Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Master's Thesis

End-to-end dynamic bandwidth resource allocation in SDN

Author

Arpana Shanta

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. Arpana Shanta

Informatics

Thesis title

End-to-end dynamic bandwidth resource allocation in SDN

Objectives of thesis

The thesis aims to investigate if SDN can deliver standard Quality of service (QoS) to specific hosts. The research will establish that an End-to-end bandwidth guarantee between hosts should be possible with QoS in SDN.
To attain a high level of QoS, the following functions ought to be applied:

1. Guaranteed prioritized traffic on the configured fixed limits.
2. By providing necessary bandwidth resources and improving utilization of the underlying infrastructure.
3. Bandwidth guarantee for various services between hosts.
4. Reducing the packet loss rate of QoS flows will increase the performance.

Methodology

The theoretical part of the work is based on the study and analysis of professional and scientific information sources.

Building a QoS routing mechanism within an SDN-based core transport network to assure QoS in terms of packet loss, delay, jitter, and bandwidth. An experiment will also be conducted to assess the suggested QoS model's performance. As a result of the thesis, a proper theoretical technique will be developed to obtain QoS, and suitable tests will be selected and performed to demonstrate that QoS can be achieved with SDN.

In this network emulation, OpenFlow will be used as the standard communication interface between the control and forwarding layers of an SDN network. Open vSwitch (OVS) will also be used to interconnect different virtual machines within a host and virtual machines across the network. To create a realistic virtual network with a networking component, the Mininet network emulation tool will be used.

Based on the synthesis of knowledge of the theoretical part and evaluation of the results of the practical part, the conclusions of the work will be formulated.

The proposed extent of the thesis

50-60 pages

Keywords

Quality of Service (QoS), Software Defined Networking (SDN), Mininet, Wireshark, Bandwidth

Recommended information sources

Aljawad, Ahmed & Shah, Purav & Gemikonaklı, Orhan & Trestian, Ramona. (2018). Policy-based QoS Management Framework for Software-Defined Networks. 1-6. 10.1109/ISNCC.2018.8530994.

A. V. Akella and K. Xiong, "Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN)," 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, 2014, pp. 7-13, doi: 10.1109/DASC.2014.11.

H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in Proc. Signal Inf. Process. Assoc. Summit Conf., pp. 1-8, Dec. 2012.

J.M.Boley, E.S. Jung, and R.Kettimuthu, "Adaptive QoS for data transfers using software-defined networking." 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), IEEE, pp. 1-6, 2016.

K. Greene, (2009), "TR10: Software-defined networking. MIT Technology Review, March/April 2009" http://www2.technologyreview.com/article/ 412194 /tr10-software-defined-networking/

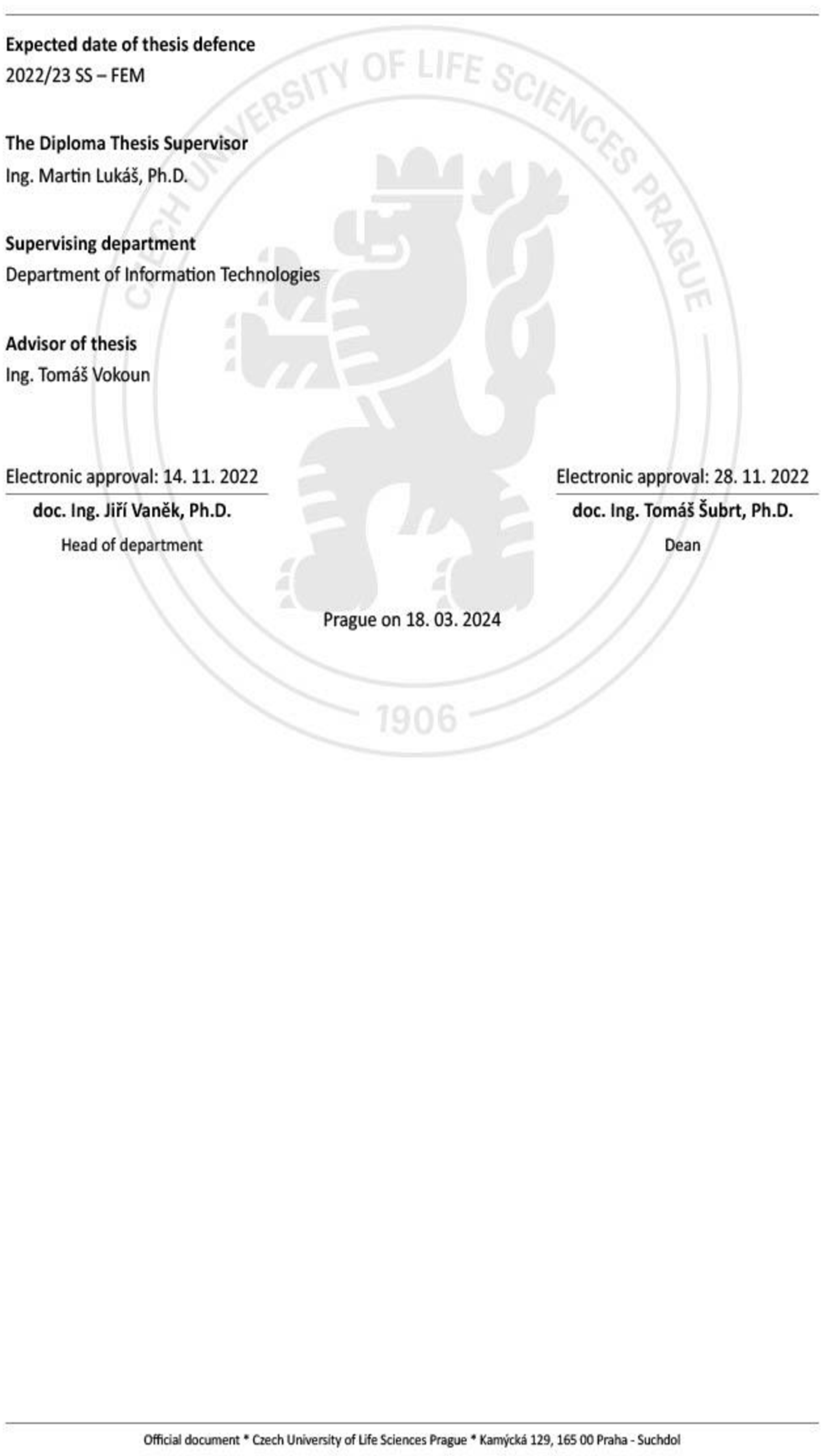
M. Jarschel, F. Wamser, T. Hohn, T. Zinner and P. Tran -Gia, " SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming," In the Proceedings of the Second European Workshop on Software Defined Networks (EWSDN), pp. 87-92, Berlin, Germany, Oct. 2013.

N. Thazin, K. M. Nwe and Y. Ishibashi, "End-to-End Dynamic Bandwidth Resource Allocation Based on QoS Demand in SDN," 2019 25th Asia-Pacific Conference on Communications (APCC), 2019, pp. 244-249, doi: 10.1109/APCC47188.2019.9026511.

P. Goransson, and B. Chuck. "Software-Defined Networks A Comprehensive Approach." In IEEE Communication Surveys & Tutorials, pp. 7-17, 2014.

P. Jha, "End-to-end Quality-of-Service in Software Defined Networking" by University of Dublin, Trinity College," no. September, thesis, 2017.

S. U. Baek, C. H. Park, E. Kim and D. Shin, "Implementation and verification of QoS priority over software-defined networking," In Proceeding of the International Conference on Internet Computing (ICOMP). The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.



Expected date of thesis defence
2022/23 SS – FEM

The Diploma Thesis Supervisor
Ing. Martin Lukáš, Ph.D.

Supervising department
Department of Information Technologies

Advisor of thesis
Ing. Tomáš Vokoun

Electronic approval: 14. 11. 2022	Electronic approval: 28. 11. 2022
doc. Ing. Jiří Vaněk, Ph.D.	**doc. Ing. Tomáš Šubrt, Ph.D.**
Head of department	Dean

Prague on 18. 03. 2024

Declaration

I announce that I have worked on my master's thesis entitled "Comprehensive evaluation of software-defined network deployment using the OpenFlow protocol" and have used only the references listed at the end of the thesis. I declare that this thesis does not breach any person's copyrights.

In Prague on 31.03.2024 ____________________________

Acknowledgment

I extend my sincerest gratitude to my master's thesis supervisor, Ing. Martin Lukáš, Ph.D., and consultant Ing. Tomáš Vokoun, for their unwavering guidance and invaluable insights throughout the development of this thesis. Their expertise and continuous support were pivotal in the successful completion of this work.

My heartfelt appreciation also goes to Goutam Kumar Saha, whose encouragement and motivation played a crucial role during my thesis journey. Your assistance and support were indispensable, and I am deeply grateful for having you by my side throughout this process.

Most importantly, I owe my deepest gratitude to my parents. You not only brought me into this world but also nurtured me with your teachings, blessings, and unconditional love. Your faith in me and your endless support have been my cornerstone in every step of my life. I stand here today because of you, and for that, I am eternally thankful.

End-to-end dynamic bandwidth resource allocation in SDN

Abstract

This thesis presents a detailed exploration into enhancing network efficiency and performance through Quality of Service (QoS)-based traffic engineering within Software Defined Networking (SDN). At its core, the research seeks to address the growing demands on network infrastructure by deploying dynamic bandwidth allocation, prioritized flow management, and innovative routing strategies to facilitate high-priority traffic handling. Leveraging the centralized control and flexibility of SDN, the study introduces an architecture capable of dynamically responding to network conditions, thereby improving congestion management and throughput. Through a comprehensive comparative analysis with traditional and multipath routing methodologies, supported by simulations in an emulated SDN environment, the effectiveness of the proposed approach is validated. Results underscore significant improvements in critical network performance metrics, such as throughput, delay, jitter, and packet loss, indicating a substantial potential for enhanced network performance and QoS fulfilment.

Keywords:
Software Defined Networking (SDN), Quality of Service (QoS), Dynamic Bandwidth Allocation, Network Performance Metrics, Traffic Engineering, Congestion Management, Flow Management, SDN Controllers, Open vSwitch (OVS), Network Simulation, Routing Strategies, VoIP Traffic, TCP and UDP Traffic, Network Efficiency, Emulated Networking Environments.

Alokace dynamické šířky pásma z bodu na bod v SDN

Abstrakt

Tato diplomová práce představuje podrobné zkoumání zvyšování efektivity a výkonnosti sítě prostřednictvím inženýrství provozu založeného na Kvalitě služeb (QoS) v rámci Software Defined Networking (SDN). Ve své podstatě se výzkum snaží řešit rostoucí požadavky na síťovou infrastrukturu nasazením dynamické alokace šířky pásma, prioritního řízení toků a inovativních strategií směrování za účelem usnadnění zpracování provozu s vysokou prioritou. Využíváním centralizovaného řízení a flexibilitu SDN, studie představuje architekturu schopnou dynamicky reagovat na podmínky v síti, čímž se zlepšuje řízení zácpy a propustnost. Prostřednictvím komplexní komparativní analýzy s tradičními a multipath metodami směrování, podporované simulacemi v emulovaném prostředí SDN, je potvrzena účinnost navrhovaného přístupu. Výsledky zdůrazňují významné zlepšení v kritických metrikách výkonnosti sítě, jako jsou propustnost, zpoždění, jitter a ztráta paketů, což naznačuje značný potenciál pro zvýšenou výkonnost sítě a splnění QoS.

Klíčová slova:
Software Defined Networking (SDN), Kvalita služby (QoS), Dynamická alokace šířky pásma, Metriky výkonnosti sítě, Inženýrství provozu, Řízení zácpy, Řízení toků, Kontroléry SDN, Open vSwitch (OVS), Simulace sítě, Strategie směrování, VoIP provoz, TCP a UDP provoz, Efektivita sítě, Emulované síťové prostředí.

Table of content

List of figures

List of tables

1. Introduction

In the realm of network engineering, Quality of Service (QoS) emerges as a pivotal element for augmenting the delivery and reliability of digital communication systems. As networks grow increasingly complex, accommodating a spectrum of services from Voice over IP (VoIP) to high-definition video streaming, the demand for advanced network management strategies is undeniable. This thesis ventures into the implementation of QoS within the dynamic landscape of Software-Defined Networking (SDN), emphasizing the strategic deployment of Open vSwitch (OVS) queues for precise network traffic management, aimed at sustaining high performance across diverse application requirements.

The research outlines an orchestrated approach to QoS in SDN through the creation of specialized OVS queues, each crafted to manage distinct types of network traffic such as TCP, UDP, and VoIP. These queues effectively distribute resources, akin to service lines in a complex service ecosystem, prioritizing customer satisfaction at various service echelons. The analogy drawn between service lines and network traffic management illustrates how SDN's dynamic bandwidth allocation and traffic prioritization can substantially uplift the quality of network services.

At the core of this exploration is the formulation and implementation of an advanced traffic classification system. This system is designed to accurately identify and sort incoming traffic, guiding it towards the appropriate queue based on specific characteristics. Such a mechanism is vital for ensuring that each type of traffic receives adequate bandwidth and priority, thereby optimizing overall network performance.

Through empirical testing and simulations in a controlled SDN setting, the study rigorously assesses the efficiency of the proposed QoS strategies. Although specific findings are not discussed, the research framework and methodologies applied reflect a comprehensive evaluation of how QoS mechanisms within SDN can potentially refine critical network performance metrics.

This thesis explores QoS in SDN, enriching both theory and practice for network professionals. It links concepts to applications, showing how QoS transforms network management. The study advances a more streamlined, resilient, and user-focused network ecosystem, highlighting QoS's key role in network engineering.

2. Objectives and Methodology

2.1. Objectives

The thesis aims to investigate if SDN can deliver standard Quality of service (QoS) to specific hosts. The research will establish that an End-to-end bandwidth guarantee between hosts should be possible with QoS in SDN.

To attain a high level of QoS, the following functions ought to be applied:

1. Guaranteed prioritized traffic on the configured fixed limits.
2. By providing necessary bandwidth resources and improving utilization of the underlying infrastructure.
3. Bandwidth guarantee for various services between hosts.
4. Reducing the packet loss rate of QoS flows will increase performance.

2.2 Methodology

The theoretical part of the work is based on the study and analysis of professional and scientific information sources.

Building a QoS routing mechanism within an SDN-based core transport network to assure QoS in terms of packet loss, delay, jitter, and bandwidth. An experiment will also be conducted to assess the suggested QoS model's performance. As a result of the thesis, a proper theoretical technique will be developed to obtain QoS, and suitable tests will be selected and performed to demonstrate that QoS can be achieved with SDN.

In this network emulation, OpenFlow will be used as the standard communication interface between the control and forwarding layers of an SDN network. Open vSwitch (OVS) will also be used to interconnect different virtual machines within a host and virtual machines across the network. To create a realistic virtual network with a networking component, the Mininet network emulation tool will be used.

Based on the synthesis of knowledge of the theoretical part and evaluation of the results of the practical part, the conclusions of the work will be formulated.

3. Literature Review

3.1 Traditional network

In traditional networks, various interconnected devices exchange data. Within these devices, such as switches and routers, both the control plane, which determines what to do with incoming packets, and the forwarding plane, which actually moves the packets along their path, are housed within the same hardware unit. This means the decision-making capabilities and the data transmission functions coexist in a single device.

3.2 Software-defined network- a new paradigm

In the early 2000s, rising traffic volumes and the need for more dependable and efficient networks led operators to seek better ways to manage tasks like directing traffic, a practice known as traffic engineering. Traditional methods for routing and traffic control, while straightforward, were no longer sufficient. This led to the emergence of Software-Defined Networking (SDN), a sophisticated approach that revolutionized network technology.

One of the key features that distinguishes SDN and contributes to its popularity is its capacity to make networks programmable. This is achieved by separating the control plane, responsible for network decision-making, from the data plane, which handles the actual traffic forwarding – a contrast to conventional network devices where these two planes are integrated (1). Software-Defined Networking (SDN) has emerged as a potent alternative in the networking realm.

Software-Defined Networking (SDN) presents a solution to various issues faced in traditional networking. It addresses challenges such as scaling networks, surging traffic demands, and complexities in network troubleshooting. SDN also overcomes the rigidity of older systems by introducing programmability to network management. With SDN, network operators can manage, track, and monitor network devices through software applications, thereby enhancing network reliability (2)

Centralized Architecture

The transformative idea of programmable networks is at the heart of Software-Defined Networking's (SDN) innovation. SDN's architecture supports modern network needs through its dynamic, manageable, and highly adaptable framework, fostering novel ways to design and manage networks (3). The specific architecture of SDN will be thoroughly discussed in the subsequent sections of the document.

SDN layers in OSI Model

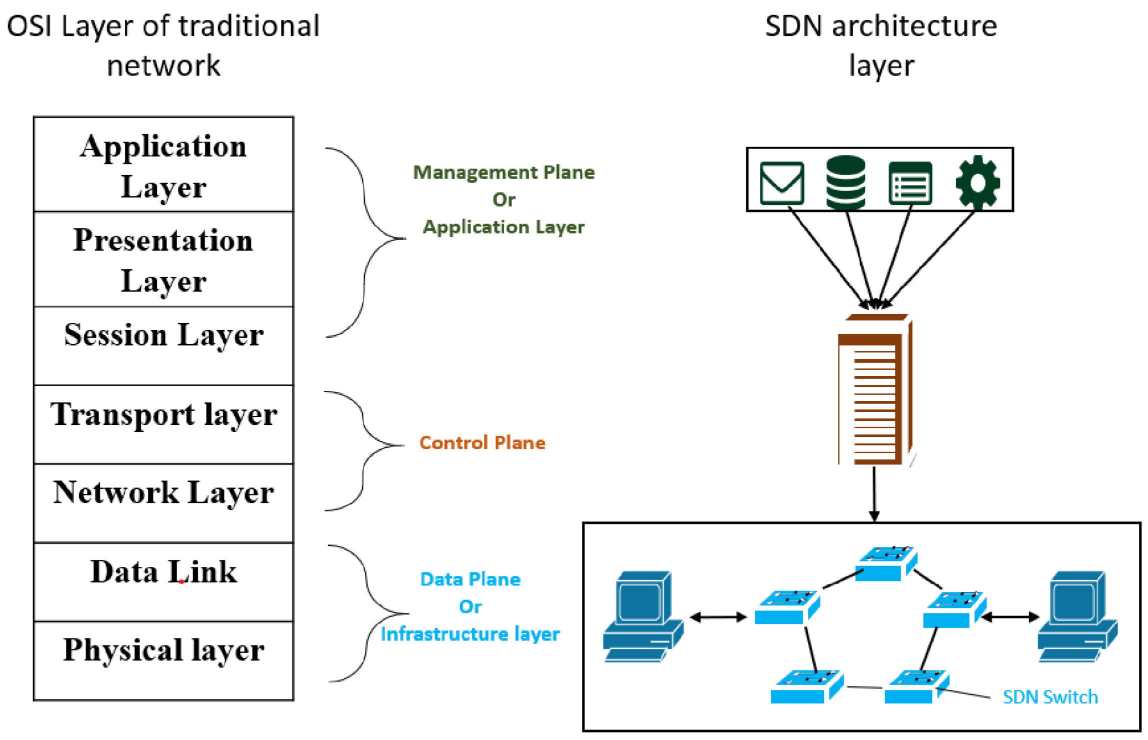


Figure 1: SDN layer relation with OSI model (own work)

In figure 2, The comparison between SDN layers and the traditional OSI model reveals that in SDN, the data link layer and the physical layer correspond to the data plane, also known as the infrastructure layer.

The SDN control plane performs functions analogous to the OSI model's transport and network layers, while the OSI model's top three layers equate to the SDN's application layer, or management layer. Although SDN is application-centric, it doesn't merely fit within the OSI's application layer; rather, it's a distinct network architecture with separate, clearly defined layers.

Characteristics of SDN architecture

Following are some characteristics of SDN architecture:

Table 1: Advantages of SDN

Characteristics	Advantage
Programmability	Network control can be directly programmable as it is decoupled from the forwarding functions.
Agile	It enables the network administrator to dynamically adjust traffic flow on the network to meet the network demand. This process can also become intelligent by implementing machine learning algorithms on the application and controller layer.
Centrally controlled	In software-based controllers, the network features can be consolidated with a global network vision. This controller is considered to be a single logical switch to the program and policy engines.
Programmable configuration	Because the SDN programs do not depend on proprietary software, network administrators can write their programs to configure, manage, secure, and optimize network resources. Because the SDN programs are automated and dynamic this can be done very quickly as well.
Open-standard-based and vendor-neutral	SDN is developed and implemented by the network community and is initiated by many volunteer sources. SDN simplifies the architecture and function of the network by introducing Open Standards, as it is not vendor-specific equipment and protocols. (4)

Table 2: Disadvantages of SDN

Characteristics	Disadvantages
Centralized control	Without redundancy, the single controller is a single point of failure. If the controller becomes not functional, the whole network will be affected.
Single point of control	With the controller being the single point of control, all the updates happen from the controller hence it might be the bottleneck of the

	network. Pushing every small update to all the NEs from the controller might end up in huge overhead.

Components of SDN

Data Plane: In the 2014 Open Networking Foundation (ONF) paper, the infrastructure layer of the SDN architecture is defined as comprising various network elements. This layer typically includes different infrastructure nodes that support SDN protocols, which are tasked with transporting and processing data packets under the guidance of instructions from the SDN control plane.
The SDN controller communicates decisions and actions through the Data-Controller Plane Interface (D-CPI) to infrastructure nodes. D-CPI specifies the method of communication and direction-taking from the control plane to the controller. This process enables network elements within the data or forwarding plane to implement necessary modifications, aligning with user or network requirements. The exchange of information between the controller and the data plane encompasses control instructions, routing policies, and resource configurations, ensuring coherent network operation and management (1).

Data plane resources serve as the manifestations of physical network elements and their functionalities. Essentially, the data plane consists of a network of nodes capable of manipulating traffic through actions such as consuming, producing, storing, dropping, or forwarding it. Network Elements (NEs), interconnected by links, act as the interface of the network, connecting clients and other nodes through external data plane ports. Highlighting an SDN advantage, it's noteworthy that a single controller has the capacity to manage multiple data planes, enhancing network flexibility and control efficiency (1).

The data plane supports a variety of models, including: (5)

- Protocols like IPv4, IPv6, and Ethernet for packet forwarding.
- Optical and circuit switching with technologies like MPLS.
- Wireless integrations, detailing flows, and handovers.
- LTE and advanced packet core for high-speed mobile data. (6).

Networking elements (Forwarding plane): The foundational layer of SDN architecture consists of Network Elements (NEs), which include traditional hardware such as switches and routers. These NEs are designed to support programmable interfaces like OpenFlow, enabling them to integrate seamlessly with the dynamic and flexible nature of SDN environments. (see section 3.2). These devices within the SDN architecture can either be physical hardware switches, offering high performance and greater bandwidth, or software-based switches, such as Open vSwitch (see section 3.2) While hardware switches excel in delivering robust performance, software switches stand out for their adaptability, providing the needed flexibility to accommodate rapid changes in network conditions (7).

Southbound interface: The southbound interface enables SDN controllers to communicate with forwarding devices, sending packet instructions, and managing alarms via protocols like SNMP and OpenFlow. Another key protocol in this setup is OVSDB, which aids in the detailed management of network elements (8).

Control Plane: The SDN controller represents the control plane, functioning as the "brain" of the SDN architecture. Operating on dedicated hardware or servers, this software layer is positioned above the data or infrastructure layer. Its main role is to direct the forwarding plane, processing information from and sending instructions to it regarding packet routing and handling. Communication with the data plane is facilitated through southbound interfaces. In complex networks, multiple software controllers may collaborate or operate within a group, managing control information across various clusters of Network Elements (NEs) (4).

Having just one controller in a network introduces a single point of failure, posing a risk to the entire network's stability. If this lone controller fails or becomes non-functional, it can halt the operation of the entire network. To mitigate this issue, networks can be designed with multiple controllers, enhancing redundancy and reliability. A network operator can design controller setup by following:

1. Centralized – A network configuration can include a single, centralized controller that maintains a comprehensive, global view of the network. This model simplifies network management by centralizing decision-making and control. Such a setup is particularly suited for small to medium-sized networks, where the ease of managing the network through a centralized point outweighs the risks of a single point of failure.

2. Distributed- A setup with multiple controllers in the network introduces redundancy, resiliency, and improved performance. These controllers may independently manage the entire network or specific segments of it. They communicate with each other upon receiving a packet to establish an end-to-end route beyond their individual domains, enhancing the network's overall efficiency and fault tolerance.

Centralized SDN architecture is most common, with controllers designed for high concurrency using several threads. This setup suits most networks, excluding large-scale ones like data centers, by utilizing multi-core systems. Additionally, application isolation in these controllers improves resiliency.

Network Operating System: The networking operating system, often referred to as the SDN controller, powers the foundational services necessary for network operation. Essential services provided include topology management, inventory and statistics service, and host tracking. These services together maintain a topological graph detailing the interconnections between Network Elements (NEs), facilitating efficient network management and operation.

Switches can be instructed to use Link Layer Discovery Protocol (LLDP) packets to ascertain the location and positioning of nodes within the network, thereby mapping out the network topology. Additionally, specialized packets can be deployed to retrieve various details about the Network Elements (NEs), such as their OpenFlow version, capacity, and capabilities. A statistics service can further collect data regarding the incoming and outgoing traffic on specific interfaces, flow counters, and flow table information. Moreover, a host tracker can identify any NE within the network by intercepting traffic flows and utilizing the IP or MAC addresses of the hosts, often in coordination with the virtual machine platform (7).

Application-controller plane interface (A-CPI): In the southbound architecture, the Application-Controller Plane Interface (A-CPI) allows SDN network applications to integrate with the SDN controller. A key role of A-CPI is to offer a simplified abstraction of the network infrastructure to higher layers, often presenting the underlying network as a single large switch to applications. For effective interaction between applications and the controller, native plugins running alongside controllers may be necessary. These interactions are facilitated through a programming language-based API, enabling applications to use directive API calls to manage network behaviour and gather network data.

SDN Application: SDN network applications are empowered to dictate network behaviour, enforce network policies, and carry out a range of network functions. Network administrators have the capability to develop network programs tailored to their organization's specific needs and requirements. SDN facilitates this by offering a programmable abstraction layer, thereby enabling customizable and flexible network management (7).

3.3 OpenFlow

OpenFlow, the key communication protocol for SDN, catalysed its development and is standardized by the Open Networking Foundation (ONF) (1). The Open Networking Foundation (ONF), a user-operated organization, established OpenFlow as the inaugural standard communication interface for SDN. ONF's specifications detail the interaction and communication methods between the control and forwarding layers in an SDN architecture. Being an open-source initiative, ONF promotes and supports the development of various SDN applications by its user community, driving the evolution and maturation of SDN technology.

Architecture of OpenFlow

In figure 4, The architecture of an OpenFlow network primarily consists of OpenFlow-compatible switches and a controller. OpenFlow switches evolve the concept of the traditional Ethernet switch by segregating the data plane from the control plane, a separation achieved through a flow table mechanism. Communication between these switches and the SDN controller occurs over a secure channel, ensuring that control messages are transmitted securely within the network infrastructure (4).

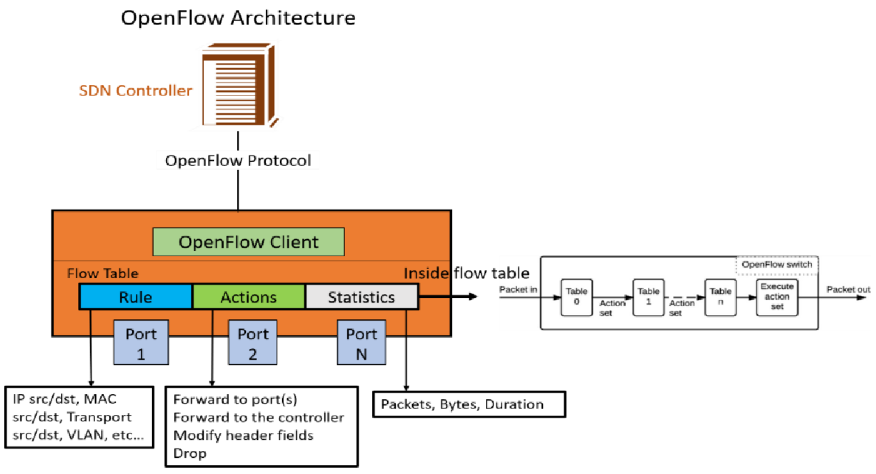


Figure 2: OpenFlow Architecture (own work)

OpenFlow transforms data plane network elements into straightforward devices capable of processing packets solely based on the directives from the controller. It enables a centralized control model, distinguishing it from other interfaces like ForCES (Forwarding and Control Element Separation), which supports distributing logic across one, several, or all network elements for more flexibility in control. The primary goal of the OpenFlow protocol is to consolidate control plane functionalities, offering centralized network management and decision-making capabilities. (9).

The architecture of the OpenFlow protocol is built around three main components:

- The Data Plane, composed of OpenFlow switches, is responsible for forwarding packets based on rules set by the controller.
- The Control Plane is formed by the OpenFlow controller, which dictates the behaviour of the switches, managing the flow of data across the network.
- A Secure Channel serves as the communication link between the OpenFlow switches and the controller, ensuring that the data exchanged is protected. (6)

OpenFlow Switch

An OpenFlow switch comprises a flow table and a group table, with the possibility of having multiple flow tables. These tables play crucial roles in packet lookup and forwarding. Communication between the switch and the controller is secured through one or more secure channels. Utilizing the OpenFlow switch protocol, the controller can manage the switch by altering flow data within the flow tables, including adding, deleting, or updating entries. Such modifications can be made proactively or in response to incoming packets.

Each flow table contains several entries, equipped with matching fields, counters, and specific instructions for action. If an incoming packet matches these fields, the switch executes the corresponding instructions. Additionally, the switch features group tables, which comprise group entries associated with action buckets based on group types. Actions on packets are processed through these buckets, offering nuanced packet management before forwarding. (3) (4).

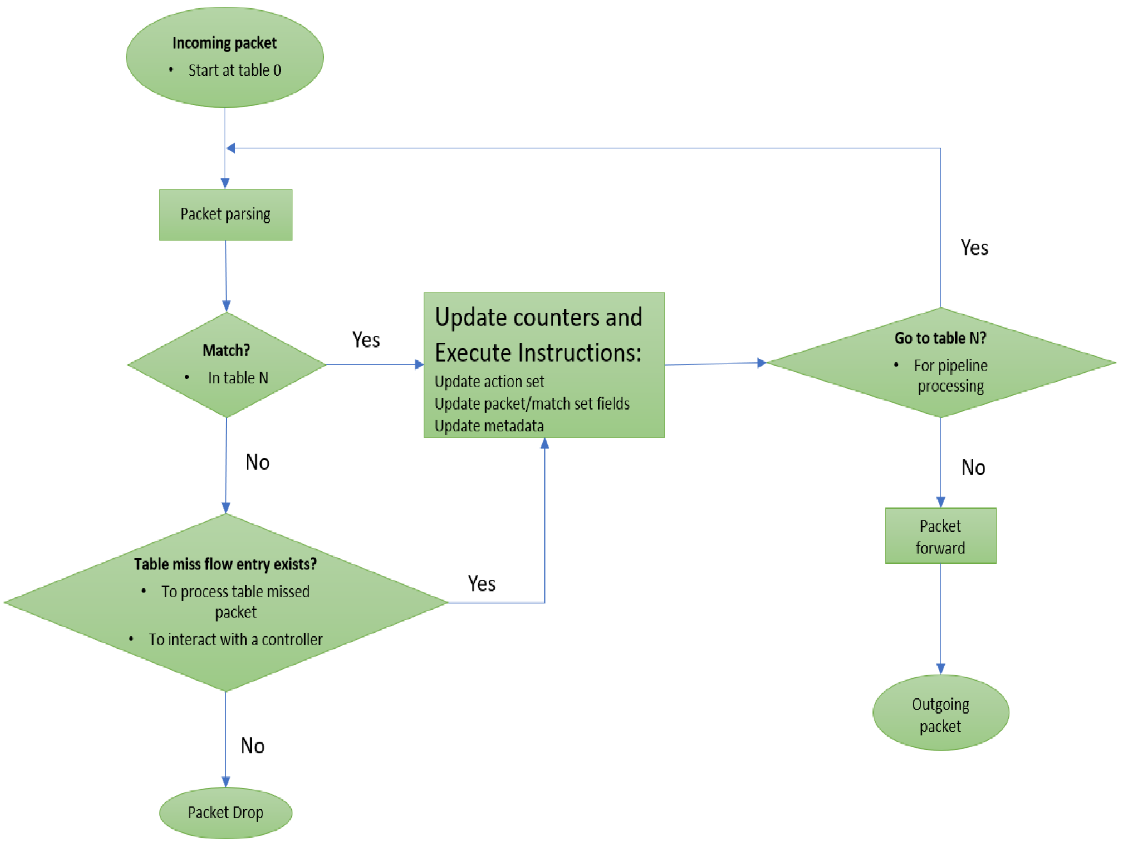


Figure 3: Flowchart detailing packet flow through an OpenFlow switch (4) (10)

OpenFlow Channel

The channel interface links OpenFlow switches to the SDN controller, enabling a single controller to connect with multiple switches and vice versa. This arrangement enhances redundancy and supports network load management. Connection initiation is switch-driven, based on configured host and port settings. After establishing a connection, two-way communication between the switch and controller is possible. OpenFlow protocol facilitates three main communication types: control-to-switch, asynchronous, and symmetric, each encompassing various specific interactions (7). Messages to switches are typically sent by the centralized controller, which may or may not receive confirmation from the switches depending on the scenario. Switches can send asynchronous messages without prior requests from the controller. Symmetric messages, allowing communication from either direction without specific prompts, ensure that either side can initiate communication whenever necessary (11).

Open vSwitch

This switch supports the OpenFlow protocol and functions as a multilayer device, capable of operating at layers 2, 3, and 4. It facilitates the connection of virtual machines within a single host as well as across different hosts via networks. Additionally, it can be utilized in dedicated switching hardware, making it a crucial component of an SDN solution. Compatible with multiple platforms, including Linux and FreeBSD, an Open vSwitch can be configured either locally or remotely (12).

OpenFlow Protocol

OpenFlow is a pivotal network communication protocol central to Software-Defined Networking (SDN), facilitating centralized network management from the SDN controller. It serves as the primary conduit between the control plane and the data plane, outlining the structure of control messages transmitted via a secure channel from the controller to OpenFlow switches. For effective communication, both the controller and the switch must understand and generate messages in the specific format prescribed by the OpenFlow protocol.

The OpenFlow protocol, integral to the OpenFlow specification, is essential for both the control plane and the switches within the OpenFlow framework. It establishes the guidelines for switch operations, which are activated under specific network traffic conditions. These conditions might include the traffic's origin port, IP or MAC addresses, or adherence to certain security protocols. The protocol's directives, known as flows, are catalogued in the switch's flow table. Both the network's controller and the switch itself can generate these flow entries. A controller has the capability to distribute a flow entry to one or several switches throughout the network. In instances where a switch encounters a packet that does not match any existing flow table entry, it forwards a detailed message about the packet to the controller. The controller then responds with an appropriate flow entry for the packet, thereby enabling control over the network's operation. Many in the field consider the OpenFlow protocol a pioneering standard within the Software-Defined Networking (SDN) arena (13).

3.4 Quality of Service (QoS)

In the realm of Software-Defined Networking (SDN), the quest for ensuring Quality of Service (QoS) is a vibrant field of exploration that has captured the attention of scholars far and wide. While the term "QoS" lacks a universally accepted definition, it sprang from the domain of telecommunications to capture essential aspects of how data is transmitted across networks. This chapter delves into the foundational idea of QoS within network systems and highlights several notable studies that have contributed to our understanding of QoS in the context of SDN. Additionally, this section introduces a structured classification of various applications alongside their specific QoS needs, offering readers a comprehensive overview of how diverse requirements drive the evolution of network services and management.

Understanding Quality of Service

The pursuit of Quality of Service (QoS) stands as a dynamic and critical area of study within Software-Defined Networking (SDN), engaging researchers globally. Although the concept of QoS does not have a single, widely accepted definition, its origins in the telecommunications field emphasize the crucial elements involved in the transmission of data over networks. This section explores the fundamental principles of QoS in networking systems, spotlighting key research contributions that have enriched our comprehension of QoS in the SDN landscape. Moreover, it presents a systematic categorization of different applications and their unique QoS demands, providing a detailed view of the diverse requirements that shape the development and management of network services. (14)

QoS Across Applications

Quality of Service (QoS) has expanded its reach, stretching from the core of the network all the way to the application layer, to guarantee that the particular requirements of various applications are in sync with network attributes such as bandwidth and latency. This strategy is pivotal in enabling networks to accommodate real-time applications, providing consistent and predictable performance essential for activities ranging from video streaming to online gaming. By aligning the demands of applications with the capabilities of the network, this advanced approach to QoS significantly improves the reliability and efficiency of digital services universally. (14)

3.5 QoS Measurement Parameters

The proposed Quality of Transmission (QT) approach is evaluated against three existing methods: conventional shortest path routing, multipath routing, and Hedera in a fat tree topology, focusing on QoS performance metrics. The key QoS parameters are (14):

I. Throughput: The success rate of message delivery across the network, measured in bits or bytes per second. Higher rates denote improved throughput.

II. Delay: The time duration for data to traverse from source to destination, usually in milliseconds. While the ITU-T recommends (ITU-T Rec. G.161 (06/2004) (15) a max one-way delay of 400 ms for overall network design, interactive applications should aim for less than 150 ms to preserve user experience. This study concentrates on transmission delay.

III. Jitter: The variability in packet arrival times, leading to potential packet loss and network congestion. It's the variation in delay between successive packets.

IV. Packet Loss: The occurrence of packets not reaching their intended endpoint, often caused by congestion or transmission errors.

The QT approach is anticipated to outperform the compared methods in throughput and packet loss reduction by ensuring efficient end-to-end bandwidth resource allocation. Additionally, its delay estimation module within the QoS routing framework aims to minimize delays for flows demanding low latency, thereby offering enhanced QoS for various traffic types.

3.6 QoS Provisioning in Traditional Network

In traditional network settings, managing Quality of Service (QoS) is essential for maintaining optimal performance across various applications, each with its unique demands for network resources. Achieving QoS involves strategies that manage how data travels through the network and how much bandwidth is available for different types of traffic. Here’s a breakdown of the main strategies and technologies used:

1. Resource Reservation: This method involves setting aside network bandwidth for specific data flows, such as a video streaming session, based on the application's QoS requirements. It ensures that these flows receive the necessary resources according to a bandwidth management policy.

2. Prioritization: In this approach, network traffic is sorted into categories, with resources allocated based on their importance. Critical data flows receive preferential treatment, ensuring they meet their QoS demands.

Applications may have diverse QoS needs, and several protocols and algorithms have been developed to address these requirements:

1. ReSerVation Protocol (RSVP): This protocol allows for the reservation of network resources by having the receiver send a request. RSVP ensures bandwidth, timing, and buffer needs are met by maintaining a temporary reservation state that needs regular renewal. (10)

2. Differentiated Services (DiffServ): DiffServ simplifies traffic management by classifying flows into a few categories and assigning each a specific treatment. It uses parts of the IP header to identify and prioritize traffic, facilitating service quality without per-flow management. (16)

3. Multi-Protocol Label Switching (MPLS): MPLS enhances data forwarding and bandwidth management by using labels in packet headers to direct traffic flows through the network.

4. Subnet Bandwidth Management (SBM): SBM focuses on the data link layer, offering a way to organize and prioritize traffic in IEEE 802 networks. (17)

5. These methods and protocols provide the foundation for delivering tailored QoS in network environments, accommodating the varied requirements of different applications and ensuring optimal performance.

Ethernet (802.3) vs SDH/PDH

Ethernet (IEEE 802.3) and Synchronous Digital Hierarchy (SDH) / Plesiochronous Digital Hierarchy (PDH) are both standards for data communication, but they serve different purposes and are used in distinct contexts within the realm of telecommunications and networking. Here's a comparative overview (18):

Origin and Purpose:

1. Ethernet (802.3): Developed originally for local area networks (LANs), Ethernet has become the most widely used method for connecting devices in wired networks. Its

simplicity, flexibility, and scalability have led to its dominance in both enterprise and home networking.

2. SDH/PDH: These are standards for telecommunication networks that are used to transmit large volumes of data over digital transport networks. PDH was developed first, and SDH was designed to succeed and improve upon PDH. Both are primarily used in the backbone of telecommunications networks to deliver high-speed data and voice services.

Technology and Operation:

1. Ethernet uses a range of technologies and protocols to enable devices to communicate over a network. It operates primarily at the physical and data link layers of the OSI model, providing services including framing, addressing, and error detection. Ethernet is known for its use of the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) method for controlling access to the network medium, although this is less relevant with the advent of full-duplex and switched Ethernet.
2. SDH/PDH operates by aggregating multiple digital signals into higher-level frames, allowing for the synchronous transmission of data across optical fiber networks. SDH and PDH differ in how they handle synchronization and data rates; SDH provides a more flexible and reliable framework for data transmission by allowing for easier multiplexing and offering better error correction and network management capabilities.

Speed and Scalability:

1. Ethernet offers a wide range of speeds, from 10 Mbps (10BASE-T) up to 400 Gbps in the latest standards, catering to various needs and network sizes. Ethernet networks are highly scalable, with the ability to connect thousands of devices within a network.
2. SDH/PDH systems were designed to support high-speed telecommunications, with PDH supporting up to 140 Mbps and SDH going much higher, up to 40 Gbps and beyond. SDH, in particular, is designed for large-scale telecommunications networks, offering high levels of flexibility and scalability.

Applications:

1. Ethernet is predominantly used in LANs, metropolitan area networks (MANs), and as part of Internet infrastructure. Its versatility makes it suitable for a wide range of applications, from office networks to industrial environments and data centers.
2. SDH/PDH is used in wide area networks (WANs), particularly in the backbone of telecommunications networks. SDH is also used in large enterprise networks and for connecting Internet service providers (ISPs) because of its reliability and high capacity.

3.7 QoS Provisioning in Software-Defined Networking

In Software-Defined Networking (SDN), traditional QoS mechanisms like IntServ and DiffServ face scalability challenges in larger networks. This difficulty stems from managing resources and traffic efficiently across distributed networks, where protocols operate on various devices such as routers and switches. Configuring these devices and updating policies to support diverse services becomes complex, especially with the need to maintain state and adapt to evolving needs using limited commands on standard hardware. As networks grow and requirements become more intricate, manual configuration adjustments become cumbersome, highlighting the limitations of traditional network management in meeting the dynamic demands of modern networks.

QoS Support in Different Versions of OpenFlow

Since its inception, OpenFlow has acknowledged the importance of Quality of Service (QoS), albeit with initial limitations. With each update, OpenFlow has enhanced its QoS capabilities. Here's a brief overview of how OpenFlow's approach to QoS evolved through its versions (19):

- OpenFlow 1.0 to 1.1: These early versions introduced support for queues that could specify minimum rates, laying the groundwork for basic QoS functionality.

- OpenFlow 1.2 and later: Starting from version 1.2, OpenFlow expanded its queue capabilities to include both minimum and maximum rates, providing more control over bandwidth allocation.

OpenFlow queues have gained widespread acceptance and are now supported by many software switch platforms, like OVS (Open vSwitch) and CPqD Of
SoftSwitch (20), as well as by hardware from vendors such as HP and Pica8. This broad support underscores OpenFlow's role in enabling QoS across diverse network environments.

Table 3: QoS Related Features in Different OpenFlow Versions (19)

OpenFlow	**Specific Features**
1.0	Enqueue action, minimum rate property for queues and new header fields
1.1	More control over VLA and MPLS
1.2	Maximum rate property for queues and controller query queues from switches
1.3	Introducing the meter table, rate-limiting and rate monitoring feature
1.4	Introducing several monitoring features
1.5	Replacing meter action to meter instruction

OpenFlow version 1.3 introduced meter tables, marking a significant step forward in achieving more refined Quality of Service (QoS) control in OpenFlow networks. While queues help manage the rate at which traffic exits a network (egress rate), meter tables offer the ability to monitor and control the rate at which traffic enters a network (ingress rate). Essentially, queues and meter tables work together, each handling a different aspect of traffic flow, making them complementary tools for network administrators.

Moreover, OpenFlow switches gained the capability to interact with the Type of Service (ToS) bits in the IP header. This field is crucial for identifying packets as part of a flow entry, allowing for more precise management of network traffic.

Together, these features empower network administrators to implement sophisticated QoS strategies in their networks, ensuring that traffic is efficiently managed to meet various service quality requirements.

3.7.1 Queues

OpenFlow initially introduced basic rate-limiting queues in version 1.0 and expanded this to include both minimum and maximum rate limits in version 1.2. While OpenFlow specifies how to use queues for traffic management, it doesn't manage the queues directly. Instead, queue management tasks like creation, deletion, and modification are handled by external protocols: OF-Config for general OpenFlow switch configuration and OVSDB for Open vSwitch configurations. Additionally, OpenFlow controllers can oversee queue operations by querying switch statistics, enabling precise control over network traffic to meet Quality of Service (QoS) goals. (21)

3.7.2 OVSDB

OVSDB, along with OF-Config, plays a crucial role in managing switch operations beyond routing, such as tunnel setup, port status monitoring, and QoS configuration. It uses a variety of tables within Open vSwitch for managing flows, ports, and QoS settings (21). Unique to OVSDB, QoS and queue tables can be adjusted regardless of their link to ports, where ports can optionally be linked to QoS settings. This setup allows for detailed traffic management, including directing specific flows to designated queues using the OpenFlow "set queue" action, effectively controlling traffic flow rates on the network. This system underlines the intricate ways in which OpenFlow and OVS manage network operations and traffic.

The OpenFlow specification outlines two key properties related to queues that help manage data flow rates within a network (22):

1. Min Rate: This is the minimum data rate guaranteed for a queue. When set, the switch ensures this rate by prioritizing traffic to fulfil the specified minimum. If multiple queues on a single port collectively exceed the port's capacity, their rates are adjusted downward proportionally to maintain fairness.

2. Max Rate: Represents the upper limit of data rate a queue can handle. Should the flow rate surpass this maximum, the switch will take measures, such as delaying or dropping packets, to adhere to this limit.

Although these principles are set forth by OpenFlow, the actual implementation is dependent on the hardware and software of the switch itself. For instance, Open vSwitch, a popular software-based switch running on Linux, employs the Linux Kernel's Traffic Control (TC) system to enforce these queue management rules. This allows for flexible and efficient management of network traffic, ensuring that data flows are regulated according to predefined minimum and maximum rates.

3.7.3 Linux Traffic Control

Linux Traffic Control (TC) is a powerful tool within the Linux Kernel for managing how data moves through a network. It's designed to optimize network performance and ensure Quality of Service (QoS) through various means (23):

1. Traffic Shaping: This feature helps control the speed of data flow, ensuring that traffic is evenly distributed over time, which helps prevent sudden spikes that could overwhelm the network.

2. Scheduling: TC can prioritize certain packets over others, ensuring that more critical data is transmitted first. This is particularly useful during large data transfers to maintain stable network performance.

3. Policing: TC can enforce network policies at the point where data enters the network, helping to ensure that traffic complies with predefined rules.

4. Dropping: It can also drop data packets that exceed the available bandwidth, either as they enter (ingress) or leave (egress) the network, to prevent congestion.

TC utilizes queuing disciplines (Qdiscs), classes, and filters to accomplish these tasks. Qdiscs manage how packets are queued for transmission, with simple ones acting as basic FIFO (First In, First Out) queues. More complex arrangements, involving classes and filters, allow for sophisticated management techniques like hierarchical queuing.

In the context of Open vSwitch (OVS), two advanced queuing disciplines, Hierarchical Token Bucket (HTB) (18) and Hierarchical Fair Service Curve (HFSC) (23), enable detailed bandwidth management and the ability to "borrow" bandwidth as needed. For this thesis, HTB will be the focus for exploring queue management strategies.

3.7.4 Hierarchical Token Bucket (HTB)

The Hierarchical Token Bucket (HTB) is a sophisticated queuing discipline that improves upon the older Class-Based Queuing (CBQ) by offering finer control over network bandwidth. It uses a multi-level token system where tokens, representing bandwidth units, are generated at a fixed rate. Packets need a token to be transmitted, ensuring a regulated flow of traffic. HTB supports multiple classes, each with its own token bucket, allowing for detailed management of different traffic types. This structure enables precise bandwidth allocation and prioritization across a network. (25)

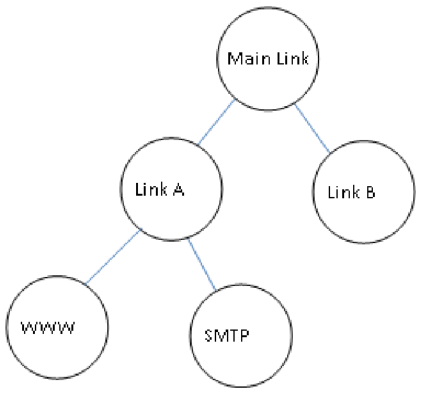


Figure 4: Sample HTB Class Hierarchy (26)

Figure 6 demonstrates a simple HTB hierarchy for solving the following problem:
"*Two customers A and B are connected to the internet via the same connection. We need to allocate 40Kbps and 60 Kbps to A and B respectively. As bandwidth needs to be subdivided into 30Kbps for WWW and 10Kbps for other applications. Any unused bandwidth should be shared among the two customers.*" (27)

In a scenario where 40Kbps is allocated to a user, A, specifically for WWW traffic, any bandwidth not used by A can be redirected to handle other traffic types, as long as the total usage doesn't surpass 40Kbps. Should A's total demand fall below this threshold, the remaining bandwidth could be allocated to another user, B. However, OpenFlow Queue's design limits hierarchy to just two levels, meaning a root class can have children, but those child classes cannot have their own sub-classes.

Key properties of HTB classes include (23):

a. Rate: The maximum guaranteed bandwidth for a class and its children, similar to a Committed Information Rate (CIR).

b. Ceil rate: The absolute maximum bandwidth a class is permitted to use.

c. Priority: Determines the order in which classes access any available extra bandwidth. A lower priority number means higher priority, but this doesn't impact the guaranteed rates of other classes.

In Open vSwitch (OVS), the `ovs-vsctl` command is used for queue creation. This command logs the queue in the OVSDB and applies it through Linux's Traffic Control (TC) system, ensuring the specified bandwidth management rules are enforced. An example of creating QoS and queues in an OVS port is shown below:

```
ovs-vsctl -- set port s1-eth1 qos = @newqos -- --id = @newqos create qos type = linux-htb
other-config:max-rate = 1000000000 queues = 0 = @q0,1 = @q1,2 = @q2 -- \
--id = @q0 create queue other-config: max-rate = 10000000 --
--id = @q1 create queue other-config:max-rate = 30000000 --
--id = @q2 create queue other-config:max-rate = 60000000
```

The above script demonstrates the process of setting up Quality of Service (QoS) and associated queues for the port `eth1` on switch `s1`. This setup involves creating new entries in both the Queue and QoS tables within the Open vSwitch Database (OVSDB). Following these additions, OVSDB establishes a link between the newly created QoS entry and the `eth1` entry in the Port table. As a result, the `eth1` port is configured to operate under the defined QoS rules.

During this configuration process, the switch utilizes the Linux Traffic Control (TC) application to automatically generate the required qdiscs (queueing disciplines) and classes in the system's background. This ensures that the traffic passing through `eth1` is managed according to the specified QoS parameters, effectively applying the defined bandwidth and priority rules to manage network traffic efficiently.

3.7.5 Meter Tables

Introduced in OpenFlow 1.3, meter tables brought a novel approach to monitoring and managing network flow rates. Unlike queues that manage the rate of outgoing traffic (egress), meter tables are designed to oversee incoming traffic (ingress) rates at the flow level (11). Here's a simplified overview:

a. Functionality: Meter tables contain entries for individual flows, allowing for the precise control of their rates. These meters are linked with flow entries rather than ports, enabling the implementation of various QoS strategies, such as rate-limiting. When combined with per-port queues, they can support complex QoS frameworks like DiffServ.

b. Operation: Flows specified to pass through a meter are subject to rate measurements and actions based on those rates, facilitated by Meter Bands. Developers have the flexibility to determine which flows should be metered, and while a flow can pass through several meters sequentially, it cannot be attached to multiple meters simultaneously.

c. Components:

I. Meter Identifier: A unique 32-bit number identifying the meter.

II. Meter Band: The component that executes actions based on the flow rate, such as dropping packets or modifying DSCP values for traffic exceeding predefined rates.

III. Counters: Used for gathering statistical data on packet processing by the meter.

IV. Meter Bands: With options for dropping excessive traffic or remarking DSCP values, meter bands enforce the desired traffic handling actions. When traffic exceeds the defined rate, the appropriate band's actions are triggered to manage the flow according to set QoS policies. (11)

This mechanism of meter tables and meter bands offers a powerful tool for managing network traffic, ensuring efficient and prioritized data flow across the network. (28)

4. Practical part (Design and Implementation)

4.1 The Proposed End-To-End QoS Implementation

The proposed end-to-end Quality of Service (QoS) strategy aims to focus on the utilization of Open vSwitch (OVS) queues to achieve Quality of Service (QoS) within a software-defined networking environment. This approach involves the creation of distinct queues for different types of traffic, each with specifically allocated bandwidth parameters to ensure optimal network performance and resource allocation. Upon the arrival of network traffic, a key process is the identification of the traffic type, which then dictates the appropriate queue placement. This methodology is pivotal in managing network congestion and ensuring that varying traffic demands are met with the requisite priority and bandwidth, thereby enhancing the overall efficiency and reliability of the network infrastructure.

4.2 Flow Requirements

Each network flow requires specific performance metrics like bandwidth, delay, or low error rates. For example, video streams need paths with sufficient capacity. To ensure these needs are met, resources may be allocated exclusively to certain flows, a key concept in Quality of Service (QoS).

The SDN controller, as the central decision-maker, manages all network flows by using network topology to guide forwarding decisions based on each flow's unique needs. It maintains a database of flow requirements and reserved resources to ensure continuous QoS for every flow, effectively keeping track of allocated paths to prevent service interruptions.

4.3 Flow Priority

In networks, traffic priorities vary based on Quality of Service (QoS) needs. An SDN controller manages this by assigning different priorities to flows, using either a priority field in flow rules or dynamic policy adjustments. High-priority traffic is crucial to maintain QoS, but network capacity limits may require prioritizing certain flows during congestion. Priorities typically range from 1 (highest) to 16, ensuring critical services remain high quality even in congested scenarios by determining which flows to reroute first.

4.4 Queue Implementation

This section covers configuring queues on switch interfaces to enhance Quality of Service (QoS) by categorizing traffic into three priority levels. High-priority queues cater to essential traffic needing more bandwidth, while lower-priority ones manage less critical traffic. There's a distinction between Soft QoS, providing flexible bandwidth without guarantees, and Hard QoS, which secures bandwidth with strict policies, potentially denying flows, if necessary, bandwidth isn't available to maintain quality.

Switch ports use three priority queues—high, medium, and low—to organize incoming traffic based on QoS needs. Services needing low latency, like voice and video, go into high-priority queues. The SDN controller dynamically reroutes flows to ease bottlenecks, ensuring top QoS demands are met first, optimizing network use and maintaining quality.

4.5 Ryu Libraries

Ryu, a prominent SDN framework, offers a wide array of libraries and supports a variety of southbound protocols essential for network management and configuration. Among these protocols, Ryu extends its functionality to include the Open vSwitch Database Management Protocol (OVSDB), OF-Config, NETCONF, as well as Sflow and Netflow or network traffic analysis through packet sampling and aggregation techniques. Additionally, Ryu incorporates support for several third-party protocols, enhancing its versatility and application scope.

Key third-party libraries integrated into Ryu include the Open vSwitch Python binding, which facilitates interaction with Open vSwitch, the Oslo configuration library for managing configuration files, and a Python library designed for NETCONF clients, enabling efficient network configuration. Moreover, Ryu's packet library stands out as a particularly powerful tool, allowing network developers to dissect, analyse, and construct packets for a range of protocols such as VLAN and MPLS. This comprehensive support for diverse protocols and libraries empowers network developers to effectively manage and tailor network behaviour to meet specific requirements.

4.6 OpenFlow Protocol and Controller

Within the Ryu framework, an integral component is its built-in controller that interfaces with the OpenFlow (OF) protocol, a key southbound protocol Ryu supports. The framework's compatibility spans from the initial OpenFlow version 1.0 up to the more recent version 1.4, showcasing Ryu's commitment to staying current with OpenFlow advancements. A concise overview of the OpenFlow protocol messages alongside the

Table 4: OpenFLow Protocol messages and corresponding API of RYU

Controller to switch message	Asynchronous message	Symmetric message	Structures
• Handshake • switch-config • flow-table-config modify/read state • queue-config • packet-out, barrier • role-request	• Packet-in • flow-removed port-status • Error.	• Hello • Echo-Request & Reply • Error experimenter	• Flow-match
• send_msg API and packet builder APIs	• set_ev_cls API and packet parser APIs	• Both Send and Event APIs	

corresponding APIs offered by the Ryu controller is provided in Table 7, illustrating the framework's extensive support capabilities.

The architecture of Ryu positions the OpenFlow controller as a central element that acts as an internal source of events, capable of orchestrating switch management and handling various network events efficiently. Moreover, Ryu is equipped with a specialized library for encoding and decoding OpenFlow protocol messages. This library not only simplifies the interaction between the controller and network devices but also enhances the Ryu framework's ability to manage complex network configurations and operations seamlessly.

To launch a custom network topology in Mininet utilizing a specific topology file, the following command structure can be used. This example demonstrates how to initiate Mininet with a topology defined in a file named **simple_topo.py**, where the topology is identified within the file as **mytopo**:

The command used to launch a custom network topology is this:

Sudo mn --custom /simple_topo.py --topo mytopo --controller =remote, ip=, port= 5589

Here's a breakdown of the command components:

--**custom simple_topo.py** specifies the path and name of the file containing the topology definition. This tells Mininet where to find the custom topology settings.

--**topo mytopo** indicates the name of the topology as defined within the **simple_topo.py** file. This is the identifier for Mininet to understand which topology layout to implement from the file.

By combining these options, we're instructing Mininet to create a network based on the custom topology defined in **simple_topo.py**, ensuring each host has a unique MAC address for clear network traffic routing and identification.

4.7 Traffic Generator and Measurement Tools

In our academic study, while absolute measurement precision isn't crucial, we've selected tools precise enough to demonstrate general trends and specific behaviours under different Quality of Service (QoS) settings. These tools are:

1. iPerf: Used to measure the maximum bandwidth that IP networks can achieve. It generates TCP and UDP traffic to assess throughput between network endpoints. (29)

2. Wireshark: A network protocol analyser that captures and lets users browse traffic on a network. It's essential for troubleshooting and understanding traffic flow. (30)

3. DITG (Distributed Internet Traffic Generator): Generates various traffic patterns, including TCP, UDP, and ICMP. Designed for network performance testing, it can simulate multiple flows, aiding in complex QoS analyses. (31)

These tools collectively enable detailed monitoring and analysis of network performance, helping researchers to study the effects of various QoS configurations on traffic patterns and network efficiency.

4.8 Distributed Internet Traffic Generator (DITG)

The Distributed Internet Traffic Generator (D-ITG) is crucial for generating network traffic in our experiments, simulating real-world conditions by accurately replicating network flow traces. Unlike alternatives like TCPreplay (32) and TCPopera (33), D-ITG supports multiple flows and easy modifications to flow properties, along with advanced network metric measurement and recording capabilities. It also offers an analytical model-based generation mode for realistic network workload simulations (31), although this feature wasn't used in our current tests. D-ITG's versatility and depth in traffic simulation and analysis make it the preferred choice, enabling both trace-based and complex model-based traffic pattern generation.

4.9 Implementation with Mininet

In the development of the testbed for this research, the Ryu framework is employed as the Software-Defined Networking (SDN) controller, functioning from a dedicated host computer to facilitate interaction with the emulated network topology through a TCP connection. This network topology, created using Mininet, is a key component of the experimental setup, designed to examine various networking theories and practices. To guarantee the reproducibility of the experimental environment—a critical factor for validating research findings and supporting subsequent studies—the versions of Ryu, Mininet, and the OpenFlow protocol utilized in this setup are meticulously recorded in Table 5.

Table 5: Testbed requirements

No	Name	Specification
1	Operating System	Ubuntu 16.04 LTS (64 bits)
2	Ryu controller [Ryu]	Version 4.30
3	Mininet Emulator [Mininet]	Version 2.2.1
4	OpenFlow Protocol [OpenFlow]	Version 1.3

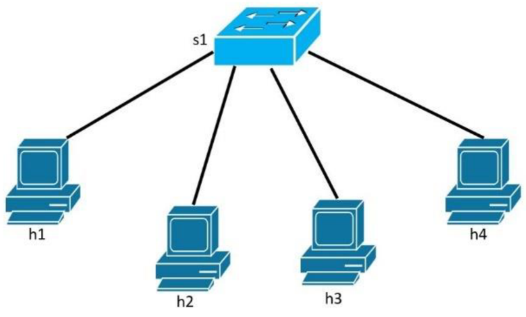


Figure 5: Simple Network Topology (own work)

The network topology for this testbed, as visualized in Figure 8, comprises a singular switch (s1) interconnected with four host computers (h1, h2, h3, h4), forming a basic yet versatile framework for conducting network experiments. This topology is instantiated through a Mininet script, detailing the configuration of the host nodes and the switch, along with the connections between them. Such a setup not only simplifies the network structure but also provides a foundational platform for analysing network behaviours, testing the effectiveness of QoS strategies, and exploring the dynamics of traffic management under the SDN paradigm. By executing this script within Mininet, the simulated network is brought to life, offering a controlled environment to undertake comprehensive assessments of network performance and the efficacy of SDN-controlled routing and traffic prioritization.

4.10 Experimental test

A. TCP traffic test (Simple Topology and l4 switch application)

TCP traffic test performed from host h1 to host h4 in a Software-Defined Networking (SDN) environment, using tools like iPerf and Open vSwitch (OVS) with OpenFlow13 protocols.

The primary goal is to generate TCP traffic from h1 (the sender) to h4 (the receiver) to measure the bandwidth from h1 to h4 in a network simulated using Mininet. In the given topology, there is a switch (s1) with four connected hosts (h1, h2, h3, h4).

The iPerf tool is used to test the network bandwidth between two points. Initially, iPerf in server mode is started on h4 using the command **`iperf -s`**. Subsequently, the iPerf client is initiated on h1 with the command **`iperf -c h4`**, establishing a TCP connection to the server on h4 and beginning the traffic flow for testing.

Bidirectional TCP traffic test

This test is conducted between host h1 and host h4 in a Software-Defined Network (SDN) setup, utilizing Mininet to emulate the network, iPerf for generating and measuring the traffic, and Open vSwitch (OVS) with OpenFlow for network management.

The objective of the TCP traffic test was to assess the handling of bidirectional traffic in an SDN environment using Mininet, iPerf, and OVS with OpenFlow. Simultaneously measuring traffic from host h1 to h4 and back, the test aimed to reflect real-world conditions. The setup involved a network topology with a switch and four hosts to initiate and analyze TCP flows, focusing on bandwidth and network management efficiency.

B. UDP traffic test

UDP traffic test between host h1 and host h4 within an emulated network environment using Mininet, with the Open vSwitch (OVS) and OpenFlow protocols for network control. The analysis will explain how UDP traffic testing is conducted and how it can measure network performance metrics such as bandwidth, latency, jitter, and packet loss.

Unlike TCP, UDP is a connectionless protocol and allows for flexibility in packet transmission, including control over packet sizes and transmission rates. In the context of network testing, UDP is used to analyze the network's capability to handle streaming media where packet loss can be tolerated to a certain extent, but bandwidth, latency, and jitter are critical.

The iPerf command **`iperf -u -c h4 -b 10m`** on h1 initiates a UDP traffic test to h4, specifying a bandwidth limit of 10 Mbps. The **`-u`** option indicates that the test is for UDP, **`-c h4`** specifies the connection to host h4, and **`-b 10m`** sets the target bandwidth to 10 Mbps.

C. VoIP traffic test

Voice over IP (VoIP) traffic test in a simulated network environment, specifically designed to evaluate the performance of UDP streaming, characteristic of VoIP calls. Using Mininet for network emulation, the iPerf tool for traffic generation and measurement, and Open vSwitch (OVS) with OpenFlow for network control, the testing examines both single and multiple VoIP calls.

Creating the Topology and Controller Setup:

1. Topology Initialization: The network topology is created with the command **`sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4`**. This establishes a network with one switch (**`ovsk`**) and four hosts (**`h1-h4`**), using OpenFlow protocol version 13 for SDN control.

2. RYU Controller: The RYU controller is started using **`ryu-manager l4_switch.py`**, which loads a custom Layer 4 switching application to manage traffic flows based on SDN policies.

VoIP Traffic Testing:

1. Single 64Kbps VoIP Call Test:

- Server Setup: The iPerf UDP server is initiated on host h4 with settings that simulate a VoIP call (**`--server --udp --len 300 --tos 184 -fk --interval 5`**). The TOS (Type of Service) field is set to 184 to prioritize the traffic as voice.

- Client Setup: On host h1, the iPerf UDP client is configured to connect to h4 (**`-c 10.1.1.4`**) with options (**`--udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60 --listenport 5002`**) reflecting a single VoIP call for 60 seconds.

2. Multiple Parallel VoIP Calls Test:

- Server Setup for Parallel Calls: The iPerf server on h4 is set to handle multiple streams simultaneously (`--parallel 4`), indicating multiple VoIP calls.

- Client Setup for Parallel Calls: Host h1 runs a similar iPerf command as the single call test but designed to establish multiple streams in parallel.

D. QoS using SDN (end-to-end bandwidth guarantee)

Quality of Service (QoS) is achieved in SDN through the implementation of OVS (OpenVSwitch) Queues. This method is aimed at ensuring Quality of Service for TCP, UDP, and VoIP traffic. The goal is to allocate dedicated bandwidth for each type of traffic as follows:

I. TCP = 3 Mbps
II. UDP = 1 Mbps
III. VoIP = 1 Mbps

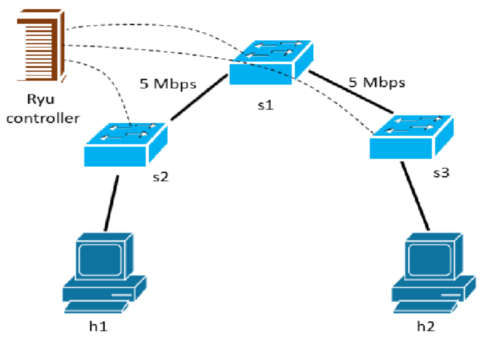


Figure 6 Simple topology for QoS testing.

In this setup, a simple tree topology is utilized, where the bandwidth of the links connecting the switches is specified to be 5 Mbps.

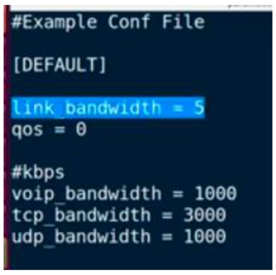
4.10.1 Test 1: Without activating the QoS feature on the controller.

Initially, the focus is on observing the network behavior when traffic is generated from h1 to h2 without enabling the Quality of Service (QoS) feature on the controller. The experiment involves generating a specified amount of traffic from h1 directed towards h2.

I. TCP = auto
II. UDP = 6 Mbps
III. VoIP = 670 Kbps (10 calls, 67 Kbps per call)

Problem Statement: Given the bandwidth limitations between the switches, a scenario arises where some traffic must be discarded due to congestion. As a result, it is expected to observe the dropping of traffic, which could include TCP, UDP, or VoIP types. Consequently, congestion within the switch ring leads to traffic being discarded.

For a demonstration of the issue, the Mininet topology is initiated without enabling the Quality of Service (QoS) functionality on the controller. The controller application, developed using the Ryu app **`simple_switch_13`**, is started with QoS functionality turned off. This is achieved by setting **`QoS = 0`** in the parameter file, named **`param.conf`**.

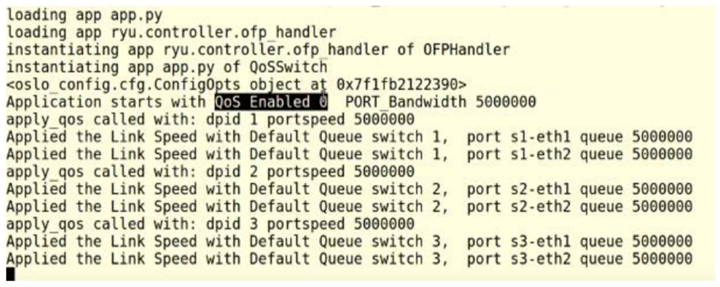


```
#Example Conf File

[DEFAULT]

link_bandwidth = 5
qos = 0

#kbps
voip_bandwidth = 1000
tcp_bandwidth = 3000
udp_bandwidth = 1000
```

Figure 7: Parameter configuration file

Within the topology file, it is configured to automatically initiate the traffic test upon starting the topology. This setup ensures that TCP, UDP, and VoIP traffic generation commences for a duration of 60 seconds immediately after the topology is launched. Once the controller starts, it will show the configured bandwidth between switches and if the QoS function is enabled or not.

With TCP traffic generation set to automatic, it will attempt to push as much traffic as possible through the network. Additionally, UDP traffic is specified to push 6 Mbps, and similarly, VoIP traffic is configured to attempt 670 Kbps, both striving to reach their specified bandwidth limits.

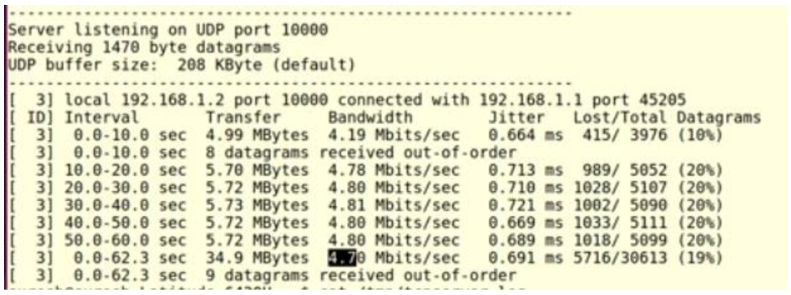


```
loading app app.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app app.py of QoSSwitch
<oslo_config.cfg.ConfigOpts object at 0x7f1fb2122390>
Application starts with QoS Enabled 0  PORT_Bandwidth 5000000
apply_qos called with: dpid 1 portspeed 5000000
Applied the Link Speed with Default Queue switch 1,  port s1-eth1 queue 5000000
Applied the Link Speed with Default Queue switch 1,  port s1-eth2 queue 5000000
apply_qos called with: dpid 2 portspeed 5000000
Applied the Link Speed with Default Queue switch 2,  port s2-eth1 queue 5000000
Applied the Link Speed with Default Queue switch 2,  port s2-eth2 queue 5000000
apply_qos called with: dpid 3 portspeed 5000000
Applied the Link Speed with Default Queue switch 3,  port s3-eth1 queue 5000000
Applied the Link Speed with Default Queue switch 3,  port s3-eth2 queue 5000000
```

Figure 8: Controller showing QoS status

After the test concludes, examining the server logs will reveal the amount of traffic that successfully reached the other host.

First, the focus is on reviewing the server log for UDP traffic to assess how much of it was successfully transmitted to the other host.

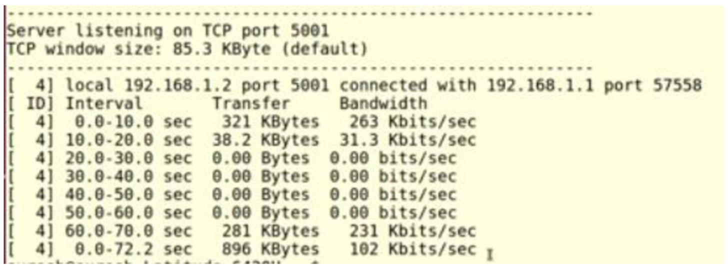


```
------------------------------------------------------------
Server listening on UDP port 10000
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.2 port 10000 connected with 192.168.1.1 port 45205
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  4.99 MBytes  4.19 Mbits/sec   0.664 ms  415/ 3976 (10%)
[  3]  0.0-10.0 sec  8 datagrams received out-of-order
[  3] 10.0-20.0 sec  5.70 MBytes  4.78 Mbits/sec   0.713 ms  989/ 5052 (20%)
[  3] 20.0-30.0 sec  5.72 MBytes  4.80 Mbits/sec   0.710 ms 1028/ 5107 (20%)
[  3] 30.0-40.0 sec  5.73 MBytes  4.81 Mbits/sec   0.721 ms 1002/ 5090 (20%)
[  3] 40.0-50.0 sec  5.72 MBytes  4.80 Mbits/sec   0.669 ms 1033/ 5111 (20%)
[  3] 50.0-60.0 sec  5.72 MBytes  4.80 Mbits/sec   0.689 ms 1018/ 5099 (20%)
[  3]  0.0-62.3 sec  34.9 MBytes  4.70 Mbits/sec   0.691 ms 5716/30613 (19%)
[  3]  0.0-62.3 sec  9 datagrams received out-of-order
```

Figure 9: UDP server log showing amount of data transferred without QoS

Despite attempting to transfer 5.70 Mbps of UDP traffic across a link with a bandwidth limit of 5 Mbps, only 4.70 Mbps of the traffic successfully reached its destination, indicating that the UDP traffic occupies the majority of the available bandwidth.

Next, attention is turned to examining the server log for TCP traffic to understand how much of it managed to traverse to the other host.

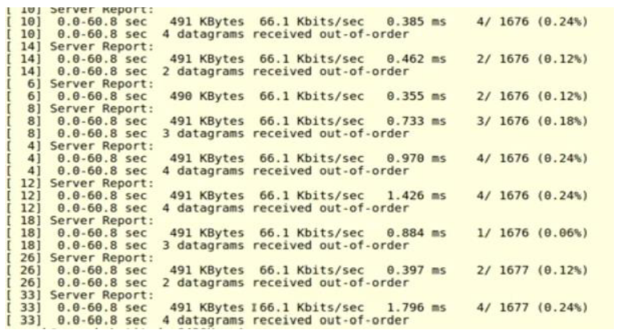


```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.2 port 5001 connected with 192.168.1.1 port 57558
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   321 KBytes   263 Kbits/sec
[  4] 10.0-20.0 sec  38.2 KBytes  31.3 Kbits/sec
[  4] 20.0-30.0 sec  0.00 Bytes  0.00 bits/sec
[  4] 30.0-40.0 sec  0.00 Bytes  0.00 bits/sec
[  4] 40.0-50.0 sec  0.00 Bytes  0.00 bits/sec
[  4] 50.0-60.0 sec  0.00 Bytes  0.00 bits/sec
[  4] 60.0-70.0 sec   281 KBytes   231 Kbits/sec
[  4]  0.0-72.2 sec   896 KBytes   102 Kbits/sec
```

Figure 10: TCP traffic log without QoS

Given that UDP traffic has already consumed the majority of the available bandwidth on the link, TCP traffic was restricted to achieving only 102 Kbits/sec.

Following is the server log from VoIP.



```
[ 10] Server Report:
[ 10]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.385 ms    4/ 1676 (0.24%)
[ 10]  0.0-60.8 sec  4 datagrams received out-of-order
[ 14] Server Report:
[ 14]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.462 ms    2/ 1676 (0.12%)
[ 14]  0.0-60.8 sec  2 datagrams received out-of-order
[  6] Server Report:
[  6]  0.0-60.8 sec   490 KBytes  66.1 Kbits/sec   0.355 ms    2/ 1676 (0.12%)
[  8] Server Report:
[  8]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.733 ms    3/ 1676 (0.18%)
[  8]  0.0-60.8 sec  3 datagrams received out-of-order
[  4] Server Report:
[  4]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.970 ms    4/ 1676 (0.24%)
[  4]  0.0-60.8 sec  4 datagrams received out-of-order
[ 12] Server Report:
[ 12]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   1.426 ms    4/ 1676 (0.24%)
[ 12]  0.0-60.8 sec  4 datagrams received out-of-order
[ 18] Server Report:
[ 18]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.884 ms    1/ 1676 (0.06%)
[ 18]  0.0-60.8 sec  3 datagrams received out-of-order
[ 26] Server Report:
[ 26]  0.0-60.8 sec   491 KBytes  66.1 Kbits/sec   0.397 ms    2/ 1677 (0.12%)
[ 26]  0.0-60.8 sec  2 datagrams received out-of-order
[ 33] Server Report:
[ 33]  0.0-60.8 sec   491 KBytes I66.1 Kbits/sec   1.796 ms    4/ 1677 (0.24%)
[ 33]  0.0-60.8 sec  4 datagrams received out-of-order
```

Figure 11: VoIP server log without QoS

VoIP traffic managed to secure only 66.1 Kbits/sec out of the 5 Mbps link bandwidth, underscoring the absence of quality-of-service (QoS) mechanisms in the network.

In the subsequent test, the intention is to implement QoS, wherein bandwidth will be specifically allocated for each type of service to ensure more equitable distribution of network resources.

4.10.2 Test 2: With QoS feature on the controller.

For the implementation of Quality of Service (QoS), bandwidth allocation for each service type is specified as follows:

I. TCP = 3 Mbps
II. UDP = 1 Mbps
III. VoIP = 1 Mbps

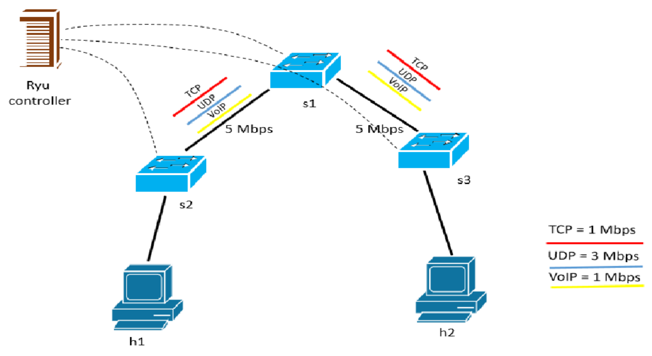
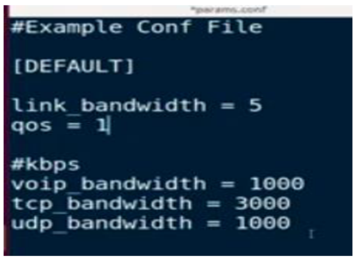


Figure 12: QoS setup on the same topology

This specific bandwidth per service has been taken as a random number by considering the link bandwidth of 5 Mbps between the switches. The total bandwidth of the various services matches the link bandwidth which will be measured after the test if the specific services has achieved this assigned bandwidth or not. This number can be varied upon the requirements and the available minimum link bandwidth of the network infrastructure.

The controller is programmed with specific logic to implement Quality of Service (QoS) effectively:

1. OVS queues are utilized to manage QoS, allowing for differentiated handling of traffic types.
2. Queues are established for each type of traffic, with specific bandwidth allocations defined for each queue to ensure that each service type is accorded the necessary resources.
3. Traffic classification mechanisms are put in place to identify incoming traffic types. This enables the system to direct each traffic type to its designated queue based on the classification, ensuring that traffic is managed according to the classification.

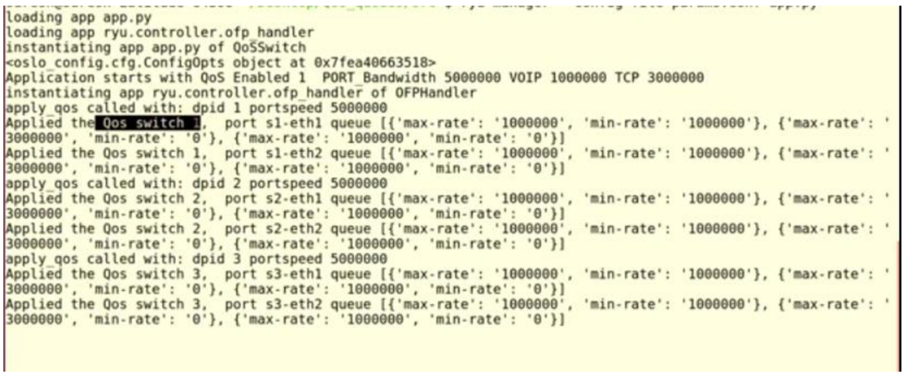


```
#Example Conf File

[DEFAULT]

link_bandwidth = 5
qos = 1

#kbps
voip_bandwidth = 1000
tcp_bandwidth = 3000
udp_bandwidth = 1000
```

Figure 13: Configuration file to enable QoS with bandwidth set for each queue

Upon activating the Quality of Service (QoS) features, the same test will be repeated to observe the impact of QoS on network performance. The parameter file is updated to include the specified bandwidth allocations for each traffic type, ensuring that the network adheres to these settings during the test to effectively manage and prioritize traffic according to the newly implemented QoS policies.

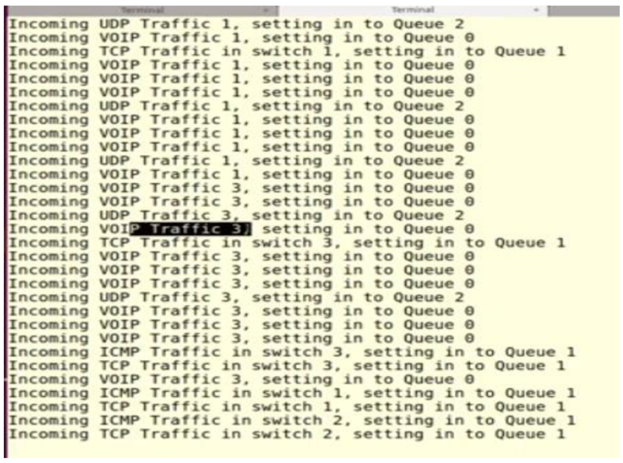
As the controller initiates, it displays the settings of the Queues, including the configured bandwidth allocations for each. Additionally, it indicates whether the QoS feature is activated, providing clear visibility into the operational status of QoS mechanisms within the network. This information is crucial for verifying that the network is prepared to manage traffic according to the predefined QoS parameters.



```
loading app app.py
loading app ryu.controller.ofp_handler
instantiating app app.py of QoSSwitch
<oslo_config.cfg.ConfigOpts object at 0x7fea40663518>
Application starts with QoS Enabled 1  PORT_Bandwidth 5000000 VOIP 1000000 TCP 3000000
instantiating app ryu.controller.ofp_handler of OFPHandler
apply_qos called with: dpid 1 portspeed 5000000
Applied the Qos switch 1,  port s1-eth1 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
Applied the Qos switch 1,  port s1-eth2 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
apply_qos called with: dpid 2 portspeed 5000000
Applied the Qos switch 2,  port s2-eth1 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
Applied the Qos switch 2,  port s2-eth2 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
apply_qos called with: dpid 3 portspeed 5000000
Applied the Qos switch 3,  port s3-eth1 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
Applied the Qos switch 3,  port s3-eth2 queue [{'max-rate': '1000000', 'min-rate': '1000000'}, {'max-rate': '
3000000', 'min-rate': '0'}, {'max-rate': '1000000', 'min-rate': '0'}]
```

Figure 14: Controller showing queue settings applied on switch ports

In the following output table, it is observed that the controller is effectively distinguishing between various types of traffic and allocating them to the appropriate Queue designated for each traffic type. This demonstrates the controller's capability to classify incoming traffic and manage it according to the established QoS policies, ensuring that each traffic type is processed within its allocated bandwidth constraints for optimal network performance.



```
Incoming UDP Traffic 1, setting in to Queue 2
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming TCP Traffic in switch 1, setting in to Queue 1
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming UDP Traffic 1, setting in to Queue 2
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming UDP Traffic 1, setting in to Queue 2
Incoming VOIP Traffic 1, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming UDP Traffic 3, setting in to Queue 2
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming TCP Traffic in switch 3, setting in to Queue 1
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming UDP Traffic 3, setting in to Queue 2
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming ICMP Traffic in switch 3, setting in to Queue 1
Incoming TCP Traffic in switch 3, setting in to Queue 1
Incoming VOIP Traffic 3, setting in to Queue 0
Incoming ICMP Traffic in switch 1, setting in to Queue 1
Incoming TCP Traffic in switch 1, setting in to Queue 1
Incoming ICMP Traffic in switch 2, setting in to Queue 1
Incoming TCP Traffic in switch 2, setting in to Queue 1
```

Figure 15: Controller output showing 3 different queue setup for incoming traffic

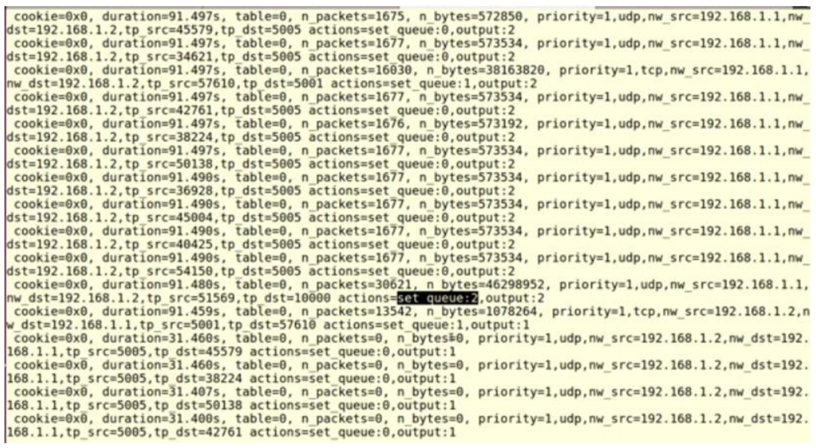
The controller establishes three distinct Queues to manage the diverse traffic types within the network, adhering to the predefined Quality of Service (QoS) policies. Specifically:

I. VoIP traffic = Queue 0
II. TCP traffic = Queue 1
III. UDP traffic = Queue 2

The controller functions to identify incoming traffic, classify it according to its type, and subsequently place it in the designated Queue. This process ensures that each traffic type is managed and prioritized appropriately, in line with established Quality of Service (QoS) guidelines.

The actions undertaken by the switch can be verified by inspecting the flow entries within the switch, known as dump flows. To review the dump flows of switch s2, the following command is executed: **`sudo ovs-ofctl dump-flows s2`**. This command provides a detailed overview of the flow entries managed by the switch, illustrating how traffic is being classified, routed, and prioritized according to the QoS policies implemented by the controller.

sudo ovs-ofctl -O OpenFlow13 dump-flows s2



```
 cookie=0x0, duration=91.497s, table=0, n_packets=1675, n_bytes=572850, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=45579,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.497s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=34621,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.497s, table=0, n_packets=16030, n_bytes=38163820, priority=1,tcp,nw_src=192.168.1.1,
nw_dst=192.168.1.2,tp_src=57610,tp_dst=5001 actions=set_queue:1,output:2
 cookie=0x0, duration=91.497s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=42761,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.497s, table=0, n_packets=1676, n_bytes=573192, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=38224,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.497s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=50138,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.490s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=36928,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.490s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=45004,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.490s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=40425,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.490s, table=0, n_packets=1677, n_bytes=573534, priority=1,udp,nw_src=192.168.1.1,nw_
dst=192.168.1.2,tp_src=54150,tp_dst=5005 actions=set_queue:0,output:2
 cookie=0x0, duration=91.480s, table=0, n_packets=30621, n_bytes=46298952, priority=1,udp,nw_src=192.168.1.1,
nw_dst=192.168.1.2,tp_src=51569,tp_dst=10000 actions=set_queue:2,output:2
 cookie=0x0, duration=91.459s, table=0, n_packets=13542, n_bytes=1078264, priority=1,tcp,nw_src=192.168.1.2,n
w_dst=192.168.1.1,tp_src=5001,tp_dst=57610 actions=set_queue:1,output:1
 cookie=0x0, duration=31.460s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=192.168.1.2,nw_dst=192.
168.1.1,tp_src=5005,tp_dst=45579 actions=set_queue:0,output:1
 cookie=0x0, duration=31.460s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=192.168.1.2,nw_dst=192.
168.1.1,tp_src=5005,tp_dst=38224 actions=set_queue:0,output:1
 cookie=0x0, duration=31.407s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=192.168.1.2,nw_dst=192.
168.1.1,tp_src=5005,tp_dst=50138 actions=set_queue:0,output:1
 cookie=0x0, duration=31.400s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=192.168.1.2,nw_dst=192.
168.1.1,tp_src=5005,tp_dst=42761 actions=set_queue:0,output:1
```

Figure 16: Switch dump-flows output showing action set for queue

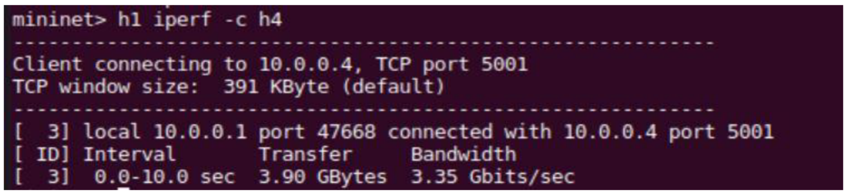
In the flow entries for switch s2, an additional field specifies actions, such as `actions=set queue:2`, which is contingent upon the type of incoming traffic. In the provided example, where the traffic is TCP, the flow entry indicates that the action taken is to direct this traffic to queue 2. This illustrates the switch's adherence to the controller's instructions for classifying and queuing traffic based on its type, ensuring that each traffic flow is handled according to the designated QoS policy.

After the completion of the 60-second test, the server logs are examined to verify the effectiveness of the Quality of Service (QoS) mechanisms. These logs provide essential insights into whether the traffic was appropriately classified and queued, and if the predefined bandwidth allocations for each type of traffic were adhered to, thereby determining the QoS's operational success.

5. Result and Discussion

TCP traffic test (Simple Topology and l4 switch application)

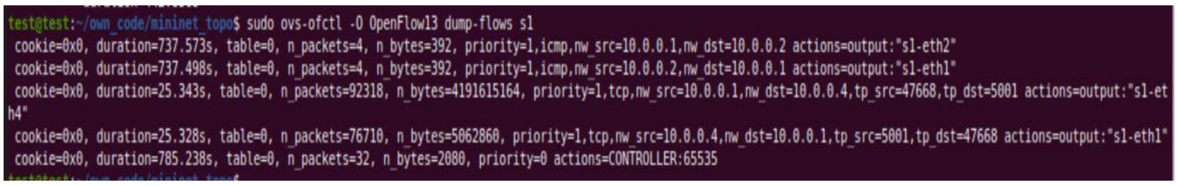
The observation of traffic through iPerf's output highlights a successful connection between h1 and h4, showcasing a data transfer rate at the level of gigabits per second (Gbps). This high transfer rate signifies the efficient bandwidth performance for data sent from h1 to h4. In contrast, the traffic returning from h4 to h1 presents a significantly lower rate, aligning with the expected behaviour of TCP acknowledgment packets. These acknowledgments, inherently smaller than the outbound data packets, illustrate the asymmetrical nature of TCP traffic flow within the network.



```
mininet> h1 iperf -c h4
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size:  391 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 47668 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  3.90 GBytes  3.35 Gbits/sec
```

Figure 17: Simple TCP traffic test

The flow analysis conducted with OVS using the ***`sudo ovs-ofctl -O OpenFlow13 dump-flows s1`*** command, the flow entries in the OpenFlow switch (s1) are inspected. This reveals the details of each flow, such as source and destination IP addresses, transport layer ports, and associated actions (like output port for forwarding). These details are crucial for understanding how the SDN controller is managing the network flows.



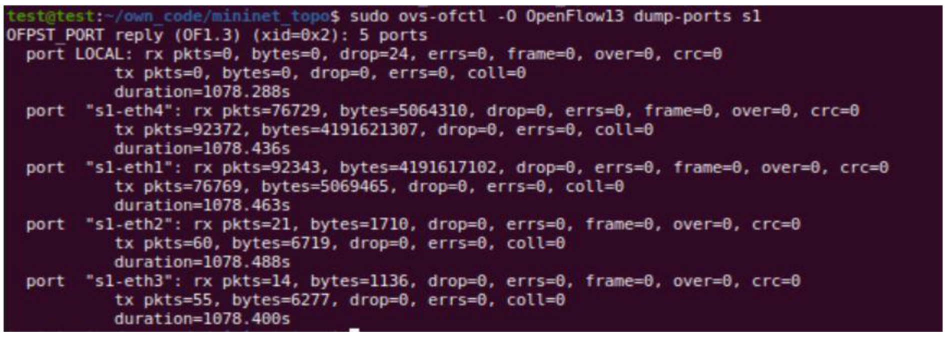
```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=737.573s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=737.498s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=25.343s, table=0, n_packets=92318, n_bytes=4191615164, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47668,tp_dst=5001 actions=output:"s1-et
h4"
 cookie=0x0, duration=25.328s, table=0, n_packets=76710, n_bytes=5062860, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47668 actions=output:"s1-eth1"
 cookie=0x0, duration=785.238s, table=0, n_packets=32, n_bytes=2080, priority=0 actions=CONTROLLER:65535
```

Figure 18: TCP traffic dump-flows

Port Analysis within the Open vSwitch (OVS) framework is facilitated by the ***`sudo ovs-ofctl -O OpenFlow13 dump-ports s1`*** provides statistics about the switch ports. The analysis by ports shows that:

- For forward traffic, h1 transmits and port1 of the switch receives this traffic. It is then transmitted out of port4 to h4.

- For acknowledgment traffic, h4 is the transmitter, and the acknowledgment packets are received by port4, then transmitted out of port1 back to h1.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=24, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=1078.288s
  port  "s1-eth4": rx pkts=76729, bytes=5064310, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=92372, bytes=4191621307, drop=0, errs=0, coll=0
           duration=1078.436s
  port  "s1-eth1": rx pkts=92343, bytes=4191617102, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=76769, bytes=5069465, drop=0, errs=0, coll=0
           duration=1078.463s
  port  "s1-eth2": rx pkts=21, bytes=1710, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=60, bytes=6719, drop=0, errs=0, coll=0
           duration=1078.488s
  port  "s1-eth3": rx pkts=14, bytes=1136, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=55, bytes=6277, drop=0, errs=0, coll=0
           duration=1078.400s
```

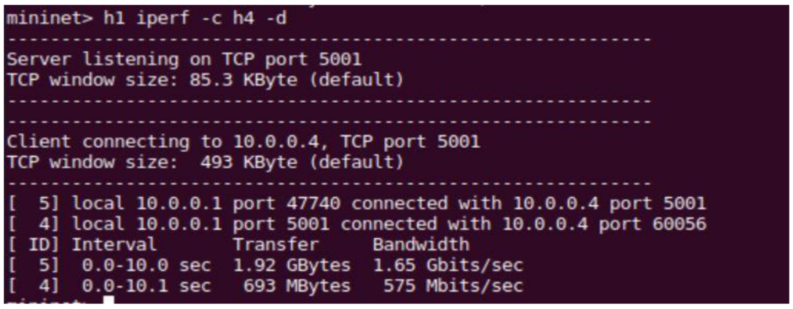
Figure 19: TCP traffic test - dump-ports

The observed high throughput rate from h1 to h4 suggests that the network is capable of handling high-bandwidth applications. The efficient routing of acknowledgments also indicates that the controller and switch configuration can effectively manage two-way communication.

The findings from the TCP traffic test between h1 and h4 shed light on the asymmetric nature of TCP traffic, where there's a notable imbalance between the high volume of data transfer in one direction and the minimal volume of acknowledgments in the opposite direction. Recognizing this pattern is vital for effective network planning, ensuring that the infrastructure is robust enough to support peak demands without succumbing to congestion. The successful management of high bandwidth demands and the strategic flow routing underscore the efficacy of the SDN controller in orchestrating network traffic. The application of iPerf and OVS commands throughout the test offers comprehensive insights into network performance and the Quality of Service (QoS) achievable. This information proves invaluable for network administrators and researchers aiming to fine-tune network configurations to meet the performance needs of diverse applications, highlighting the crucial role of detailed network behavior understanding in optimizing network capacity and functionality.

Bidirectional TCP traffic test

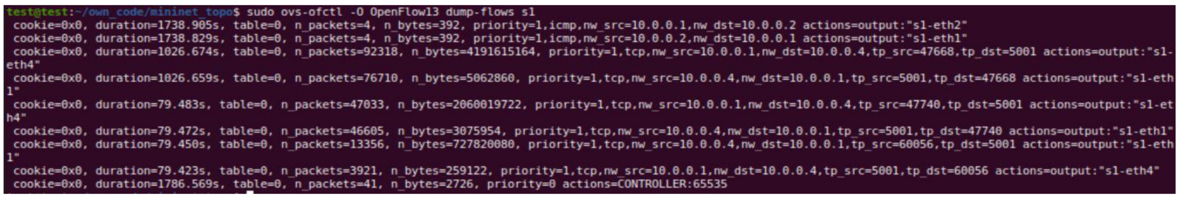
The iPerf test output presents data for two intervals, revealing the nature of traffic flow between h1 and h4, alongside the acknowledgments sent in the opposite direction. This demonstrates a significant transfer rate from h1 to h4, with 1.92 GBytes of data transmitted at a velocity of 1.65 Gbits/sec. Conversely, the return path from h4 to h1 sees a diminished rate, with 693 MBytes transferred at 575 Mbits/sec. Such asymmetry is emblematic of TCP traffic, where the bulk of data moves predominantly in one direction, while the acknowledgments, being considerably smaller, constitute a lesser volume of traffic in the return direction.



```
mininet> h1 iperf -c h4 -d
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size:  493 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.1 port 47740 connected with 10.0.0.4 port 5001
[  4] local 10.0.0.1 port 5001 connected with 10.0.0.4 port 60056
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.92 GBytes  1.65 Gbits/sec
[  4]  0.0-10.1 sec   693 MBytes   575 Mbits/sec
```

Figure 20: Bi-directional TCP test

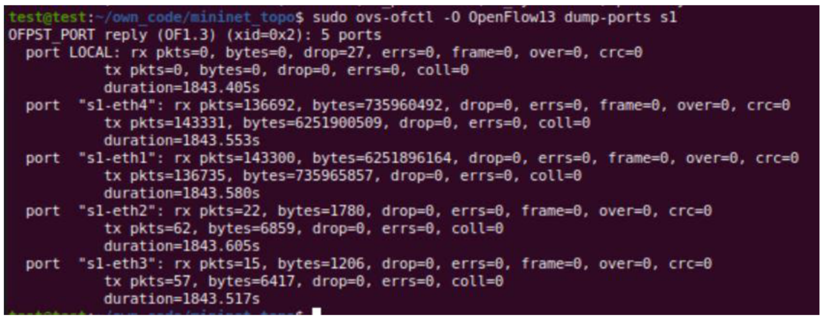
The flow entries within the OVS switch (s1) are conducted through the command ***`sudo ovs-ofctl -O OpenFlow13 dump-flows s1`***, we examine the flow entries in the OVS switch **(s1)**. These entries contain match conditions (such as IP addresses and TCP ports) and actions (such as the output port) for each flow. The dump shows multiple flows with source and destination IP addresses corresponding to h1 and h4, and transport layer ports that are involved in the TCP communication.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=1738.905s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=1738.829s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=1026.674s, table=0, n_packets=92318, n_bytes=4191615164, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47668,tp_dst=5001 actions=output:"s1-
eth4"
 cookie=0x0, duration=1026.659s, table=0, n_packets=76710, n_bytes=5062860, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47668 actions=output:"s1-eth
1"
 cookie=0x0, duration=79.483s, table=0, n_packets=47033, n_bytes=2060019722, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47740,tp_dst=5001 actions=output:"s1-et
h4"
 cookie=0x0, duration=79.472s, table=0, n_packets=46605, n_bytes=3075954, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47740 actions=output:"s1-eth1"
 cookie=0x0, duration=79.450s, table=0, n_packets=13356, n_bytes=727820080, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=60056,tp_dst=5001 actions=output:"s1-eth
1"
 cookie=0x0, duration=79.423s, table=0, n_packets=3921, n_bytes=259122, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=5001,tp_dst=60056 actions=output:"s1-eth4"
 cookie=0x0, duration=1786.569s, table=0, n_packets=41, n_bytes=2726, priority=0 actions=CONTROLLER:65535
```

Figure 21: Bi-directional TCP test-dump-flows

Port statistics on the switch s1 are scrutinized using the command ***`sudo ovs-ofctl -O OpenFlow13 dump-ports s1`*** is used to inspect the port statistics on the switch s1. The output indicates the number of packets and bytes received (rx) and transmitted (tx) on each port. For instance, Notably, the statistics reveal the data traffic transmitted and received by port1 and port4, which correspond to h1 and h4, respectively. These statistics are crucial for identifying how the network load is distributed across the switch's ports.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=27, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=1843.405s
  port  "s1-eth4": rx pkts=136692, bytes=735960492, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=143331, bytes=6251900509, drop=0, errs=0, coll=0
           duration=1843.553s
  port  "s1-eth1": rx pkts=143300, bytes=6251896164, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=136735, bytes=735965857, drop=0, errs=0, coll=0
           duration=1843.580s
  port  "s1-eth2": rx pkts=22, bytes=1780, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=62, bytes=6859, drop=0, errs=0, coll=0
           duration=1843.605s
  port  "s1-eth3": rx pkts=15, bytes=1206, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=57, bytes=6417, drop=0, errs=0, coll=0
           duration=1843.517s
```

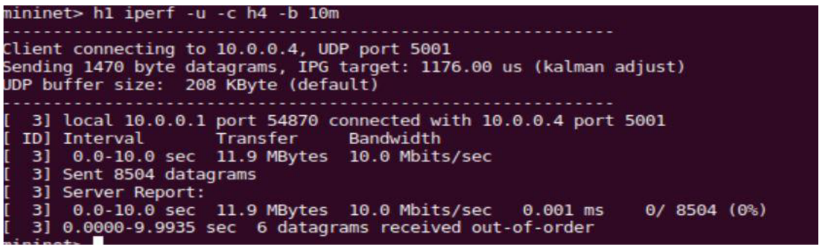
Figure 22: Bi-directional TCP test-dump-ports

The results from the iPerf and OVS commands give a clear picture of the network's capability to handle simultaneous bidirectional TCP traffic. The data transmission rates indicate how efficiently the network can handle the load, and the OVS statistics can help in identifying any bottlenecks or performance issues at the switch level.

This bidirectional test provides a holistic view of the network's performance, showing how the SDN controller and switch manage traffic in both directions. The information gleaned from this test can be used to optimize the network's configuration for better handling of simultaneous data flows, which is critical in environments where multiple services rely on the network's transport capabilities.

UDP traffic test

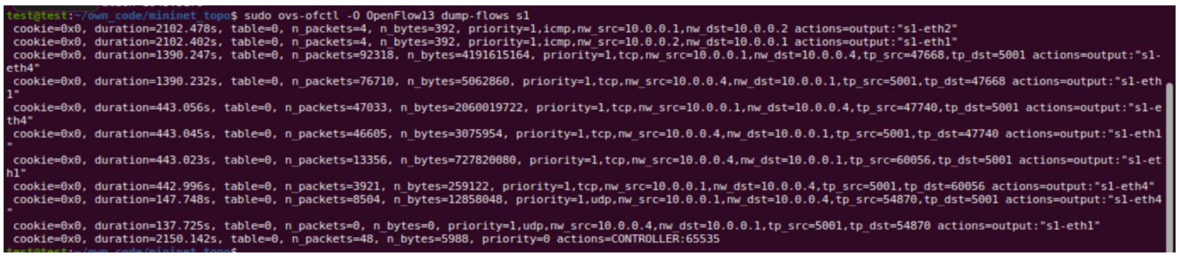
The iPerf test results show that the sender (h1) has sent data to the receiver (h4) at the specified bandwidth, transferring a total of 11.9 MBytes of data at a bandwidth of 10.0 Mbits/sec. This outcome verifies the network's throughput under controlled conditions, where 8504 datagrams have been sent with no reported packet loss or out-of-order delivery, which is critical for UDP performance analysis.



```
mininet> h1 iperf -u -c h4 -b 10m
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 54870 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] Sent 8504 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.001 ms    0/ 8504 (0%)
[  3] 0.0000-9.9935 sec  6 datagrams received out-of-order
```

Figure 23: UDP traffic test

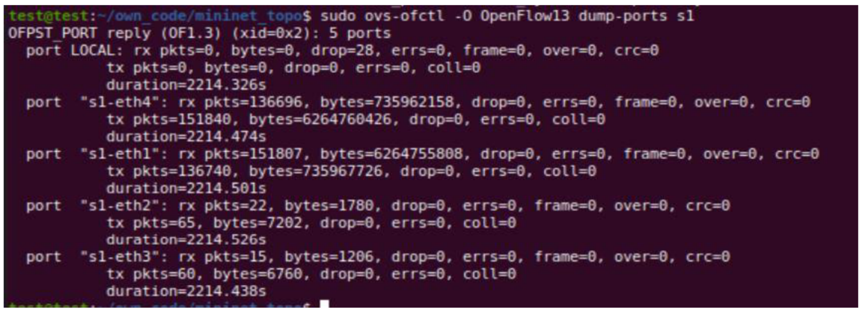
The inspection of flow entries within the switch, conducted through the ***`sudo ovs-ofctl -O OpenFlow13 dump-flows s1`***, the flow entries in the switch are examined to observe how the SDN controller has managed the UDP traffic. The output reveals match conditions that identify the UDP traffic between h1 and h4, including the source and destination IP addresses and ports, with actions that direct the traffic to the appropriate output ports of the switch.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=2102.478s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=2102.402s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=1390.247s, table=0, n_packets=92318, n_bytes=4191615164, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47668,tp_dst=5001 actions=output:"s1-
eth4"
 cookie=0x0, duration=1390.232s, table=0, n_packets=76710, n_bytes=5062860, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47668 actions=output:"s1-eth
1"
 cookie=0x0, duration=443.056s, table=0, n_packets=47033, n_bytes=2060019722, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47740,tp_dst=5001 actions=output:"s1-e
th4"
 cookie=0x0, duration=443.045s, table=0, n_packets=46605, n_bytes=3075954, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47740 actions=output:"s1-eth1
"
 cookie=0x0, duration=443.023s, table=0, n_packets=13356, n_bytes=727820080, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=60056,tp_dst=5001 actions=output:"s1-et
h1"
 cookie=0x0, duration=442.996s, table=0, n_packets=3921, n_bytes=259122, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=5001,tp_dst=60056 actions=output:"s1-eth4"
 cookie=0x0, duration=147.748s, table=0, n_packets=8504, n_bytes=12858048, priority=1,udp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=54870,tp_dst=5001 actions=output:"s1-eth4
"
 cookie=0x0, duration=137.725s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=54870 actions=output:"s1-eth1"
 cookie=0x0, duration=2150.142s, table=0, n_packets=48, n_bytes=5988, priority=0 actions=CONTROLLER:65535
test@test:~/own_code/mininet_topo$
```

Figure 24: UDP traffic test - dump-flows

The execution of ***'sudo ovs-ofctl -O OpenFlow13 dump-ports s1'*** offers a detailed analysis of port statistics within the switch, showing the number of packets and bytes sent and received on each port, which is useful for determining the effectiveness of the network's data handling capabilities and identifying any potential congestion or issues at the port level.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=28, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=2214.326s
  port  "s1-eth4": rx pkts=136696, bytes=735962158, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=151840, bytes=6264760426, drop=0, errs=0, coll=0
           duration=2214.474s
  port  "s1-eth1": rx pkts=151807, bytes=6264755808, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=136740, bytes=735967726, drop=0, errs=0, coll=0
           duration=2214.501s
  port  "s1-eth2": rx pkts=22, bytes=1780, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=65, bytes=7202, drop=0, errs=0, coll=0
           duration=2214.526s
  port  "s1-eth3": rx pkts=15, bytes=1206, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=60, bytes=6760, drop=0, errs=0, coll=0
           duration=2214.438s
```

Figure 25: UDP traffic test-dump-ports

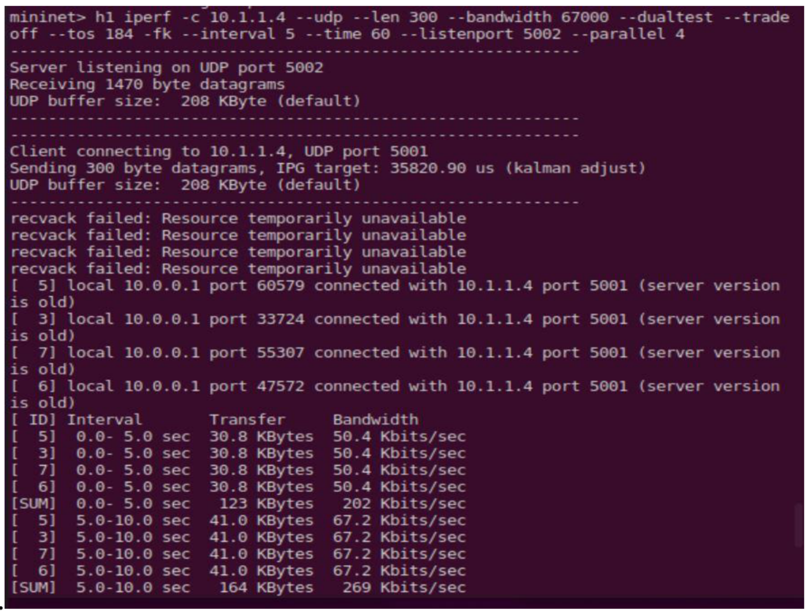
The combination of iPerf and OVS tools allows for an in-depth analysis of the network's performance. By observing metrics such as jitter (variation in latency), packet loss, and the consistency of bandwidth, network administrators can evaluate the Quality of Service (QoS) provided by the network and adjust configurations to optimize performance for UDP traffic.

This UDP traffic test provides an evaluation of the network's ability to handle datagrams efficiently and effectively, offering insights for network tuning to improve streaming quality. The findings can be leveraged to fine-tune the network's performance to ensure high-quality service delivery for real-time applications that depend on UDP transmission.

VoIP traffic test

iPerf UDP test outputs show the iPerf client (h1) connecting to the iPerf server (h4).

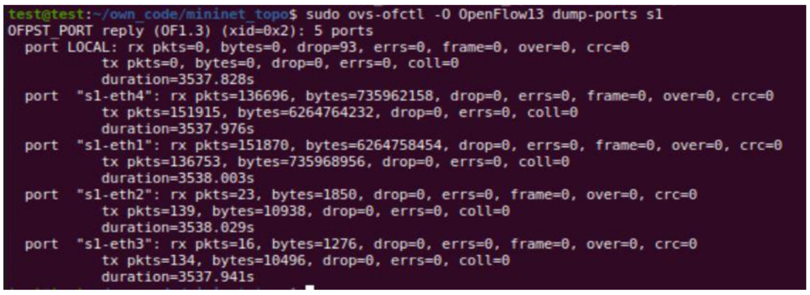
The bandwidth achieved is consistent with the specified target for a VoIP call (64 Kbps), and the performance metrics are recorded for intervals of 5 seconds over a 60-second test duration.



```
mininet> h1 iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 --dualtest --trade
off --tos 184 -fk --interval 5 --time 60 --listenport 5002 --parallel 4
------------------------------------------------------------
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.1.1.4, UDP port 5001
Sending 300 byte datagrams, IPG target: 35820.90 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
recvack failed: Resource temporarily unavailable
recvack failed: Resource temporarily unavailable
recvack failed: Resource temporarily unavailable
recvack failed: Resource temporarily unavailable
[  5] local 10.0.0.1 port 60579 connected with 10.1.1.4 port 5001 (server version
is old)
[  3] local 10.0.0.1 port 33724 connected with 10.1.1.4 port 5001 (server version
is old)
[  7] local 10.0.0.1 port 55307 connected with 10.1.1.4 port 5001 (server version
is old)
[  6] local 10.0.0.1 port 47572 connected with 10.1.1.4 port 5001 (server version
is old)
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 5.0 sec  30.8 KBytes  50.4 Kbits/sec
[  3]  0.0- 5.0 sec  30.8 KBytes  50.4 Kbits/sec
[  7]  0.0- 5.0 sec  30.8 KBytes  50.4 Kbits/sec
[  6]  0.0- 5.0 sec  30.8 KBytes  50.4 Kbits/sec
[SUM]  0.0- 5.0 sec   123 KBytes   202 Kbits/sec
[  5]  5.0-10.0 sec  41.0 KBytes  67.2 Kbits/sec
[  3]  5.0-10.0 sec  41.0 KBytes  67.2 Kbits/sec
[  7]  5.0-10.0 sec  41.0 KBytes  67.2 Kbits/sec
[  6]  5.0-10.0 sec  41.0 KBytes  67.2 Kbits/sec
[SUM]  5.0-10.0 sec   164 KBytes   269 Kbits/sec
```

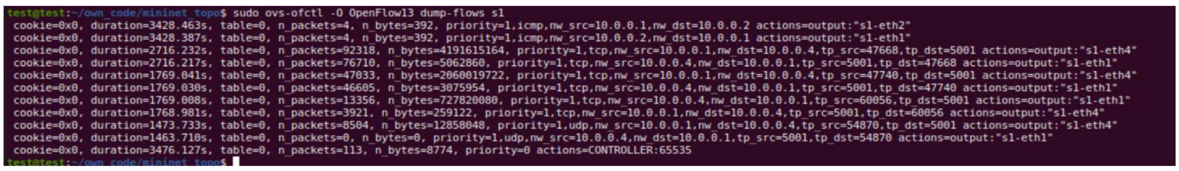
Figure 26: VoIP parallel test

The examination of OVS flows and ports is conducted through the ***‘sudo ovs-ofctl -O OpenFlow13 dump-flows s1’*** and ***‘sudo ovs-ofctl’ -O OpenFlow13 dump-ports s1’*** commands, the flow entries and port statistics in the switch s1 are inspected to observe how the SDN controller manages the VoIP traffic, including packet counts, bytes transferred, and any packet drops.



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=93, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=3537.828s
  port  "s1-eth4": rx pkts=136696, bytes=735962158, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=151915, bytes=6264764232, drop=0, errs=0, coll=0
           duration=3537.976s
  port  "s1-eth1": rx pkts=151870, bytes=6264758454, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=136753, bytes=735968956, drop=0, errs=0, coll=0
           duration=3538.003s
  port  "s1-eth2": rx pkts=23, bytes=1850, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=139, bytes=10938, drop=0, errs=0, coll=0
           duration=3538.029s
  port  "s1-eth3": rx pkts=16, bytes=1276, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=134, bytes=10496, drop=0, errs=0, coll=0
           duration=3537.941s
```

Figure 27: Parallel VoIP test-dump-flows



```
test@test:~/own_code/mininet_topo$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=3428.463s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=3428.387s, table=0, n_packets=4, n_bytes=392, priority=1,icmp,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=2716.232s, table=0, n_packets=92318, n_bytes=4191615164, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47668,tp_dst=5001 actions=output:"s1-eth4"
 cookie=0x0, duration=2716.217s, table=0, n_packets=76710, n_bytes=5062860, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47668 actions=output:"s1-eth1"
 cookie=0x0, duration=1769.041s, table=0, n_packets=47033, n_bytes=2060019722, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=47740,tp_dst=5001 actions=output:"s1-eth4"
 cookie=0x0, duration=1769.030s, table=0, n_packets=46605, n_bytes=3075954, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=47740 actions=output:"s1-eth1"
 cookie=0x0, duration=1769.008s, table=0, n_packets=13356, n_bytes=727820080, priority=1,tcp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=60056,tp_dst=5001 actions=output:"s1-eth1"
 cookie=0x0, duration=1768.981s, table=0, n_packets=3921, n_bytes=259122, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=5001,tp_dst=60056 actions=output:"s1-eth4"
 cookie=0x0, duration=1473.733s, table=0, n_packets=8504, n_bytes=12858048, priority=1,udp,nw_src=10.0.0.1,nw_dst=10.0.0.4,tp_src=54870,tp_dst=5001 actions=output:"s1-eth4"
 cookie=0x0, duration=1463.710s, table=0, n_packets=0, n_bytes=0, priority=1,udp,nw_src=10.0.0.4,nw_dst=10.0.0.1,tp_src=5001,tp_dst=54870 actions=output:"s1-eth1"
 cookie=0x0, duration=3476.127s, table=0, n_packets=113, n_bytes=8774, priority=0 actions=CONTROLLER:65535
test@test:~/own_code/mininet_topo$ 
```

Figure 28: Parallel VoIP test-dump-ports

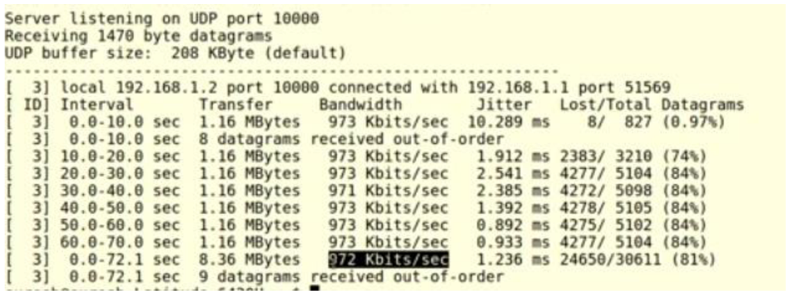
The iPerf tests demonstrate a consistent bandwidth close to the target of 64 Kbps for single VoIP calls and appropriately scaled for multiple calls. The OVS statistics indicate the efficiency of the network in handling VoIP traffic, with minimal or no packet loss. Type of Service (TOS) setting and UDP test parameters simulate VoIP traffic, allowing for the assessment of QoS, crucial for real-time communication applications. The message "Resource temporarily unavailable" suggests a potential limitation in handling the traffic, indicating the need for further network configuration or capacity scaling.

The VoIP traffic test using iPerf and OVS in an SDN environment with Mininet provides valuable insights into the network's ability to support real-time communication services like VoIP. The test demonstrates the network's capacity to maintain the specified bandwidth and QoS, ensuring the efficient transmission of voice data. The findings can guide network engineers in optimizing the network for VoIP applications, ensuring adequate resource allocation and prioritization for high-quality voice services.

5.1 QoS in SDN (end-to-end bandwidth guarrantee)

Upon completion of the 60-second test, the server logs can be examined to verify the effectiveness of the Quality of Service (QoS) implementation. This examination helps to confirm whether the QoS mechanisms functioned as intended, ensuring that traffic was managed according to the predefined priorities and bandwidth allocations.

UDP server log:

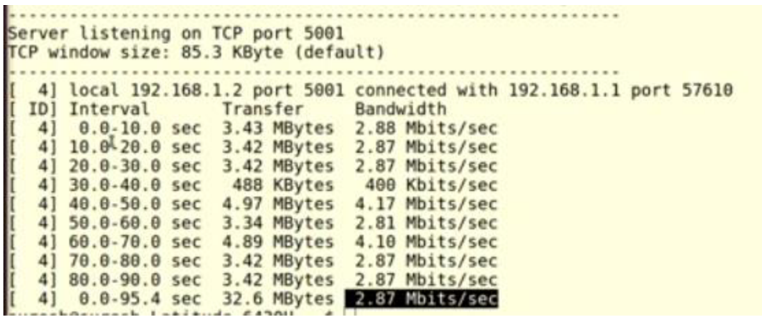


```
Server listening on UDP port 10000
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.2 port 10000 connected with 192.168.1.1 port 51569
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  1.16 MBytes   973 Kbits/sec  10.289 ms    8/  827 (0.97%)
[  3]  0.0-10.0 sec  8 datagrams received out-of-order
[  3] 10.0-20.0 sec  1.16 MBytes   973 Kbits/sec   1.912 ms 2383/ 3210 (74%)
[  3] 20.0-30.0 sec  1.16 MBytes   973 Kbits/sec   2.541 ms 4277/ 5104 (84%)
[  3] 30.0-40.0 sec  1.16 MBytes   971 Kbits/sec   2.385 ms 4272/ 5098 (84%)
[  3] 40.0-50.0 sec  1.16 MBytes   973 Kbits/sec   1.392 ms 4278/ 5105 (84%)
[  3] 50.0-60.0 sec  1.16 MBytes   973 Kbits/sec   0.892 ms 4275/ 5102 (84%)
[  3] 60.0-70.0 sec  1.16 MBytes   973 Kbits/sec   0.933 ms 4277/ 5104 (84%)
[  3]  0.0-72.1 sec  8.36 MBytes   972 Kbits/sec   1.236 ms 24650/30611 (81%)
[  3]  0.0-72.1 sec  9 datagrams received out-of-order
```

Figure 29 UDP Server log with QoS

The UDP server managed to transfer only 972 Kbit/sec out of the 6Mbps traffic generated, due to the bandwidth limit for UDP traffic set at 1Mbps. This resulted in 81% of the traffic being dropped and only 19% successfully transmitted, demonstrating the effectiveness of the Quality of Service (QoS) settings in enforcing bandwidth constraints and ensuring that the network's QoS for UDP traffic functions as intended.

TCP server log:

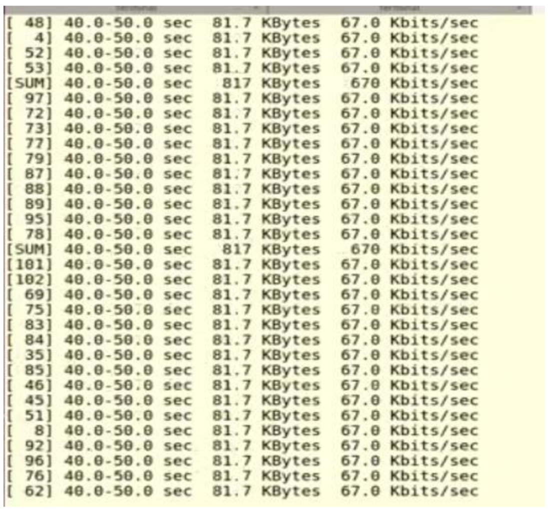


```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.2 port 5001 connected with 192.168.1.1 port 57610
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  3.43 MBytes  2.88 Mbits/sec
[  4] 10.0-20.0 sec  3.42 MBytes  2.87 Mbits/sec
[  4] 20.0-30.0 sec  3.42 MBytes  2.87 Mbits/sec
[  4] 30.0-40.0 sec   488 KBytes   400 Kbits/sec
[  4] 40.0-50.0 sec  4.97 MBytes  4.17 Mbits/sec
[  4] 50.0-60.0 sec  3.34 MBytes  2.81 Mbits/sec
[  4] 60.0-70.0 sec  4.89 MBytes  4.10 Mbits/sec
[  4] 70.0-80.0 sec  3.42 MBytes  2.87 Mbits/sec
[  4] 80.0-90.0 sec  3.42 MBytes  2.87 Mbits/sec
[  4]  0.0-95.4 sec  32.6 MBytes  2.87 Mbits/sec
```

Figure 30: TCP server log with QoS

TCP traffic achieved a rate of approximately 2.97 Mbit/sec, aligning closely with the 3Mbps bandwidth allocation set by the Quality of Service (QoS) configurations. Prior to enabling QoS, TCP traffic was limited to a mere 100 Kbit/sec. Activating the QoS feature on the system effectively guarantees a bandwidth of 3Mbps for TCP traffic, showcasing the QoS's capability to prioritize and allocate network resources to meet predefined performance standards.

VoIP server log:



```
[ 48] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[  4] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 52] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 53] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[SUM] 40.0-50.0 sec   817 KBytes   670 Kbits/sec
[ 97] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 72] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 73] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 77] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 79] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 87] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 88] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 89] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 95] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 78] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[SUM] 40.0-50.0 sec   817 KBytes   670 Kbits/sec
[101] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[102] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 69] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 75] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 83] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 84] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 35] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 85] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 46] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 45] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 51] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[  8] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 92] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 96] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 76] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
[ 62] 40.0-50.0 sec  81.7 KBytes  67.0 Kbits/sec
```

Figure 31:VoIP server log with QoS

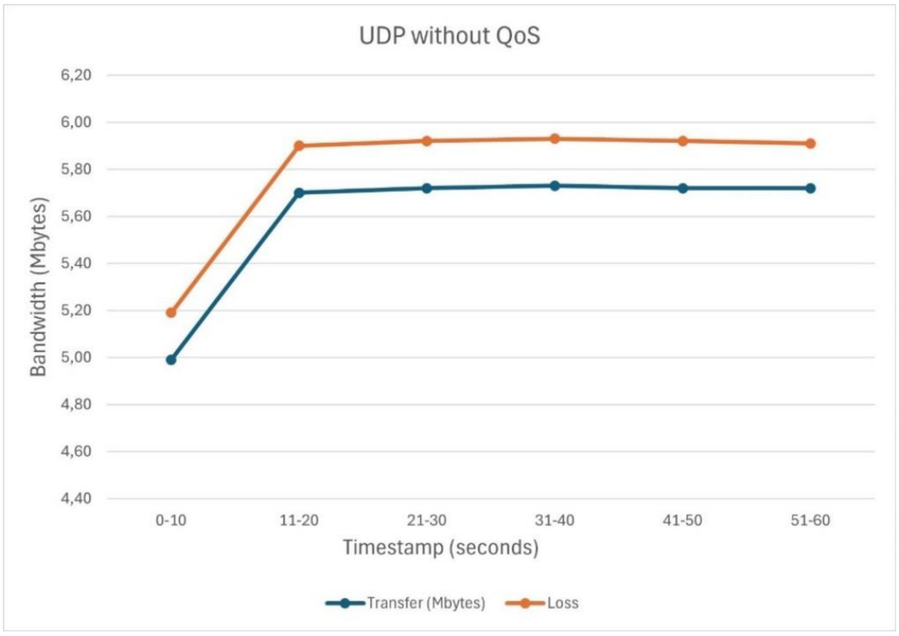
The VoIP server log indicates that with QoS enabled, the full 670 Kbit/sec traffic, as per the allocated bandwidth for this traffic type, is consistently maintained between the hosts. This ensures that all 10 calls reach the client end successfully, a feat that was unachievable without the implementation of QoS. This demonstrates the effectiveness of QoS in enforcing bandwidth allocations specifically for VoIP traffic, thereby ensuring reliable and uninterrupted communication.

5.1.1 Analysis

UDP Traffic test without QoS:

Table 5: Traffic transfer statistics without QoS

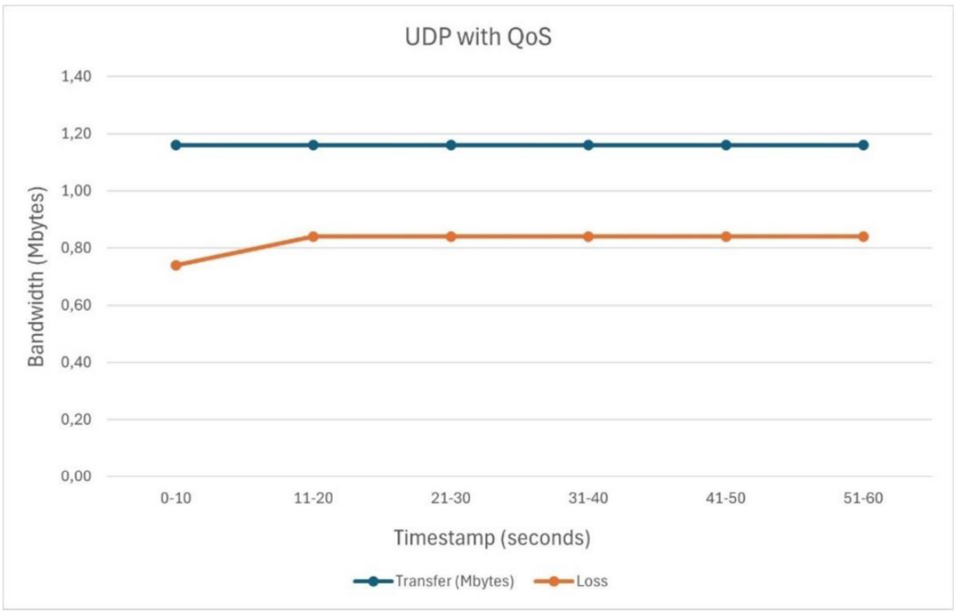
Timestamp	Transfer (Mbytes)	Bandwidth (Mbits/sec)	Jitter (ms)	Lost/Total (Datagrams)	Percent
0-10	4,99	4,19	0,664	415 / 3976	0,2
11-20	5,70	4,78	0,713	989 / 5052	0,2
21-30	5,72	4,80	0,710	1028 / 5107	0,2
31-40	5,73	4,81	0,721	1002 / 500	0,2
41-50	5,72	4,80	0,669	1033 / 5111	0,2
51-60	5,72	4,80	0,689	1018 / 5099	0,2



UDP traffic test with QoS:

Table 6: Traffic transfer statistics with QoS

Timestamp	Transfer (Mbytes)	Bandwidth (Mbits/sec)	Jitter (ms)	Lost/Total (Datagrams)	Percent
0-10	1,16	0,973	1,912	2383 / 3210	0,74
11-20	1,16	0,973	2,541	4277 / 5104	0,84
21-30	1,16	0,973	2,385	4272 / 5098	0,84
31-40	1,16	0,973	1,392	4278 / 5105	0,84
41-50	1,16	0,973	0,892	4277 / 5104	0,84
51-60	1,16	0,973	0,933	4277 / 5104	0,84



5.1.2 Result

The research employs Open vSwitch (OVS) queues as a fundamental component for Quality of Service (QoS) management within network environments. Each queue represents a distinct handling mechanism akin to different service lines at a bustling establishment, such as those designated for regular customers, VIP patrons, and elderly individuals. By analogy, specific queues are established to cater to various network traffic types, including Transmission Control Protocol (TCP) for web browsing, User Datagram Protocol (UDP) for video streaming, and Voice over Internet Protocol (VoIP) for voice communication. Each queue is allocated a predetermined bandwidth, mirroring the distribution of resources to different customer lines. As incoming network traffic traverses the infrastructure, it undergoes traffic analysis to discern its type. Subsequently, the traffic is routed to the appropriate queue based on its classification, ensuring that each type receives tailored treatment commensurate with its requirements. This method optimizes network performance and enhances the quality of service experienced by users, thus constituting a critical aspect of network management in contemporary environments.

6. Conclusion

The culmination of this research in Software-Defined Networking (SDN) with a focus on Quality of Service (QoS) through dynamic bandwidth allocation presents an in-depth investigation into enhancing network traffic management and resource optimization. The main aim was to demonstrate the feasibility of implementing end-to-end bandwidth guarantees between hosts within an SDN framework, ensuring elevated levels of QoS for varied traffic types including TCP, UDP, and VoIP. The methodology adopted involved integrating Open vSwitch (OVS) queues to manage network flows dynamically, tailoring bandwidth allocation, and prioritizing traffic to meet the distinct demands of these diverse services.

Through these testing, a comparative analysis between traditional and multipath routing methods underscored the advantages of the proposed SDN-based QoS framework. It was noted that by strategically utilizing OVS queues and effectively categorizing traffic, the network was able to adapt to changing conditions, effectively mitigate congestion, and enhance throughput. This meticulous management of traffic flows ensured that high-priority services, like VoIP, received the necessary bandwidth even under heavy network load, thus proving the hypothesis that SDN can indeed provide dedicated QoS to specific hosts.

In summary, the findings from this thesis highlight the capacity of SDN to overhaul network management approaches towards a resilient, and user-focused networking environment. The significance of this research transcends theoretical discussion, offering practical insights for network administrators and architects to develop and refine future SDN frameworks. As digital communication networks grow and transform, the strategies and knowledge gained from this study are poised to significantly influence the development of next-generation network technologies, ensuring they are well-equipped to accommodate the growing demands of contemporary applications and services.

7. References

1. Open Networking Foundation. *“Software-Defined Networking (SDN) Definition”* [Online] Available at: https://opennetworking.org/sdn-definition/. (Accessed date: 06. November 2023).

2. Taha, A. Software-Defined Networking and its Security. Master's Thesis. Aalto: Aalto University, School of Electrical Engineering, 2014.

3. Feamster, N., Rexford, J. and Zegura, E. *“The Road to SDN: An Intellectual History of Programmable Networks.”* ACM SIGCOMM Computer Communication Review, 2014. Available at: https://doi.org/10.1145/2602204.2602219. (Accesses date: 15 December 2023)

4. Braun, W., and Menth, M. *“Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices*”, 2014. DOI:10.3390/fi60203022014.

5. Shin, M., Nam, K., and Kim, H., *“Software-defined networking (SDN): A reference architecture and open APIs”*, 2012. DOI: 10.1109/ICTC.2012.6386859.

6. Rekha, P. M. and Dakshayini, M., *“A Study of Software Defined Networking with OpenFlow”*, 2015. DOI: 10.5120/21694-4798

7. Sarkar, T. K., *“SDN testbed-based evolution of flow processing-aware controller placement.”* Master's Thesis. Supervisor: Prof. Dr. Holger Karl University of Paderborn, 2017.

8. Pfaff, B. Davie, Ed. VMware., *“The Open vSwitch Database Management Protocol.”* IETF. Available from: https://tools.ietf.org/html/rfc7047 ISSN: 2070-1721 (Accessed date: 06 November 2023)

9. Lara, A., *“Using Software-Defined Networking to Improve Campus, Transport and Future Internet Architectures (Dissertation).”* Supervision of Professor Byrav Ramamurthy. Lincoln, Nebraska: December 2015. Available from: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1109&context=computerscidiss. (Accessed date: 15. January 2024).

10. L. Zhang, S. Berson, S. Herzog, S. Jamin, RFC 2205, *"Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification (1997)"*. Available at: www.ietf.org/rfc/rfc2205.txt (Accessed date: 27. January 2024)

11. Open Networking Foundation, *"OpenFlow Switch Specification"*. [Online]. Available at: https://www.open- networking.org/wp-content/uploads/.../ OpenFlow-spec-v1.3.3.pdf. (accessed 20 December 2024).

12. A Linux Foundation Collaborative Project. *"What is open vswitch?"* Available at: http://docs.openvswitch.org/en/latest/intro/what-is-ovs/ (accessed 20 December 2024).

13. Aljunaid, H. A. M., Warip, M. B. N. M., Ahmad, R. B., Anuar, S.M., Ibrahim, Z., Khairunizam, W., Razlan, M. Z. and Bakar, A. S., *"Software Defined Networks Security: Link Failure Analysis in SDN*" [Online]. IOP Conference Series: Materials Science and Engineering, 2019. Available from: https://www.researchgate.net/publication/334097438_Software_Defined_Networks_Security_Link_Failure_Analysis_in_SDN. (Accessed date: 20. February 2024). DOI: 10.1088/1757-899X/557/1/012039.

14. Karakus, Murat & Durresi, Arjan. (2016), *"Quality of Service (QoS) in Software Defined Networking (SDN): A survey. Journal of Network and Computer Applications."* 80. 10.1016/j.jnca.2016.12.019.

15. ITU-T Recommendation G.164 (1988), *"ECHO SUPPRESSORS"* International telephone connections and circuits Apparatus associated with long-distance telephone circuits.

16. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *"An Architecture for Differentiated Services. RFC 2475 (Informational)."* Updated by RFC 3260. Internet Engineering Task Force, 1998. Available at: http://www.ietf.org/rfc/rfc2475.txt. (Accessed date: 15 February 2024)

17. R. Yavatkar, D. Hoffman, Y. Bernet, F. Baker and M. Speer, *"SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style networks."* RFC 2814 (2000): 1-60.

18. Fernandes, Roulien & Alberti, Antonio. (2011*). "Ethernet-over-SDH: Technologies Review and Performance Evaluation."* Revista Telecomunicações.

19. Moeyersons, Jerico & Maenhaut, Pieter-Jan & De Turck, Filip & Volckaert, Bruno. (2020). "*Pluggable SDN framework for managing heterogeneous SDN networks*." International Journal of Network Management. 30. 10.1002/nem.2087.

20. Dezfouli, Behnam & Esmaeelzadeh, Vahid & Sheth, Jaykumar & Radi, Marjan. (2018). "*A Review of Software-Defined WLANs: Architectures and Central Control Mechanisms. IEEE Communications Surveys & Tutorials.*" PP. 1-1. 10.1109/COMST.2018.2868692.

21. Open vSwitch, *"Production Quality, Multilayer Open Virtual Switch."* [Online]. Available at : http://openvswitch.org/support/ (Accessed date: 12 December 2023)

22. S. J. Vaughan-Nichols, *" OpenFlow: The next generation of the network ?,"* IEEE Computer 44 (8) (2011) 13-15. Available at: http://dblp.unitrier.de/db/jo- urnals/ computer /computer44.html#Vaughan-Nichols11 (Accessed date: 20 December 2023)

23. D. G. Balan and D. A. Potorac, *"Linux HTB queuing discipline implementations,"* IEEE First International Conference on Networked Digital Technologies, Ostrava, Czech Republic, pp. 122-126, Jul. 2009.

24. I. Stoica, H. Zhang and T. S. E. Ng, *"A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services,"* in IEEE/ACM Transactions on Networking, vol. 8, no. 2, pp. 185-199, Apr. 2000.

25. Raussi, Petra & Kokkoniemi-Tarkkanen, Heli & Ahola, Kimmo & Heikkinen, Antti & Uitto, Mikko. (2023). *"Prioritizing protection communication in a 5G slice: Evaluating HTB traffic shaping and UL bitrate adaptation for enhanced reliability."* The Journal of Engineering. 2023. 10.1049/tje2.12309.

26. devik and Don Cohen, *"HTB Linux queuing discipline manual"* https://mirror.unpad.ac.id/orari/library/library-sw-hw/linux-1/bandwidth-manager/htb/docs/HTB%20manual.htm (Accessed date: 12 January 2024)

27. Jha, Pradeep, *"End-to-End Quality of Service in Software Defined Networking"* Available at: https://publications.scss.tcd.ie/theses/diss/2017/TCD-SCSS-DISSERTATION-2017-040.pdf (Accessed date: 15 January 2024)

28. Krishna, H., van Adrichem, N., & Kuipers, F. (2016). *"Providing Bandwidth Guarantees with OpenFlow."* In 2016 Symposium on Communications and Vehicular Technologies (SCVT) (pp. 1-6). IEEE. https://doi.org/10.1109/SCVT.2016.7797664

29. The Energy Sciences Network (ESnet) *"iperf: A TCP, UDP, and SCTP network bandwidth measurement tool"*, Abailable at: http://iperf.sourceforge.net. (Accessed date: 10 March 2024)

30. Wireshark Foundation. *"Network protocol analyzer"* [Online]. Available at: https://www.wireshark.org/ (Accessed date: 14 February 2024)

31. S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre, *"D-ITG distributed internet traffic generator,"* In Proceeding of 1st International Conference on Quantitative Evaluation of System (QEST). IEEE, pp. 316- 317, Jan. 2004.

32. Fred Klassen and AppNeta, *"Tcpreplay - Pcap editing and replaying utilities"* Available from: https://tcpreplay.appneta.com/ (Accessed date: 15 March 2024)

33. S. S. Hong and S. F. Wu, *"On interactive Internet traffic replay,"* Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3858 LNCS, pp. 247-264, 2006.