, pkts=0, errors=0, duration=129470322.099s
  port 3 queue 0: bytes=3155718, pkts=9236, errors=0, duration=129470322.099s
  port 3 queue 1: bytes=0, pkts=0, errors=0, duration=129470322.099s
  port 3 queue 2: bytes=0, pkts=0, errors=0, duration=129470322.099s
```

Figure 4.11. *Case 1 queue statistics for VOIP 15 calls test show no dropping rate because the allocated bandwidth is 1005Kbps which is not exceeded.*

From Figure 4.11, we can observe that VOIP queue shows no dropping rate of packets since the VOIP queue limit is 1500 Kbps. Another test has been triggered but this time we increased the number of VOIP calls to 25 yielding a total of 1676 Kbps so that the allocated bandwidth is exceeded the queue's specified rate 1500kbps.

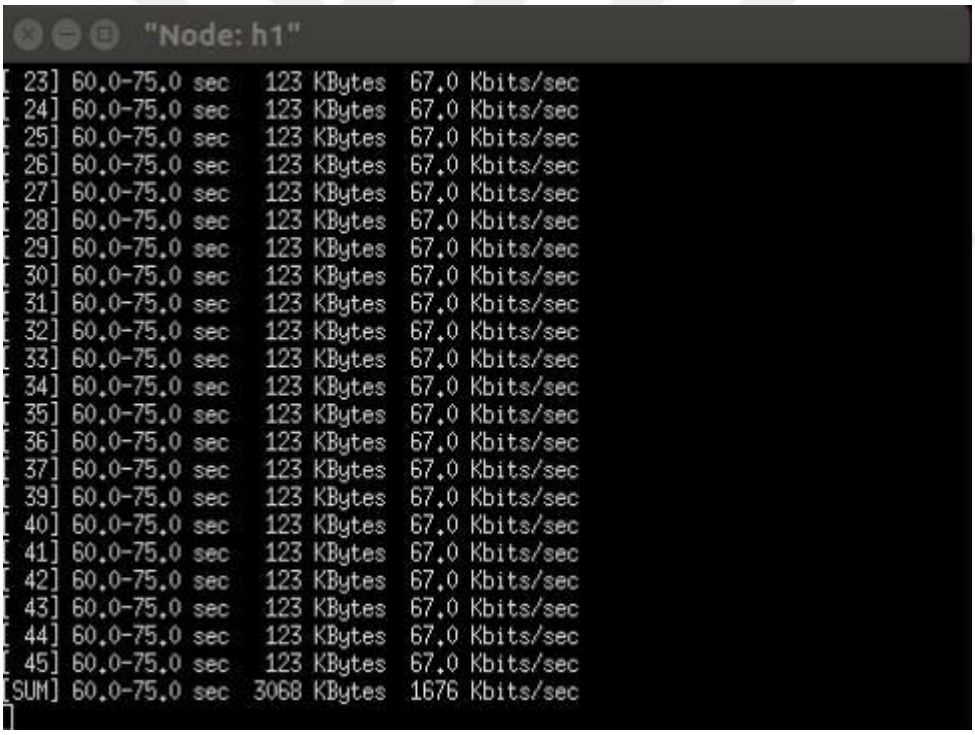

Figure 4.12. *Case 1 VOIP bandwidth allocation for 25 calls (1676 Kbps).*

According to Figure 4.12, the recorded bandwidth shows 1676 Kbps, so here the bandwidth is exceeded the allowed rate specified by the admin/user which will cause the exceeded packets to be dropped. Figure 4.13 shows dropping error rate in the queue in the field errors which represent a number of packets dropped by the queue due to the exceeded bandwidth limit.

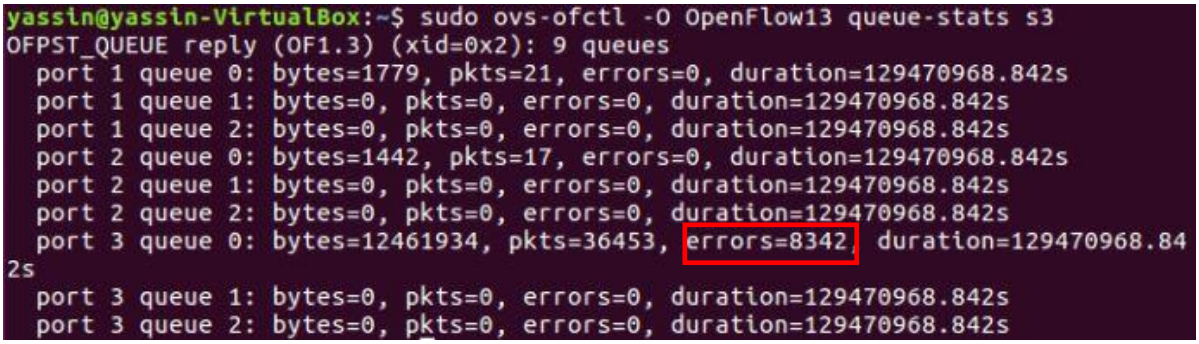

Figure 4.13. *Case 1 queue statistics for VOIP 25 calls test shows dropping rate because allocated bandwidth is 1676Kbps which is exceeded.*

In short, our framework satisfies and works our allocated bandwidth properly. As for the TCP and UDP tests, Figure 4.14, 4.15 and 4.16, show the queues utilization in case of individual traffic injection of TCP, UDP and VOIP traffic respectively. On the other hand, Figure 4.17 shows the queue utilization for the parallel traffic injection of TCP, UDP and VOIP in parallel.

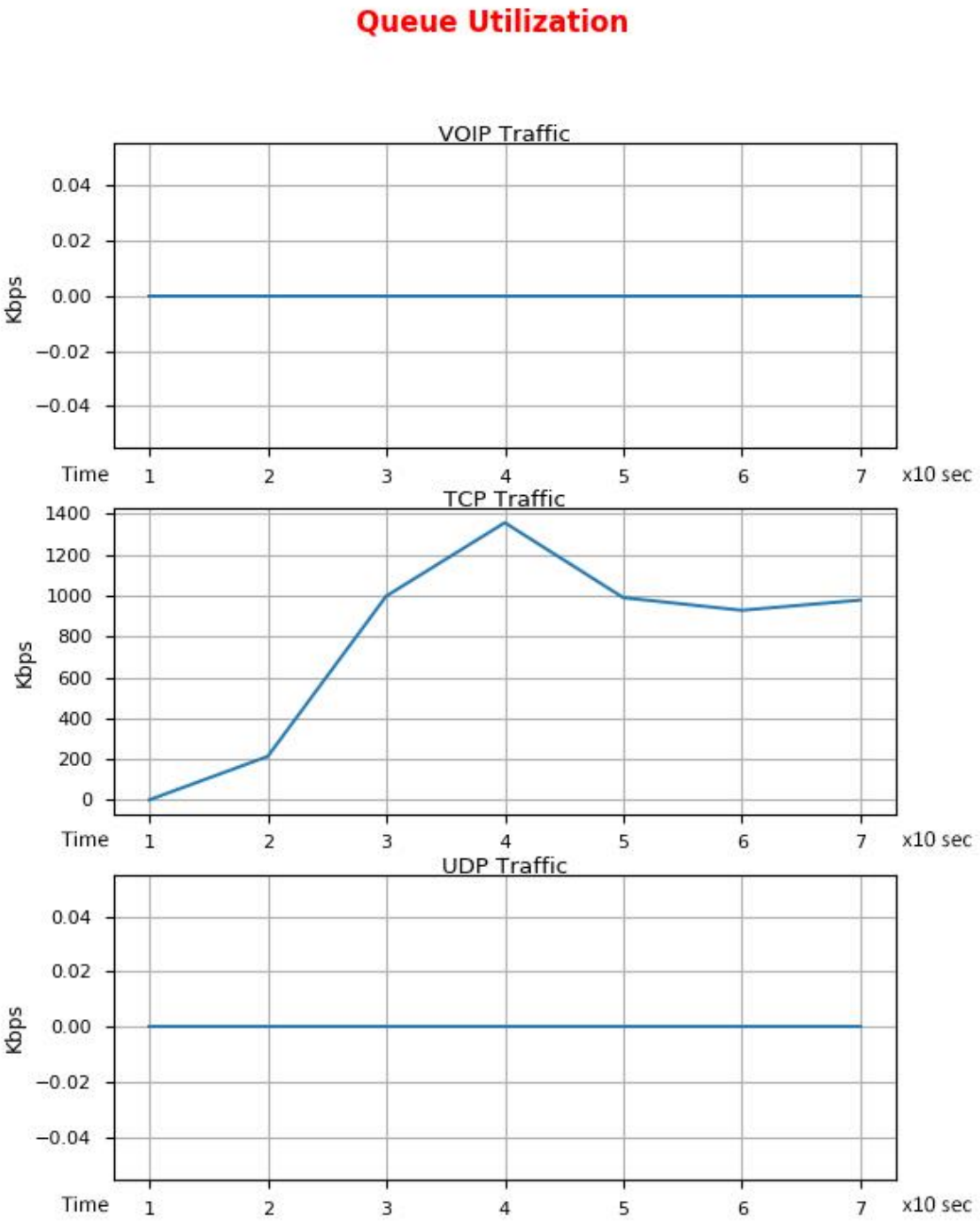

Figure 4.14. *Case 1 TCP test queue utilization.*

In Figure 4.14 we can see the queue utilization trying to comply with specified rate for TCP queue 1000Kbps. The graph represents the result of the tested tree topology.

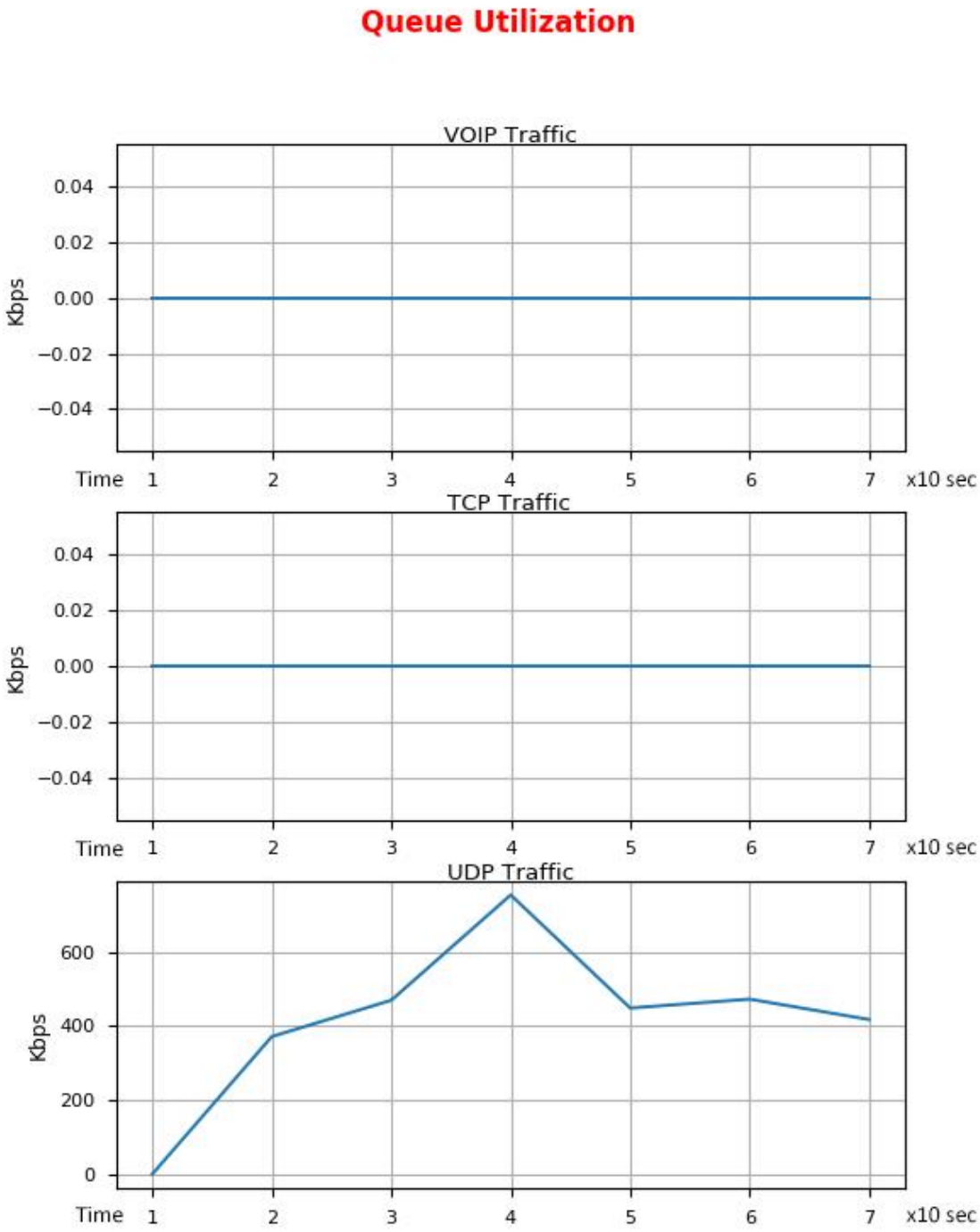

Figure 4.15. *Case 1 UDP test queue utilization.*

As shown in Figure 4.15 we can see the queue utilization trying to comply with specified rate for UDP queue 500Kbps.

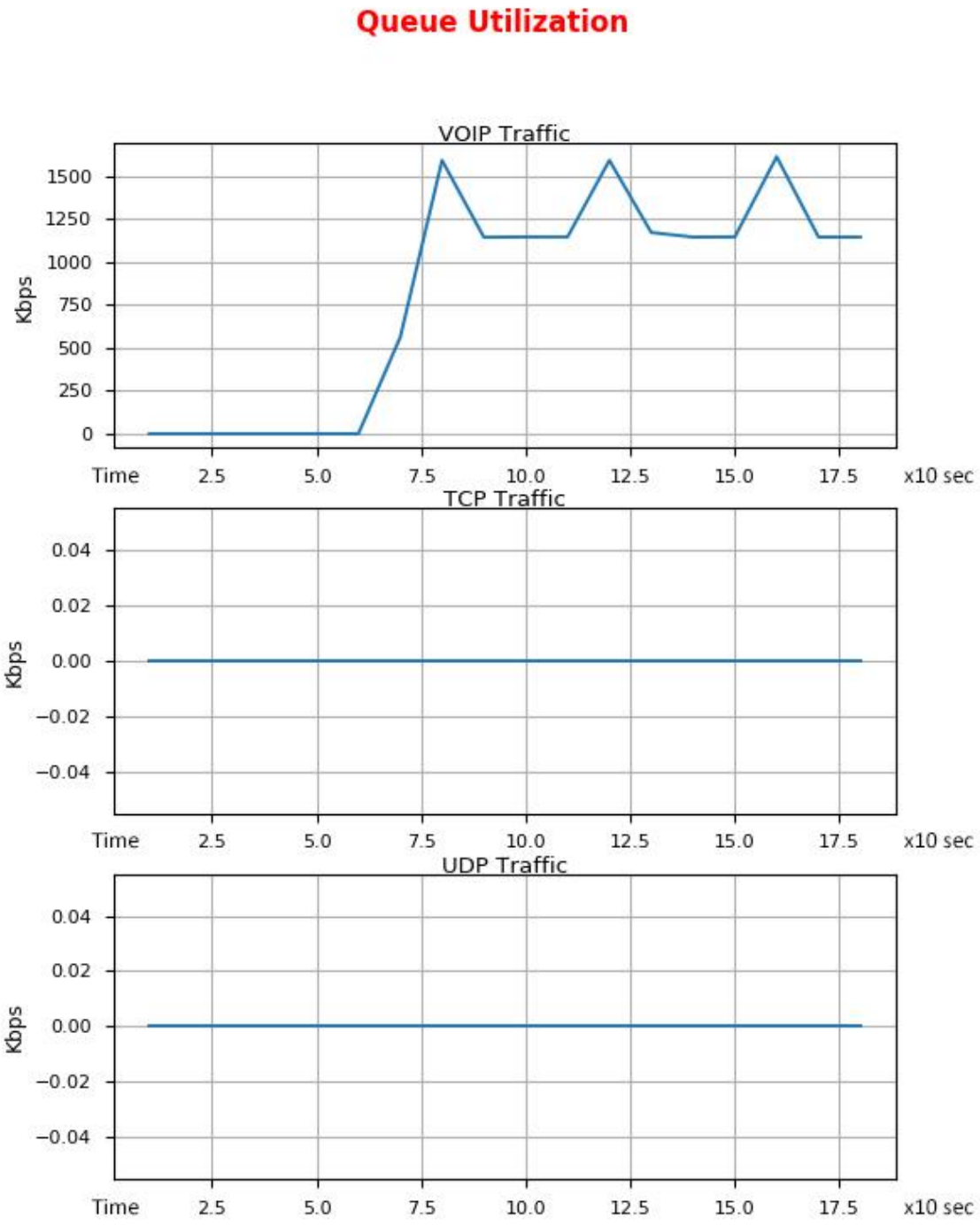

Figure 4.16. *Case 1 VOIP queue utilization.*

As shown in Figure 4.16 the queue utilization trying to comply with the recorded VOIP bandwidth of 1005Kbps.

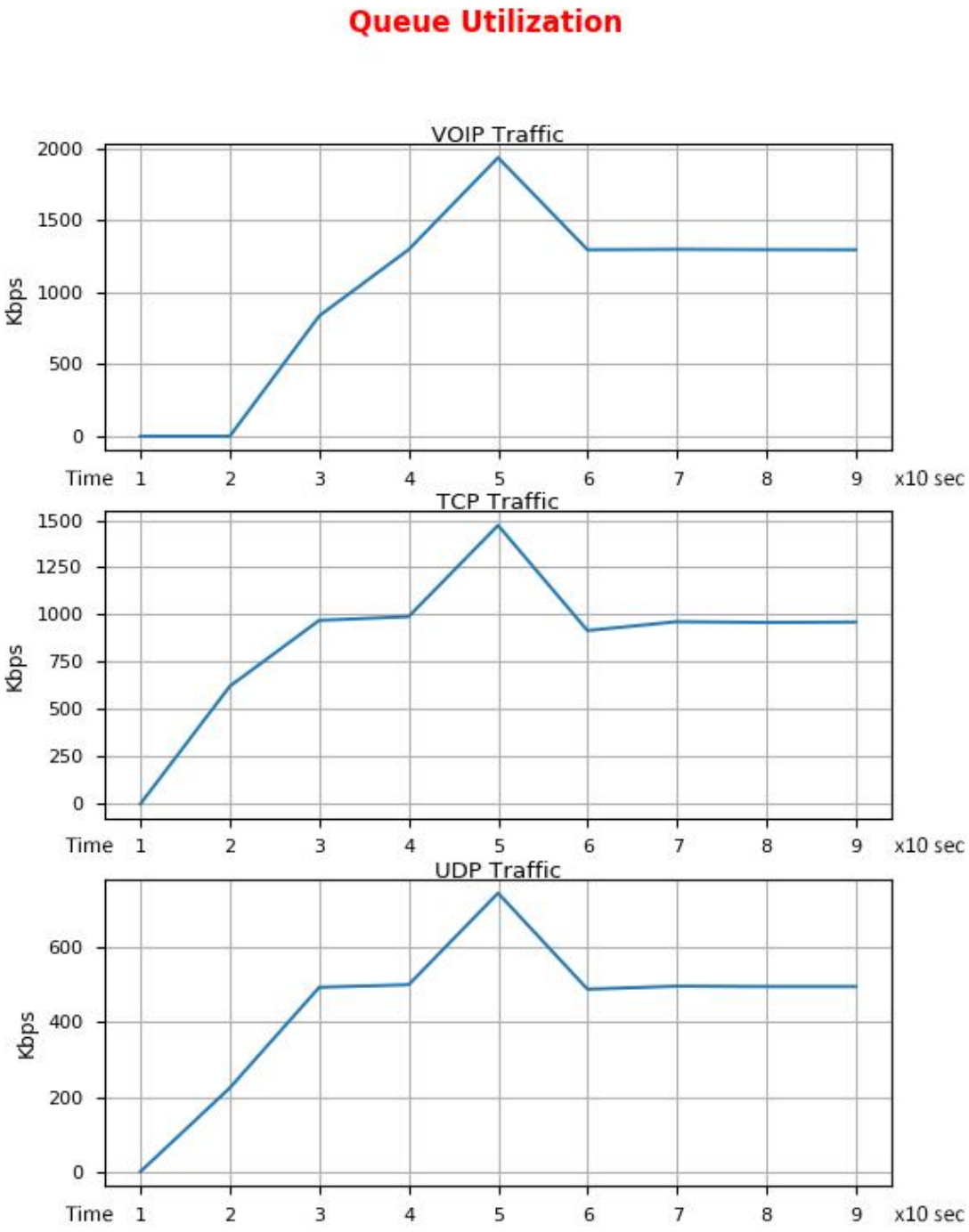

Figure 4.17. *Case 1 VOIP, TCP, and UDP parallel test queue utilization.*

In Figure 4.17 the graph shows the queue utilization of the three-traffic injected in parallel, trying to comply with the specified bandwidth rates set by the admin for TCP 1000 Kbps, UDP 500 Kbps, and VOIP 1500 Kbps.

Case 2: in this experiment a linear topology is created and tested as shown in Figure 4.4**.** Traffic are generated as the first experiment in individual and parallel traffic injection. Figure 4.18, 4.19, 4.20 and 4.21 illustrate the results obtained for the bandwidth allocated during the test (TCP, UDP, VOIP 15 calls and VOIP 25 calls). Figures 4.22 and 4.23 show the queue statistics fetched from the switches showing the dropping rate status for the 15 and 25 VOIP calls. Figure 4.24, 4.25, 4.26 and 4.27 show the queues utilization in case of parallel and individual traffic injection for the second tested topology (linear topology). We considered H1 is the client side, H8 is the server side for the first emulated topology (tree). H1 is the client side, H4 is the server side for the second emulated topology.

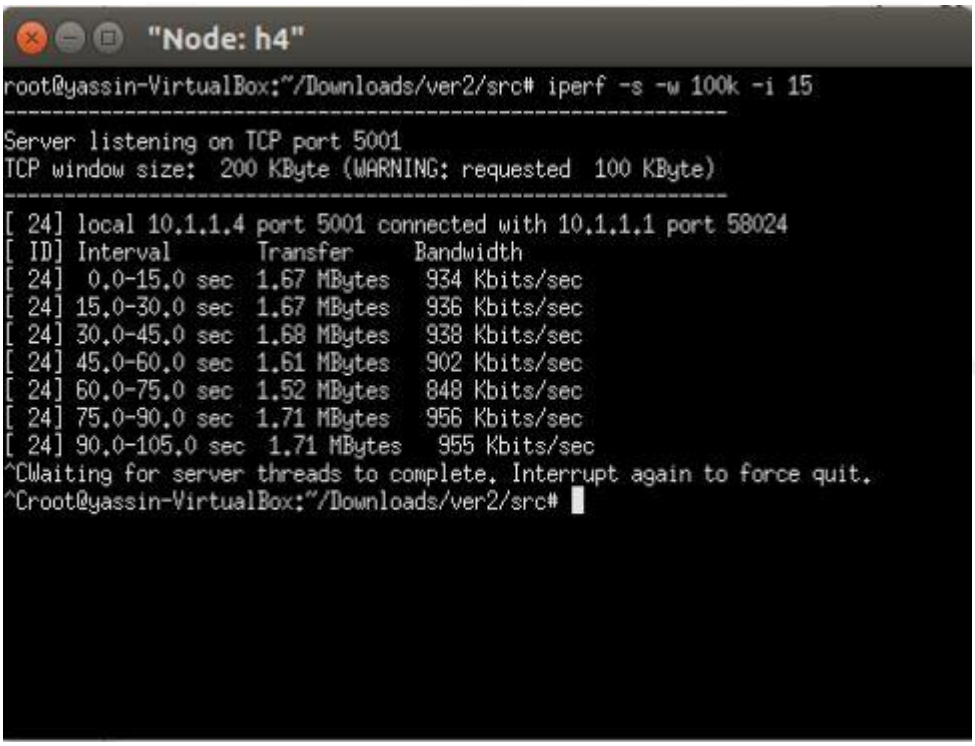

Figure 4.18. *Case 2 TCP bandwidth allocation.*

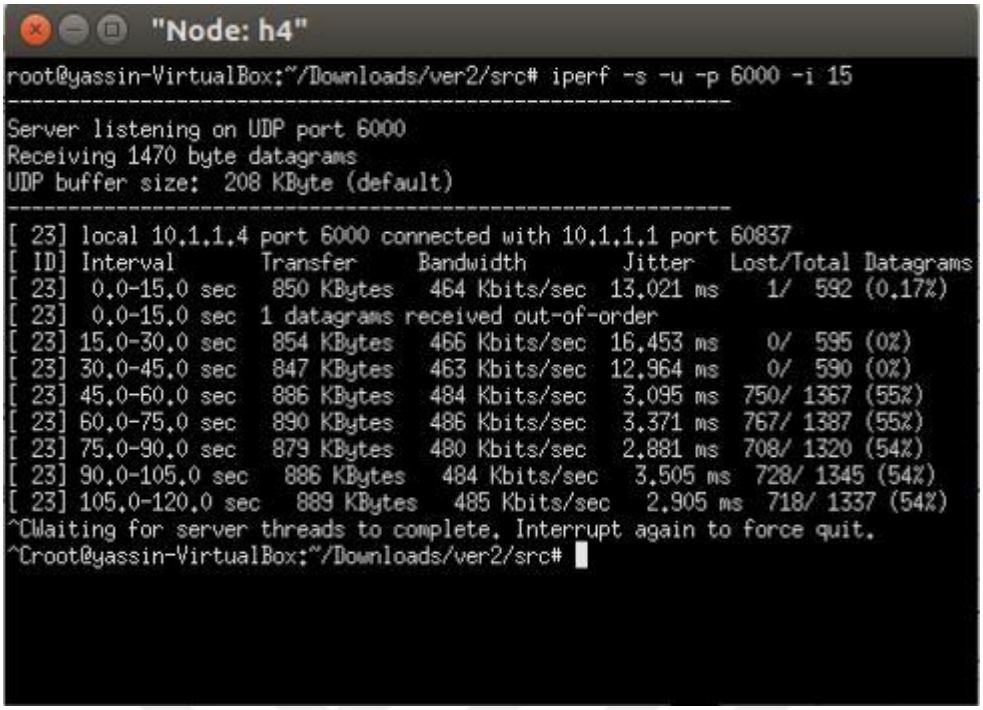

Figure 4.19. *Case 2 UDP bandwidth allocation.*

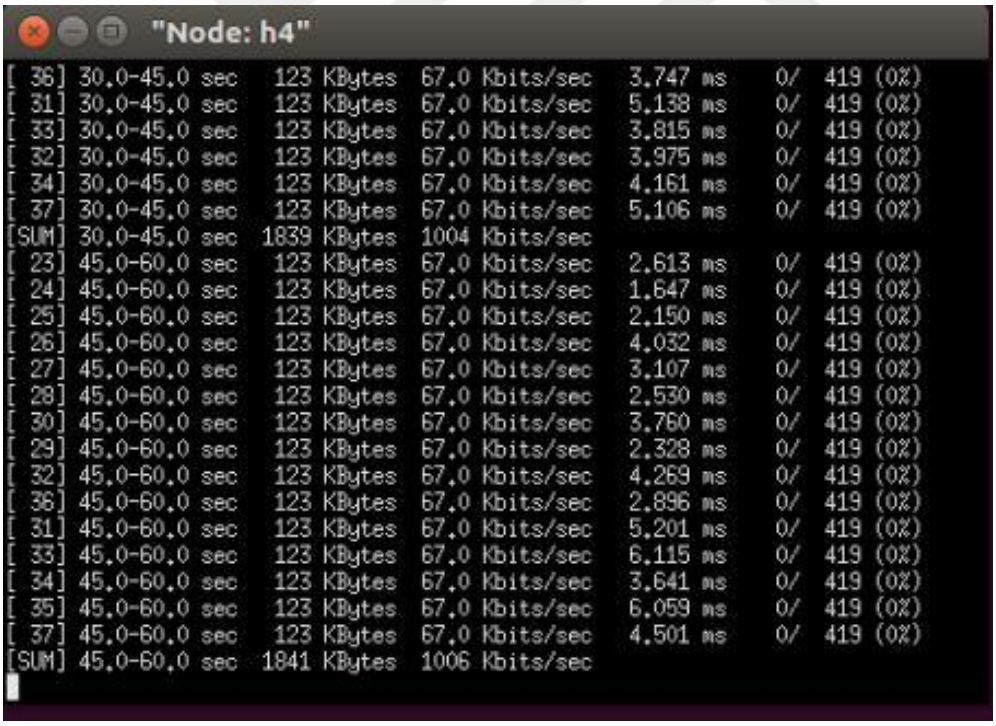

Figure 4.20. *Case 2 VOIP bandwidth allocation for 15calls.*

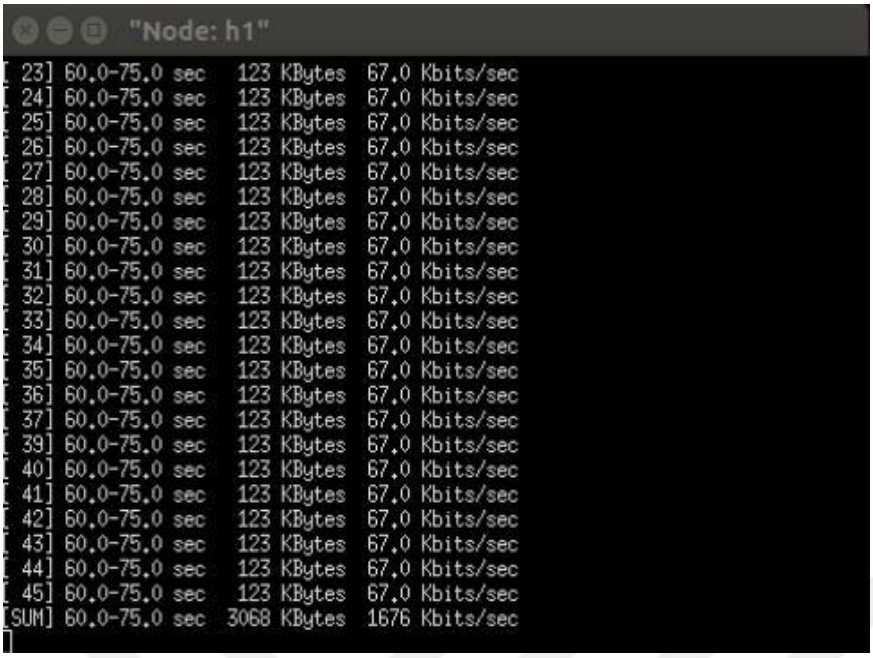

Figure 4.21. *Case 2 VOIP bandwidth allocation for 25 calls.*

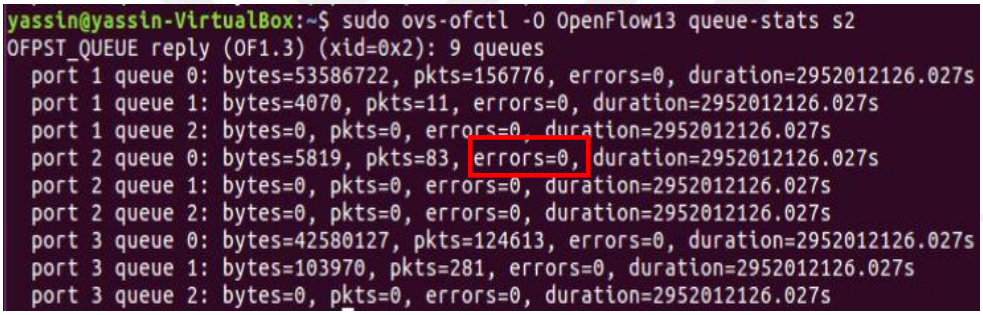

Figure 4.22. *Case 2 queue statistics for VOIP 15 calls test shows no dropping rate because the allocated bandwidth is 1006Kbps which is not exceeded.*

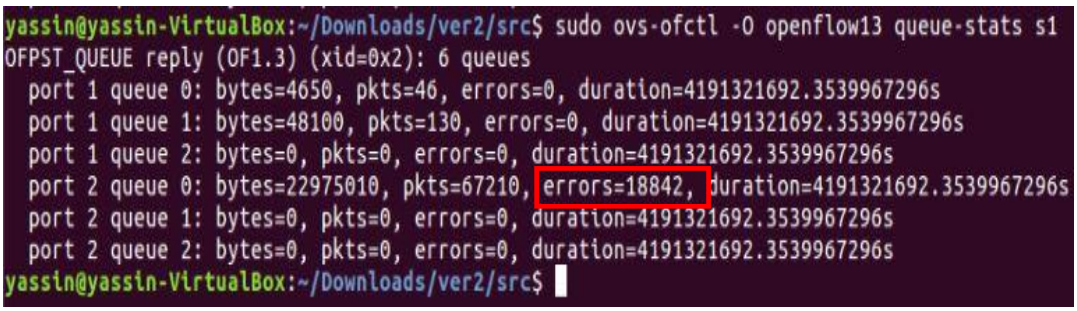

Figure 4.23. *Case 2 queue statistics for VOIP 25 calls test shows dropping rate because allocated bandwidth is 1676Kbps which is exceeded.*

In Figure 4.22 we can see there is no dropping rate since the queues haven't exceeded the specified rate 1500 Kbps for VOIP traffic, the error field represent the number of dropped packets due to the queue overrun. On the contrary, in Figure 4.23 we can see the dropping of packets since the queue exceeded the specified rate by the admin which is 1500Kbps for VOIP traffic.

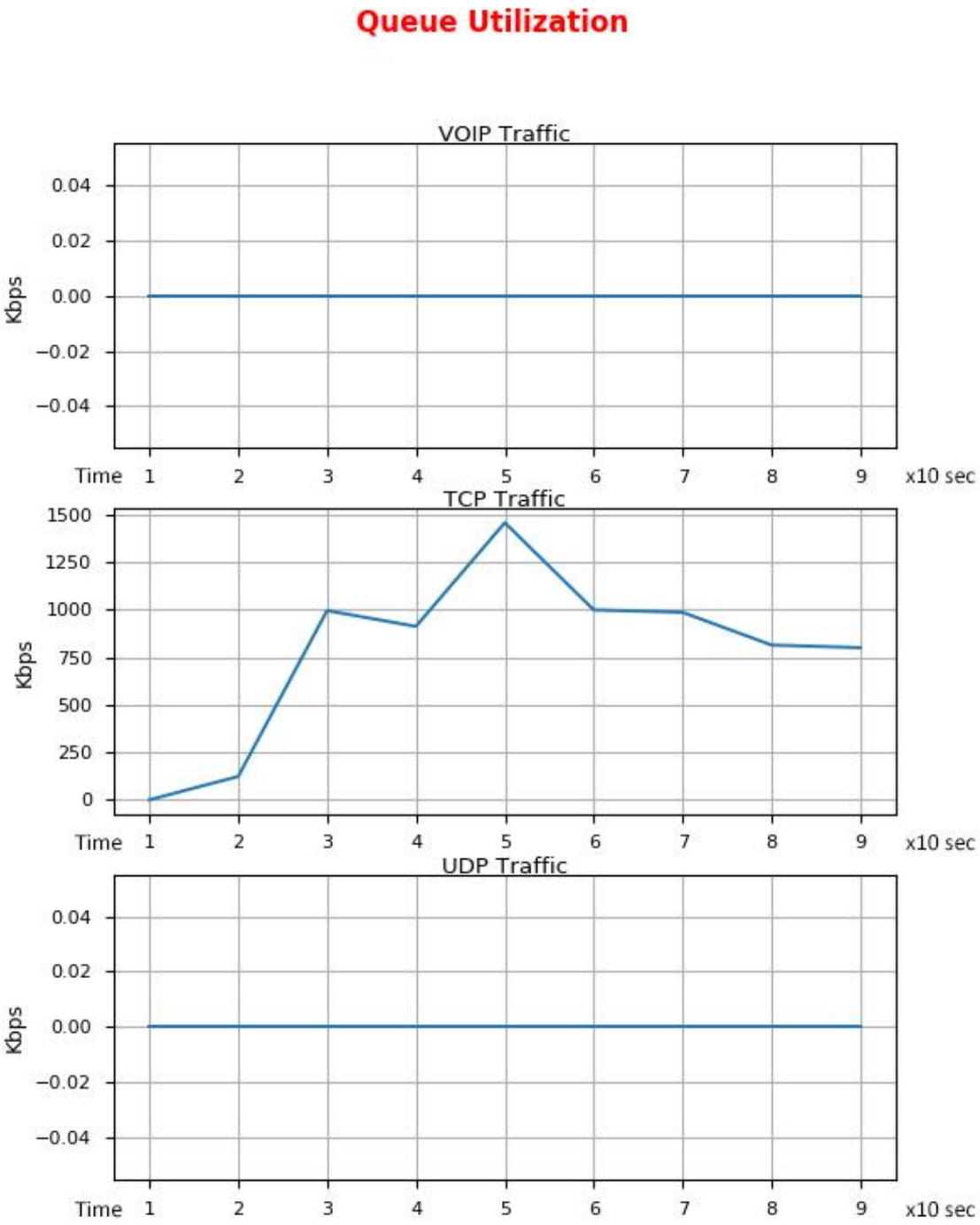

Figure 4.24. *Case 2 TCP test queue utilization.*

As shown in Figure 4.24 the graph shows the queue utilization trying to comply with the bandwidth specified by the admin 1000 Kbps. The graph represents the results of the tested linear topology.

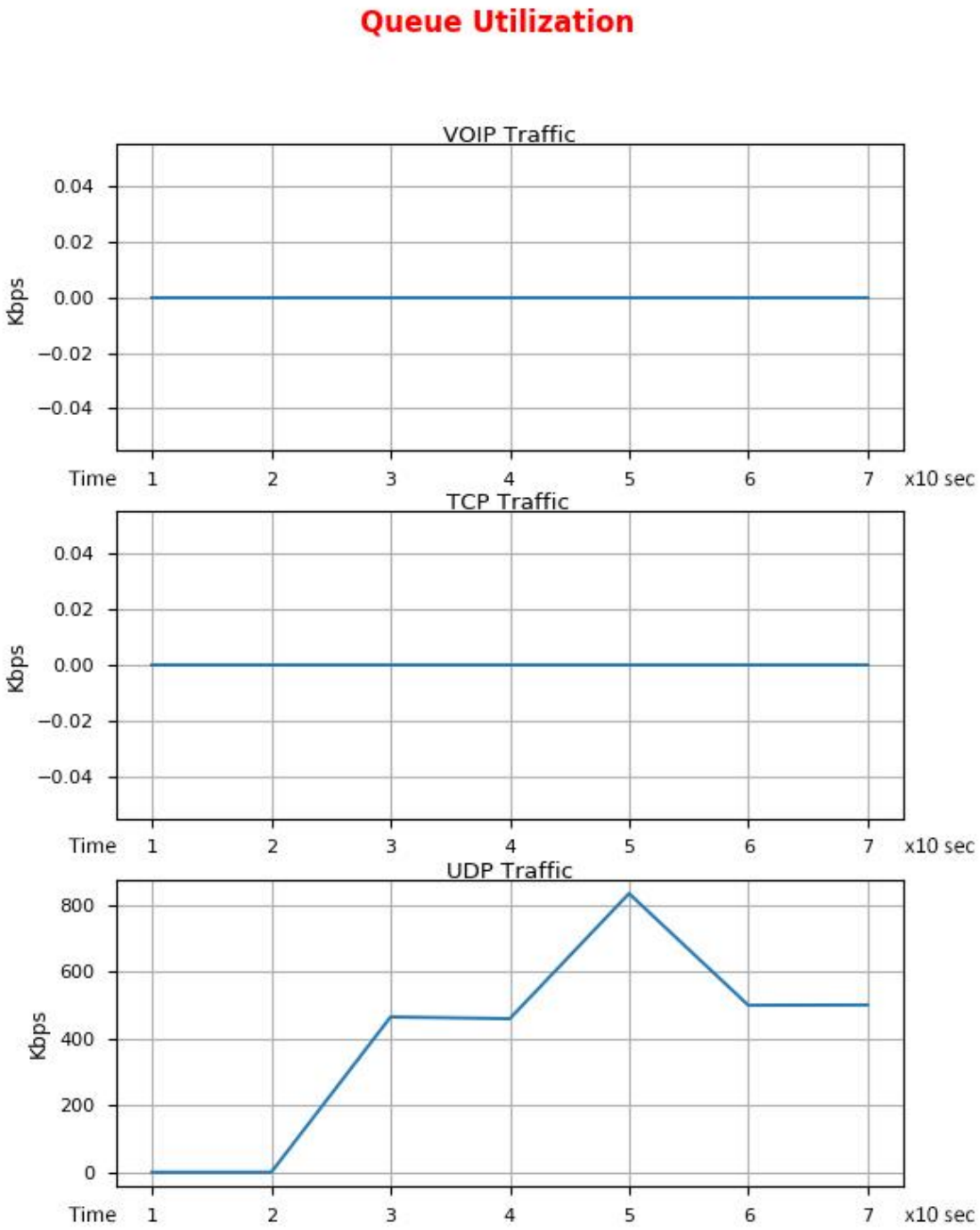

Figure 4.25. *Case 2 UDP test queue utilization.*

As shown in Figure 4.25 the graph shows the queue utilization trying to comply with the bandwidth specified by the admin 500 Kbps. The graph represents the results of the tested linear topology.

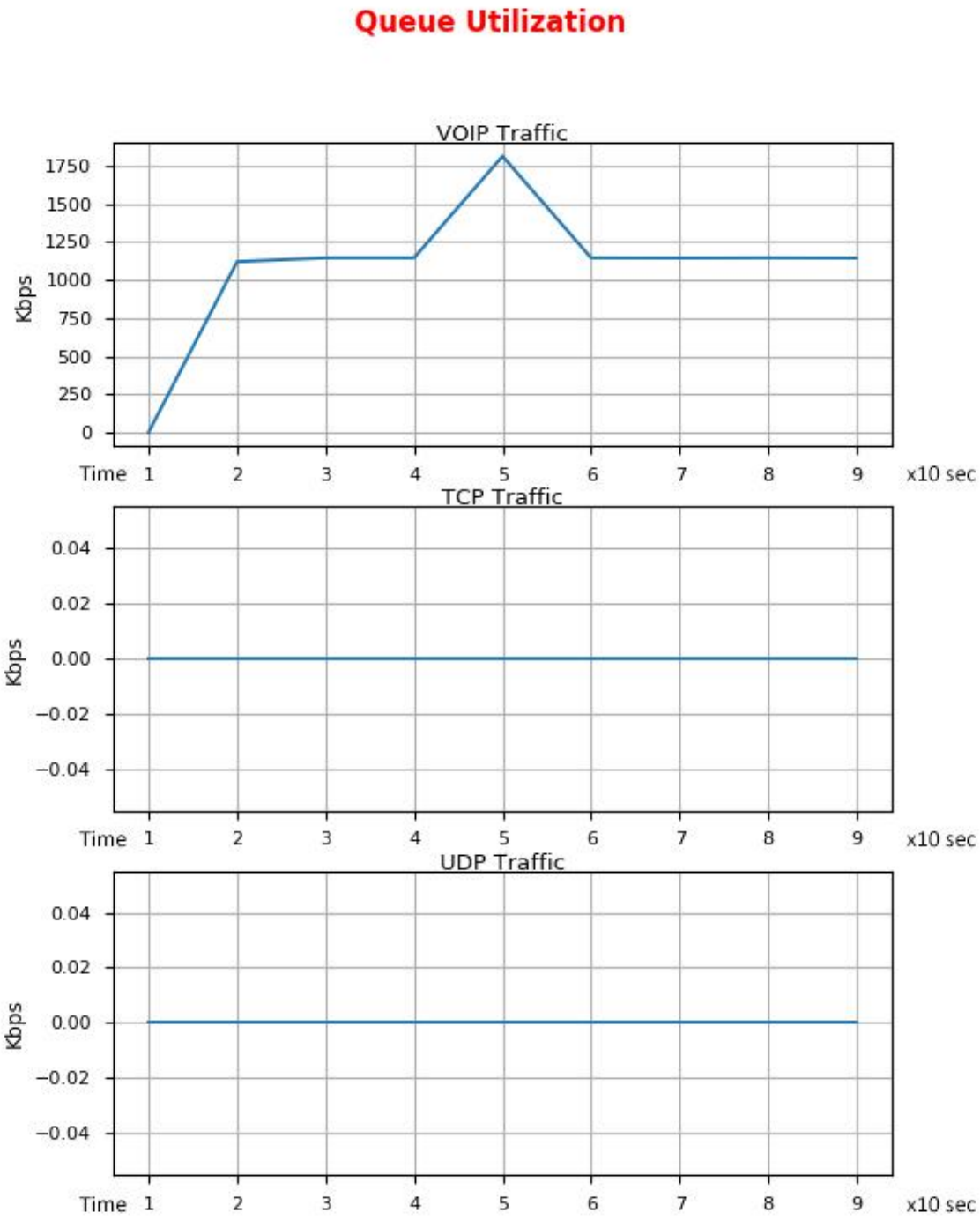

Figure 4.26. *Case 2 VOIP test queue utilization.*

As shown in Figure 4.26 the graph shows the queue utilization trying to comply with the bandwidth specified by the admin 1005 Kbps. The graph represents the results of the tested linear topology.

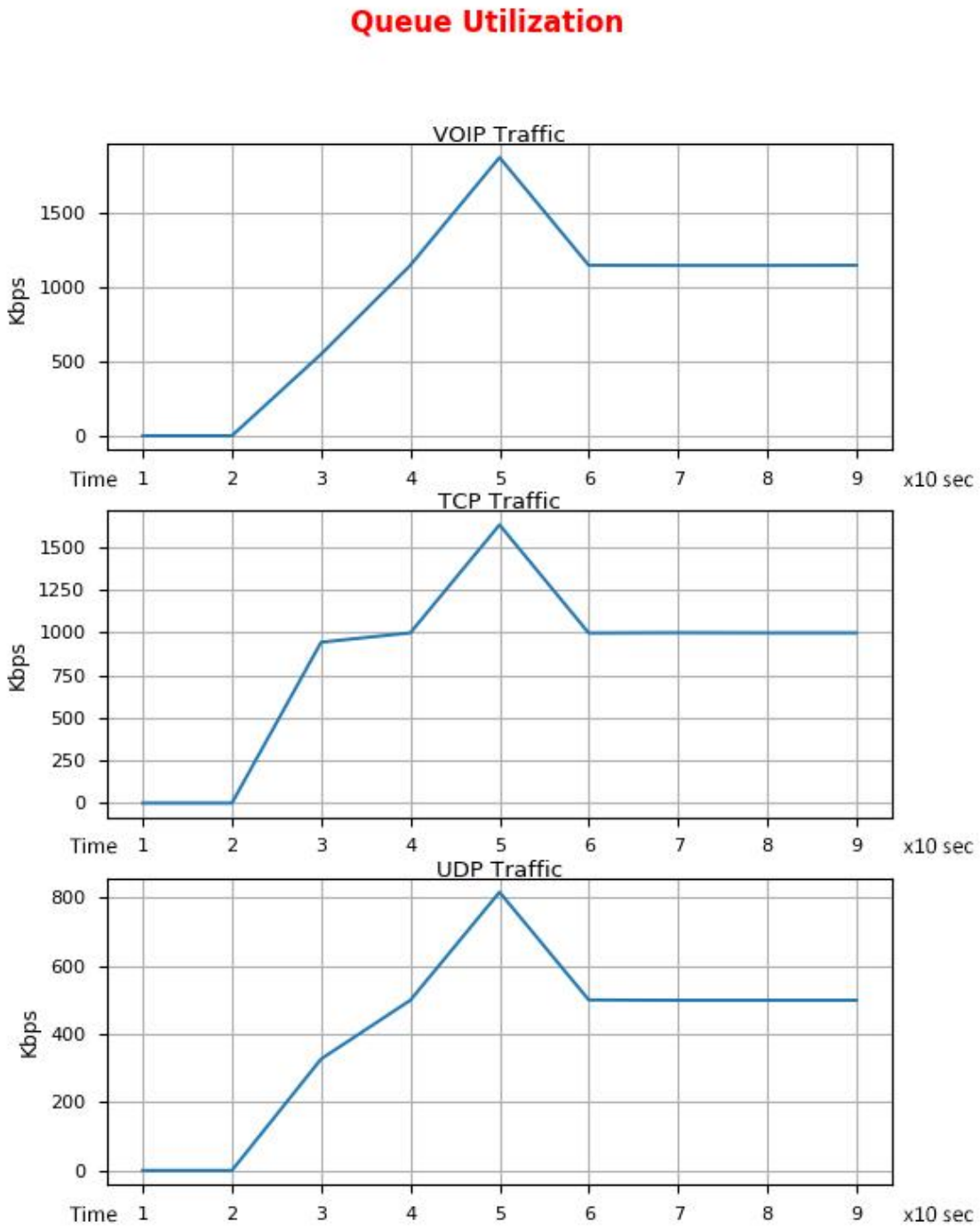

Figure 4.27. *Case 2 VOIP, TCP, and UDP parallel test queue utilization.*

In Figure 4.27 the graph shows the queue utilization of the three-traffic injected in parallel, trying to comply with the specified bandwidth rates set by the admin for TCP 1000 Kbps, UDP 500 Kbps, and VOIP 1005 Kbps.

4.3.2.3 Discussion

The two experiments are triggered approximately between 70- and 170-seconds duration. We observed similar behavior for both topologies showing that our framework can be applied to multiple topologies. The observed queue utilization approximately complies with the bandwidth, even though there are some peaks happening due to the TC control technique algorithms used in Linux kernel and Iperf tool used, so some bursts could happen. However, this phenomenon is normal, but the queues are regulating the bandwidth to maintain it around the specified bandwidth required. As we mentioned before the traffic generated is trying to comply to the specified bandwidth with TCP 1 Mbps, UDP 0.5 Mbps and VOIP 1.5 Mbps.

5. CONCLUSION

This study was aimed to install QoS criteria to an SDN network using RYU controller. Hence, an SDN-app for configuring QoS was designed and implemented using rate limiting queues installed on Open vSwitches (OVS), configured using the OVSDB management protocol. Our architecture relies on SDN technologies and allows us to discover and manage the current network. Due to the programmability of the network, efficiency is increased and the network become flexible for user/admin necessities. Moreover, the performance of the proposed framework for user defined bandwidth allocation is tested in terms of achievable bandwidth which is considered as one of the primary QoS parameters. Some of the advantages of our proposed framework: it can be applied to small to mid-size topologies, not only restricted to a certain topology. In addition, three types of traffic are injected, distinguished and tested simultaneously and individually; results are presented using graphs from a monitoring system designed in the controller.

The test results illustrate the capabilities, and give an indication about the performance of our QoS control module. The proposed system is able to provide the user defined bandwidth required for bandwidth allocations and it is proved achievable. In this study, incoming throughput is considered as the main performance criterion. The impact of bandwidth allocation approach on other performance parameters (e.g., delay, loss, jitter) can be considered as future work.

REFERENCES

[1] S. N. Yang, S. WeiHo, Y. BingLin and C. HienGan.(2016), "Amulti-RAT bandwidth aggregation mechanism with software-defined networking," *Journal of Network and Computer Applications 61, 189-198.*

[2] M. Karakus and A. Durresi.(2017), "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," *Journal of Network and Computer Applications 80, 200-218.*

[3] C. Xu, B. Chen and H. Qian.(2015), "Quality of Service Guaranteed Resource Management Dynamically in Software Defined Network," *Journal of Communication 10 (11), 843-850.*

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss.(1998), "An Architecture for Differentiated Services," Lucent Technologies, [Online]. Available: https://tools.ietf.org/html/rfc2475. [Accessed 17. 05. 2019].

[5] S. Tomovic, N. Prasad and I. Radusinovic.(2014), "SDN control framework for QoS provisioning," in *22nd Telecommunications forum TELFOR*, Serbia, Belgrade.

[6] K. Govindarajan, K. C. Meng, H. Ong, W. M. Tat, S. Sivanand and L. S. Leong.(2014), "Realizing the Quality of Service (QoS) in Software-Defined Networking (SDN) Based Cloud Infrastructure," in *2nd International Conference on Information and Communication Technology (ICoICT)*, Kuala Lumpur, Malaysia.

[7] "Welcome to RYU the Network Operating System(NOS).(2011)," Nippon Telegraph and Telephone Corporation, [Online]. Available: https://ryu.readthedocs.io/en/latest/#. [Accessed 10. 02. 2019].

[8] D. Kreutz, F. M. V. Ramo, P. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig.(2015), "Software-Defined Networking:A Comprehensive Survey," *IEEE, 103 (1), 14-76* .

[9] R. Toghraee.(2017), Learning OpenDaylight, Birmingham, Mumbai: Packt Publishing Ltd..

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner.(2008), "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review, 38 (2)* .

[11] A. R. Sharafat, S. Das, G. Parulkar and N. McKeown.(2011), "MPLS-TE and MPLS VPNs with OpenFlow," *ACM SIGCOMM Computer Communication, 41 (4), 452-453.*

[12] T. O. N. Foundation, "SDN Definition," ONF, [Online]. Available: https://www.opennetworking.org/sdn-definition/. [Accessed 12. 08. 2019].

[13] W. Braun and M. Menth.(2014), "Software-Defined Networking Using OpenFlow: Protocols,," *Future Internet, 6 (2), 302-336.*

[14] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie.(2015), "A Survey on Software-Defined Networking," *IEEE Communication Surveys & Tutorials, 17 (1)* .

[15] R. Masoudi and AliGhaffari.(2016), "Softwaredefined networks:Asurvey," *Journal of Network and Computer Applications, 67.*

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes and P. Yalagandula.(2011), "DevoFlow: Scaling Flow Management for High-Performance Networks," *ACM SIGCOMM Computer Communication , 41 (4), 254-265.*

[17] U. C. Kozat, G. Liang and K. Kökten.(2013), "Verifying Forwarding Plane Connectivity and Locating Link Failures using Static Rules in Software Defined Networks," *ACM SIGCOMM.*

[18] N. Gude, T. Koponen, J. Pettit, B. Pfaff and M. Casado.(2008), "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication.*

[19] C.Fancy and M.Pushpalatha.(2017), "Performance Evaluation of SDN controllers POX and Floodlight in Mininet Emulation Environment," in *Proceedings of the International Conference on Intelligent Sustainable Systems ICISS* .

[20] S. Badotra and J. Singh.(2017), "Open Daylight as a Controller for Software Defined Networking," *International Journal of Advanced Research in Computer Science, 8 (5)* .

[21] H. Akcay and D. Yiltas-Kaplan.(2017), "Web-Based User Interface for the Floodlight SDN Controller," *Int. J. Advanced Networking and Applications, 8 (5), 3175-3180.*

[22] F. Yamei, L. Qing and H. Qi3.(2016), "Research and Comparative Analysis of Performance Test on SDN Controller," in *First IEEE International Conference on Computer Communication and the Internet*, Beijing, China.

[23] F. Bannour, S. Souihi and A. Mellouk.(2018), "Distributed SDN Control: Survey, Taxonomy, and Challenges," *IEEE Communications Surveys & Tutorials, 20 (1), 333-354.*

[24] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao.(2013), "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine, 51 (7), 36-43.*

[25] I. T. Union.(2014), "Framework of Software Defined Networking," [Online]. Available: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.3300-201406-I!!PDF-E&type=items . [Accessed 10. 07. 2019].

[26] T. O. N. Foundation.(2015), "Framework for SDN: Scope and Requirements," ONF, [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/Framework_for_SDN_-Scope_and_Requirements.pdf. [Accessed 15. 07. 2019].

[27] C. Y. Hong, S. Mandal, M. Alfares, M. Zhu, R. Alimi, K. N. B, C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong and A. Vahdat.(2018), "B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN," in *SIGCOMM '18 Proceedings of the 2018 Conference*

of the ACM Special Interest Group on Data Communication, 74-87, Budapest, Hungary.

[28] T. O. N. Foundation.(2015), "OpenFlow Switch Specification," ONF, [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Accessed 01. 05. 2019].

[29] T. O. N. Foundation.(2014), "OpenFlow Switch Specification version 1.3.0," ONF, [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf. [Accessed 15. 07. 2019].

[30] L. Foundation, "Open vSwitch," Linux Foundation Collaborative Project, [Online]. Available: https://www.openvswitch.org/. [Accessed 20. 07. 2019].

[31] A. Mendiola, J. Astorga, E. Jacob and M. Higuero.(2017), "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering," *IEEE Communications Surveys & Tutorials, 19 (2), 918-953.*

[32] L. F. C. Projects, "Open vSwitch," [Online]. Available: http://www.openvswitch.org/support/dist-docs-2.5/. [Accessed 25. 07. 2019].

[33] L. Foundation, "Open vSwitch," Linux Foundation Projects, [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/openvswitch/latest/openvswitch.pdf. [Accessed 26. 07. 2019].

[34] B. Pfaff and B. Davie.(2013), "The Open vSwitch Database Management Protocol," VMware, [Online]. Available: https://tools.ietf.org/html/rfc7047. [Accessed 05. 02. 2019].

[35] F. L. Rodríguez, U. S. Dias, D. R. Campelo, R. d. O. Albuquerque, S.-J. Lim and L. J. G. Villalba.(2019), "QoS Management and Flexible Traffic Detection Architecture for 5G Mobile Networks," *Sensors.*

[36] 3. T. S. G. Services.(2018), "System Architecture for the 5G System," 3GPP, [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/23_series/23.501/. [Accessed 01. 08. 2019].

[37] H. Ghalwash and C.-H. Huang.(2018), "A QoS framework for SDN-based Networks," in *IEEE 4th International Conference on Collaboration and Internet Computing*, Connecticut, USA.

[38] R. Braden, D. Clark and S. Shenker.(1994), "Integrated Services in the Internet Architecture: an Overview," [Online]. Available: https://tools.ietf.org/html/rfc1633. [Accessed 10. 05. 2019].

[39] J. Kurose and K. Ross.(2017), "DiffServ," in *Computer Networks A Top Down Approach (Seventh Edition)*, USA, Pearson , pp. 795-801.

[40] C.-Y. H. (UIUC), S. Kandula, R. Mahajan and M. Zhang.(2013), "Achieving High Utilization with Software-Driven WAN," Microsoft, Hong Kong, China.

[41] C. Caba and J. Soler.(2015), "APIs for QoS configuration in Software Defined Networks," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, London, UK.

[42] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster and Y.-Q. Song.(2014), "FlowQoS: QoS for the Rest of Us," *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 207-208.*

[43] Y. Jinyao, Z. Hailong, S. Qianjun, L. Bo and G. Xiao.(2015), "HiQoS: An SDN-Based Multipath QoS Solution," *China Communication, 12 (5), 123-133.*

[44] H. E. Egilmez, S. T. Dane, K. T. Bagci and A. M. Tekalp.(2012), "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*, Hollywood, CA, USA.

[45] H. E. Egilmez and A. M. Tekalp.(2014), "Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks," *IEEE Transactions on Multimedia, 16 (6), 1597-1609.*

[46] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, R. Figueiredo, D. Morris, M. Pickavet and P. Demeester.(2014), "Implementing Quality of Service for the

Software Defined Networking Enabled Future Internet," in *2014 Third European Workshop on Software Defined Networks*, London, UK.

[47] D. Palma, J. Goncalves, B. Sousa, L. Cordeiro, P. Simoes, S. Sharma and D. Staessens.(2014), "The QueuePusher: Enabling Queue Management in OpenFlow," in *2014 Third European Workshop on Software Defined Networks*, London, UK.

[48] B. Sonkoly, A. Gulyás, F. Németh, J. Czentye, K. Kurucz, B. Novák and G. Vaszkun.(2012), "On QoS Support to Ofelia and OpenFlow," in *2012 European Workshop on Software Defined Networking*, Darmstadt, Germany.

[49] R. Durner, A. Blenk and W. Kellerer.(2015), "Performance study of dynamic QoS management for OpenFlow-enabled SDN switches," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, Portland, OR, USA.

[50] A. V. Akella and K. Xiong.(2014), "Quality of Service (QoS) Guaranteed Network Resource Allocation via Software Defined Networking (SDN)," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, Dalian, China.

[51] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer and J. A. G. Espín.(2013), "An OpenNaaS based SDN Framework for Dynamic QoS control," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, Barcelona, Spain.

[52] G. Pujolle.(2015), "New generation protocols," in *Software Networks Virtualization, SDN, 5G and Security*, Wiley and ISTE, pp. 83-87.

[53] R. Izard.(2015), "How to Use OpenFlow Queues," Big Switch, [Online]. Available: https://floodlight.atlassian.net/wiki/x/DoDl. [Accessed 03. 08. 2019].

[54] M. A. Brown, F. Bolelli and N. Patriciello.(2016), "Traffic Control HOWTO," Free Software Foundation, [Online]. Available: http://tldp.org/en/Traffic-Control-HOWTO/index.html. [Accessed 15. 05. 2019].

[55] M. D. a. devik.(2002), "HTB Linux queuing discipline manual - user guide," [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/old/htbmeas1.htm. [Accessed 18. 05. 2019].

[56] T. D. Nadeau and K. Gray.(2013), "SDN Controllers," in *Software Defined Networks*, O'Reilly, pp. 92-94.

[57] K. Haldar and D. P. Agrawal.(2014), "QoS Issues in OpenFlow/SDN," in *Network Innovation through OpenFlow and SDN*, Boca Raton, Florida, CRC Press, pp. 256-268.

[58] K. Kaur, J. Singh and N. S. Ghumman.(2014), "Mininet as Software Defined Networking Testing Platform," in *International Conference on Communication, Computing & Systems (ICCCS–2014)*, Macau, China.

[59] "Teaching and Learning with Mininet," Github, [Online]. Available: https://github.com/mininet/mininet/wiki/Teaching-and-Learning-with-Mininet. [Accessed 03. 01. 2019].

[60] "Iperf-The ultimate speed test tool for TCP, UDP and SCTP," [Online]. Available: https://iperf.fr/contact.php. [Accessed 03. 03. 2019].

[61] G. Combs, "Wireshark," [Online]. Available: https://www.wireshark.org/about.html#authors. [Accessed 03. 03. 2019].