ANGLIA RUSKIN UNIVERSITY

FACULTY OF SCIENCE AND ENGINEERING

A NEW FRAMEWORK BASED ON SOFTWARE DEFINED NETWORKS TO SUPPORT QUALITY OF SERVICE IN A SLICED ARCHITECTURE

RONAK AL-HADDAD

A thesis in partial fulfillment of the requirements of

Anglia Ruskin University for the degree of

Doctor of Philosophy

Submitted: March 2021

Dedication

In loving memory of my Grandfathers, Grandmothers, and My Father-in-Law

May their souls rest in peace.

To My parents Nidhal Fadhil and Sameer Al-Haddad

To my Mother-in-Law Fatima Al-khatib

To My husband Nather Al-khatib

To Liz & Mark Haggard

To My sisters

Fatima, Jwan, Jyan, and Shereen Al-Haddad

&

To My beloved son Ahmed and beloved daughter Lara

This thesis is for you.

United Kingdom

Cambridge

2021

Acknowledgements

All the praise goes to Almighty Allah, who was always with me in this life and who gave me the opportunity to pursue this PhD journey with determination and strength.

I would like to express immense gratitude to my supervisors, especially Dr Erika Sanchez-Velazquez, for her support, encouragement and great patience in the difficult times during the years of my study. Her guidance and support in my research process is greatly appreciated. In addition, my sincere gratitude goes to my second supervisor, Adrian Winckles, for his support, suggestions and encouragement. I would like to thank Professor Marcian Cirstea, the head of the computing school at ARU for his unstinting support on my research as well as his helpful suggestions regarding my future career. I would like to thank Dr NezHapi-Dellé Odeleye for her immense support during the difficult times that I faced; she was always there to offer me her support and best solutions. I also would like to thank Dr Silvia Cirstea for her support during the data analysis of this research.

I am deeply indebted to my mum and dad and my mother-in-law, you have always been strong supporters and a source of positive energy, inspiration, persistence and steadfastness in the face of difficulties, despite the thousands of miles between us. Thanks for your love and prayers, which have protected me beyond how you could imagine and have enabled me to reach where I am now. Also, many thanks to my sisters, you all are the light in my life. You have helped make the impossible, possible, with your brightness, joy, boundless enthusiasm and aspiration every day.

I would like to express my profound gratitude to my husband Nather Al-khatib for his exceptional support and love at all times. His support in my pursuing my study has been invaluable to me. He turned all the challenges we went through together into memories that I am very proud of. Thank you for always believing in me and for being always here to listen. My heartfelt thanks go to my son Ahmed Al-khatib and my daughter Lara Al-khatib, who have been extremely patient throughout these years. Their understanding and patience enabled me to complete this research.

Very special thanks to the Haggard family, especially Professor Mark Haggard from the University of Cambridge, for giving his precious time and valuable input and feedback on my thesis. My deepest gratitude to his wife, Liz Haggard, who was the earth mother for my family and me, here in Cambridge. She was the motivator who changed my entire PhD journey and made it full of joy and excitement, supporting me in so many ways and always knowing how to bring the best out of me. Without her, I could not have finished my PhD.

My deepest appreciation for all my friends, and colleagues at the Cambridge campus. I thank Dr Arooj Fatima for the fruitful discussions, suggestions, and cooperation throughout these years. She has been a true mentor since the first few days when I started my research. Also, thanks to other colleagues at the Chelmsford campus, especially Belema Agborubere, who shared inspirational research and teaching discussions in SAW302. Last, but not least, I would like to extend my appreciation to my friends Dr. Anna, Sawsan Al-Qaysi and Dr Ali Al-lami, for the unconditional love, support, encouragement and the lovely company. Thank you for being part of my life.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE AND ENGINEERING

DOCTOR OF PHILOSOPHY

A NEW FRAMEWORK BASED ON SOFTWARE DEFINED NETWORKS TO SUPPORT QUALITY OF SERVICE IN A SLICED ARCHITECTURE

RONAK AL-HADDAD

March 2021

This study focuses on the challenges regarding traffic performance issues, including bottlenecks and congestion in network traffic communication in Software Defined Networks (SDN). These challenges are caused by the increasing demand for network services and quality across a wide range of digital applications on the internet. The study is aimed at improving network throughput as well as reducing end-to-end delay and jitter for the packet transmission process. A framework design is proposed that involves applying four main technologies: Traffic engineering (TE), SDN, network hypervisors and network slicing. Network function virtualisation (NFV) pertains to the incorporation of comprehensive automation and centralised functions at SDN's core by combining physical and virtual systems using virtual machines (VMs), which enhances network productivity, reliability, scalability, and integrity.

In this work, a new methodology of QoSVisor and a Packet Tagging Prioritisation (PTP) Agent extension algorithm for video, audio and data over TCP SDN-slicing networks has been developed and tested. QoSVisor works via a local weighting function, which sorts packets by the port and flow policy setup, depending on each packet (flow, packet ID and weighted tag), with strict priority policy added to this algorithm to guarantee more precise quality of service (QoS) and low latency queueing (LLQ) for video and audio traffic types. Furthermore, a new traffic shaping (TS) algorithm is proposed as a new implementation of (QoS) to work as a bandwidth management technique for optimising performance in an SDN-sliced network, Two algorithms, namely "packet tagging, queueing, forwarding to queues" and "allocating bandwidth", are proposed and developed for implementing a weighted fair queuing (WFQ) technique, which works mainly as a part-function of TS the queueing mechanism, to reduce congestion and smooth traffic flow. Additionally, an ordinary queueing algorithm, First In First Out (FIFO), has been developed and implemented in an SDN-sliced framework as a baseline condition for quantitative performance comparisons to evaluate the network characteristic behaviour of SDN and QoS.

The novel comparative approach for evaluating the implemented algorithms shows significant improvement on traffic performance issues. The result show that the more advanced algorithms TS and QoSVisor deliver more effective allocation of bandwidth than FIFO, and that they significantly reduce critical delays. The results show throughput for TS and QoSVisor is similarly high for the two most demanding flows, namely video and audio and so, traffic types were prioritised. Giving lower priority for audio followed by data, it is demonstrated that TS lets through just over twice the amount of information compared to QoSVisor. Comparing audio with data flow there was an approximately five-fold advantage for the former compared to the latter data flow type.

Keywords: *Network congestion, network performance, SDN, slicing, QoS, FIFO, weighted fair queuing (WFQ), QoSVisor algorithm, Packet Tagging Prioritisation (PTP) Agent, OpenFlow, SPSS analysis of variance (ANOVA).*

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Appendixes

List of Acronyms

AF	Assured forwarding
ANOVA	Analysis of variance
API	Application programming interface
BE	Best effort
CLI	Command-line interface
CPU	Central processing unit
DiffServ	Differentiated service
DF	Degrees of freedom
DL	Deep learning
DSCP	Differentiated service code point
EF	Expedited forwarding
FIFO	First In First Out
FQ	FIFO queue
IEEE	Institute of Electric and Electronic Engineers
IntServ	Integrated services
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript object notation
KVM	Kernel-based Virtual Machine
LLQ	Low latency queuing
MAC-address	Media access control address
ML	Machine learning
MPLS	Multi-protocol label switching
NETCONF	Network Configuration Protocol
NFV	Network function virtualisation
OF	OpenFlow Protocol

ONF	Open Networking Foundation
OVS	Open vSwitch
Petasq	Partial eta squared
PHB	Per-hop behaviour
PTP Agent	Packet Tagging Prioritisation Agent
QoS	Quality of service
QoE	Quality of experience
QPLs	Queue priority levels
REST	Representational state transfer
RSVP	Resource Reservation Protocol
RTT	Round trip time
SDN	Software-defined networking/Software networks
SE	Residual standard error
SPSS	Statistical Package for the Social Sciences
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToS	Type of Service protocol
TS	Traffic shaping
TE	Traffic engineering
TTL	Time-to-Live
VLAN	Virtual local area network
VM	Virtual machine
WANs	Wide area networks
WFQ	Weighted fair queuing
5G	Fifth Generation

A NEW FRAMEWORK BASED ON SOFTWARE DEFINED NETWORKS TO SUPPORT QUALITY OF SERVICE IN A SLICED ARCHITECTURE

RONAK AL-HADDAD

COPYRIGHT DECLARATION

1. Chapter One: Introduction

1.1. Research Motivation

The demand for digital applications and associated internet traffic has increased exponentially in recent years. Aggregate statistics of all types of Internet Protocol (IP) media were published by Cisco (2019), including Internet video, IP Video on Demand (VoD), video files exchanged through file sharing, video-streamed gaming, video conferencing, Skype, and Internet of Things, online games, among others. To respond to these developments, Cisco developed a complex dynamic connection system to monitor traffic from connected devices, starting in 2012. They found 8.7 billion connected objects worldwide in December 2012, while the sum in May 2014 was over 12.3 billion (Cisco, 2013). Moreover, Cisco has estimated that between 80% and 90% of total IP traffic would stem from IP traffic by application category. By 2022, IP video traffic will represent 82% of all traffic, as shown in Figure (1-1). In the next five years, these numbers are projected to triple. Overall, between 2017 and 2022, IP traffic is projected to increase at a Compound Annual Growth Rate (CAGR) of 26%, while monthly IP traffic is projected to increase to 50 GB per capita by 2022, from 16 GB in 2017.

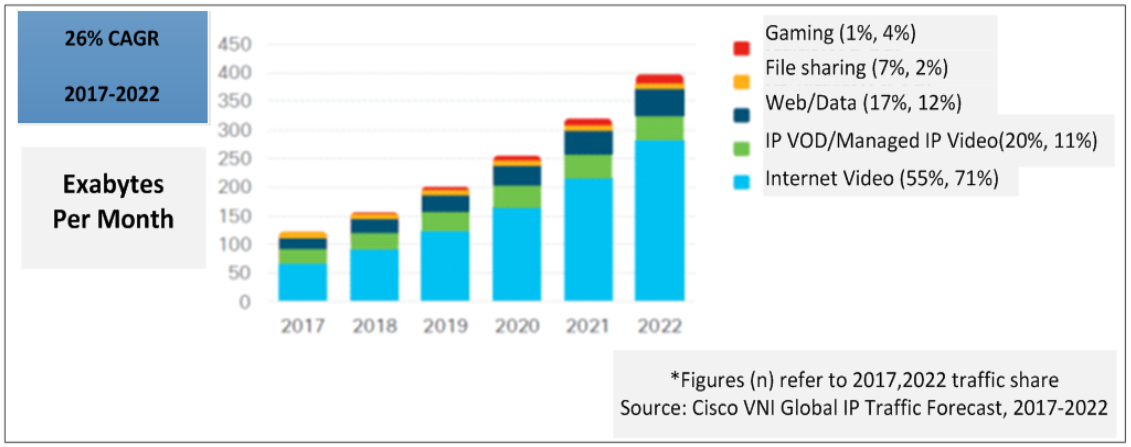


Figure (1-1) Global IP traffic by application category (Cisco, 2019)

With this increase in Internet traffic, the consequences of video developments are difficult to overestimate, with relatively stable traffic flows (stemming from peer-to-peer traffic [P2P]) evolving into more complex, dynamic traffic trends. These advances come with concomminant demand for improved network Quality of service (QoS), for example, guaranteed bandwidth and low latency (Wu et al., 2001). A future scenario with video applications is shown in Figure (1-2) along with the relevant application bandwidth requirements. At present, bandwidth requirements constitute a fraction of potential future need.

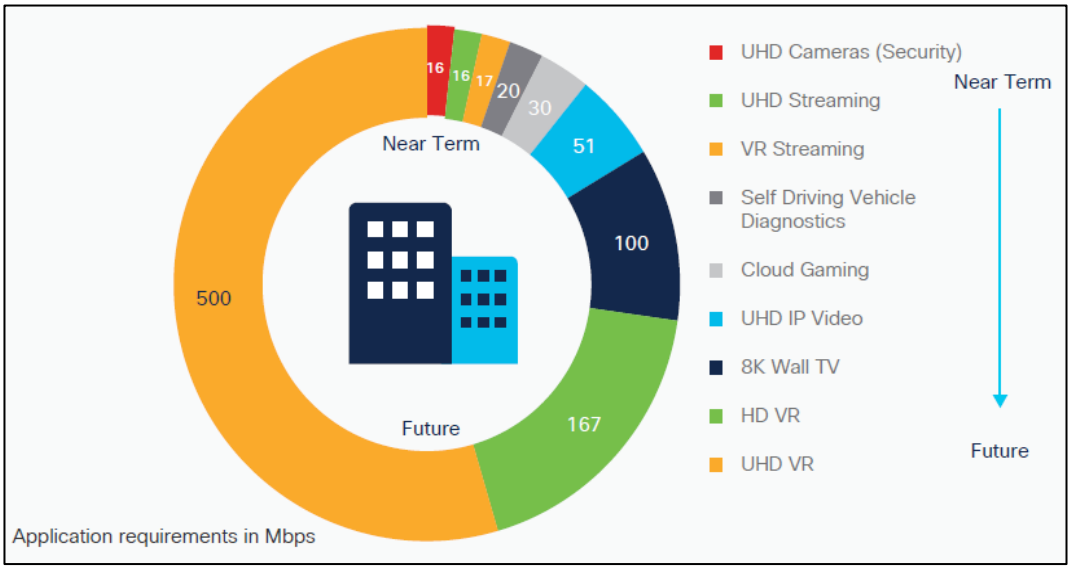


Figure (1-2) Significant demand for video in the home of the future (Cisco, 2019)

Whilst average internet traffic has been gradually growing, peak traffic (or traffic during the busiest 60 minutes of the day) in increasing at significantly faster rate. From 2017 to 2022, the global busy-hour use of the Internet (which the Cisco Annual Internet Report covers across six distinct geographic regions: Asia Pacific (APAC), Central and Eastern Europe (CEE), Latin America (LATAM), Middle East and Africa (MEA), North America (NA) and Western Europe (WE)) will have risen at a CAGR of 37% compared to 30% for average traffic levels (Figure 1-3). The underlying explanation for rapid traffic growth during peak hours is video applications. Unlike other modes of traffic, which are distributed equally over the entire day, such as browser navigation or file sharing, video appears to be used most during “prime time”. Due to this ever-increasing traffic share and given that video has a relative higher peak-to-average ratio, peak

hours are busier, and their traffic grows faster than average traffic flows. That is, regarding the evolving composition of internet content, the difference between peak and average traffic flows has increased. Real-time content, like live streaming, ambient video and video calling, has an even higher peak-to-average ratio than on-demand.

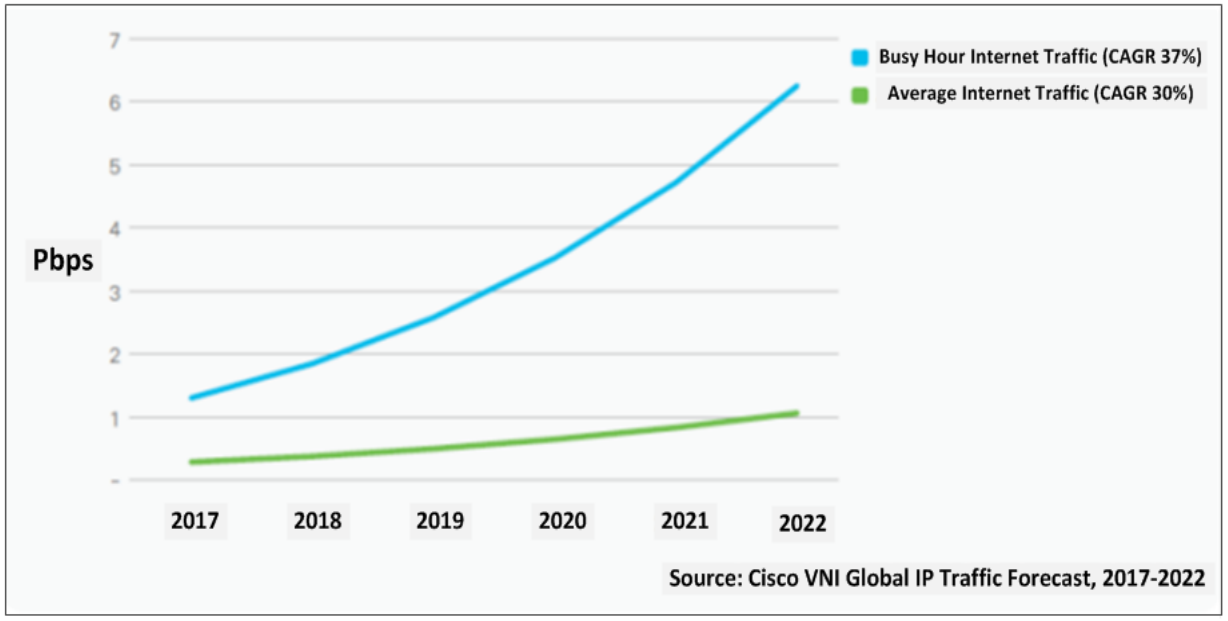


Figure (1-3) Busy hour compared with average Internet traffic growth in petabits per second (Pbps) (Cisco, 2019)

As a result of the substantial increase in connected devices, the traditional existing network architecture in the commercial sector is not sophisticated enough to fulfil the demands of individuals, businesses, and carriers. Put simply, network designers are limited by the shortcomings of the existing network technologies (Ghasempour, 2019). In 2016, the global yearly IP traffic was estimated to have exceeded 1.2 zettabytes (Cisco, 2017), and about 3.3 zettabytes yearly are projected by 2021. Furthermore, by 2021, 82% of the Internet traffic of consumers will be IP videos and 13% of the total video traffic will be live (Index, 2017). In addition, the number of devices with an Internet connection will be three times the global population headcount.

The following is a description of the shortcomings of the existing commercial network architecture as well as obstacles to fulfilling the QoS requirements of an adequate converged network.

- **Complexity:** To date, network technology has consisted mainly of discrete collections of protocols designed to link hosts efficiently across arbitrary distances, communication speeds and topologies. In recent decades, the industry has modified network protocols to provide greater efficiency and reliability, broader connectivity, and tighter security to meet business and technological needs. However, protocols tend to solve specific problems, without any fundamental abstraction, as they are usually generated in isolation. For instance, to add or transfer any device, several switches, routers, firewalls, web authentication portals, among other things must be activated by IT departments, as well as updating access-control lists (ACLs), VLANs, quality of service (QoS), and other protocol-based mechanisms using device-level management tools (Raza et al., 2014). Moreover, network topology, vendor switch model, and software version must be considered.

Because of this complexity, networks today are largely static as IT departments seek to reduce the possibility of service disruption. Today, several organisations are operating on an IP converged voice, data and video network. The implementation of these tools is manual even if current networks can provide differentiated QoS levels for different applications. IT must configure the equipment of each vendor independently as well as calibrating network bandwidth and QoS parameters, all on a per program and per-session frequency. The network cannot adjust dynamically to evolving traffic, device and user requirements due to its static characteristics (Akin and Korkmaz, 2019; Kreutz et al., 2014).

- Inconsistent policies: The complexities of existing networks make it extremely challenging for IT departments to establish clear protocols for mobile user access, network speed, safety, QoS, and other policies, thus leaving businesses vulnerable to security breaches, regulatory infringements and other negative outcomes (Ferro and Ruiz, 2015).

- Scalability challenges: As demands on data centres expand, networks require rapid growth too, as shown in the statistics above. However, with hundreds to thousands of network devices that need to be configured and maintained, the network becomes more and more complex. IT

has depended on oversubscription to scale the network, from predetermined traffic dynamics. Traffic patterns are highly evolving and volatile, thus being unpredictable in today's virtual data centres.

To remain competitive, business operators must provide consumers with increasingly high value, better-differentiated services. Multi-tenancy complicates the activities more, as the network must support various user groups with different performance needs. The current network infrastructure in business, especially at the operator level, involves key operations that are relatively straightforward, such as directing customer traffic flow to provide personalised performance control or on-demand delivery, network-level specialised devices are needed to implement new services, which increases capital, operating costs, and time-to-market expenses (Hakiri et al., 2014).

- Vendor reliability: Carriers and organisations are pursuing the development of new capacities and services to be applied rapidly to adapt to the changing market and customer requirements. However, vendors' product cycles, which can exceed three or more years, impede their ability to respond. Lacking standard open interfaces, network operators' ability to adapt their network to their environment is limited. The industry has come to a tipping point from this disparity between consumer requirements and network capacities (ONF, 2012).

Limiting physical network architecture has a major impact on aggregate network system performance (Haldar and Agrawal, 2014). Due to the exponential growth of network traffic, the difficulties posed will seriously impair the efficiency of the existing network infrastructure. The underlying hardwired application of routing rules signifies a lack of flexibility from devices, such as traditional hardware switches, in dealing with various packet types with different contents (Kim and Feamster, 2013). In addition, the networks that form the core structure of the internet must be capable of responding to changes without requiring too many hardware or software recalibrations.

Up until early 2020, the deficiencies in the packet switching for audio, video and data were largely unknown, but this changed when the COVID-19 pandemic occurred. As social distancing increased, people needed to find new ways to communicate, for instance, through video chat, for such as every school child needing communication technologies, like Skype, for

classes (also Zoom, Slack and Cisco Webex Teams, and other apps). During this engagement, users experienced significant failures and video asynchronies, causing frustration. In addition, a bigger institution like the NHS, Universities, businesses and non-governmental organisation (NGOs) also require these technologies. The pandemic disrupted all previous expectations and projections, changing how traffic generation and distribution impacted on existing networks; negatively affcting application performance (Koeze and Popper, 2020). Hence, it was acknowledged that there was a real need for high QoS and new systems to handle network traffic better by improving the network throughput, reducing end-to-end delay and dealing with traffic issues like bottlenecks and congestion.

1.2. Problem statement

Nowadays, traditional network systems cannot be reprogrammed or retasked seamlessly (Hu, et al., 2014). This is because, the control plane and the data plane are interconnected and embedded into the same hardware in traditional hardware switch components, as shown in Figure (1-4).

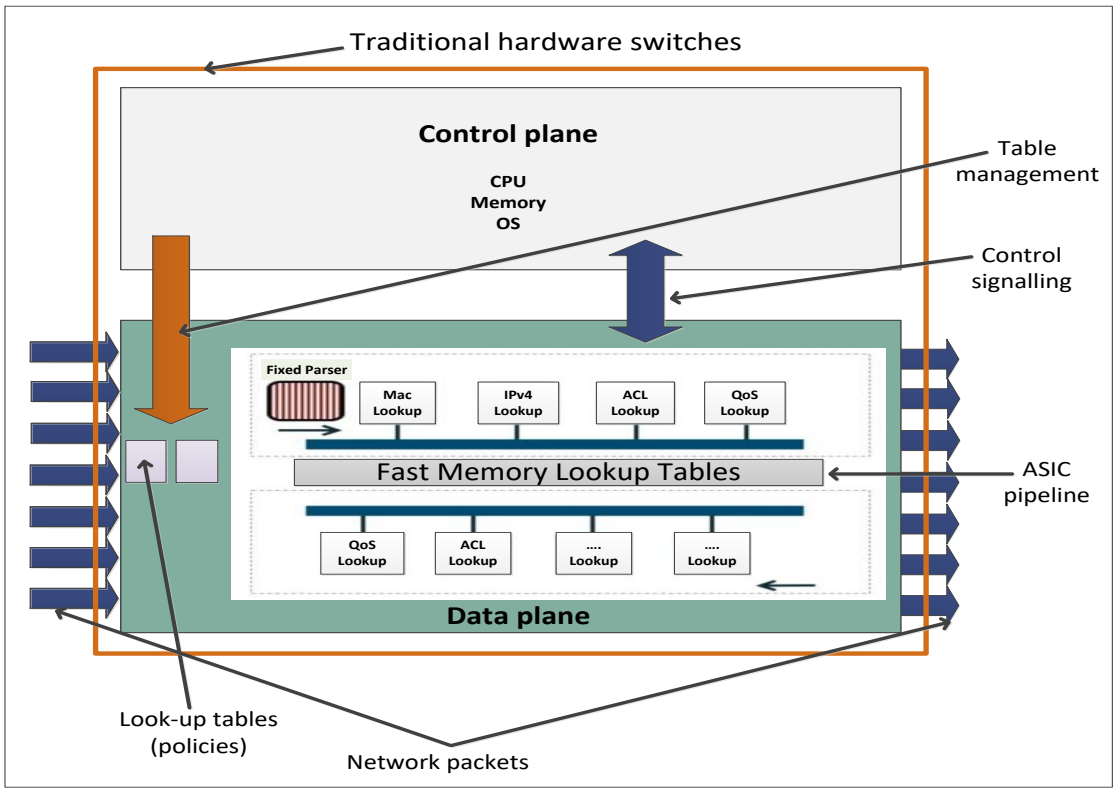


Figure (1-4) Traditional switch architecture

As a consequence, when trying to resolve a variety of network scenarios at various priority levels, this can have a negative effect on the efficiency and management of network. In particular, there is the threat of network congestion as traffic rises quickly and become unsustainable, which will lead to a service that is not consistent and likely to be interrupted. This can be remedied by increasing the bandwidth and building new channels for the data to be transmitted and fixing some of the other vulnerabilities.

However, there has also been an enormous rise in the volume of information that corporations need to manage. This means that network operators are under a greater level of pressure to provide high-quality user experience and application efficiency. Bandwidth boosting is, unfortunately, a costly operation, particularly when it is required for an individual network component and most cases, it is simply not affordable. It is not cost-efficient when there is only moderate traffic (Sllame and Aljafari, 2015). A possible solution in the shape of programmable networks was suggested, the main objective is to enable software developers to use network tools with other computing and data storage resources in a simple, efficient manner, as with other technologies.

In Software-defined networks (SDN), the data plane component comprises a particular form of hardware or software resource that is developed to address packet forwarding, whilst controllers constitute a software arrangement type that functions through a commodity hardware foundation. Through open interfaces, controller components can configure heterogeneous components with greater accuracy and at a faster pace (Nunes et al., 2014). Because of its new constitution, intelligently enforced by its separating and unified traffic engineering (TE), several people have found SDN a potential solution to handle QoS (Jarraya et al., 2014). In the context of innovation and constant networking growth using SDN technology, this literature introduces a new era, which appears to be more versatile in comparison with the complex traditional infrastructure, which manages the entire network component function (Hakiri et al., 2014). In addition, the 'network congestion issue' (Wang & Xia, 2015) and the network QoS problem' (Karakus & Durresi, 2017) have significantly grown in importance recent years, becoming the focus among researchers, both from academia and industry.

In a study that measured the current size of the software-defined networking market, several important factors were found, such as the CAGR of 26.8% projected for the global

market such networking, representing a rise from USD 8.8 billion in 2018 to USD 28,9 billion by 2023 (Cisco, 2019). The study also showed that advanced network management solutions are increasingly required on the market to handle the rising network traffic and related complexities. Hence, this is a significantly growing market. It was also revealed that the infrastructure used for virtual network features has contributed to a rapidly growing demand for SDN controllers. The major users of SDN technology are anticipated to be data centres and service providers (Cisco, 2019). Therefore, an appropriate framework needs to be established to resolve these needs and problems.

Another interesting approach, network slicing, has also been implemented by SDN technology. This a virtual networking architecture in the same family as SDN and network function virtualisation (NFV). Network slicing acts as a layer between the network forwarding and control planes (Han et al, 2015); Network virtualization technologies are transforming modern networks through software-based automation. By partitioning network architectures into virtual components, SDN and NFV provide much greater network versatility. Essentially, network slicing enables many virtual networks to be built in a single physical infrastructure. The legacy of physical infrastructures is being divided into individual, logical elements that can be programmatically managed to provide customised connectivity levels, with network slicing being at the cutting edge, offering optimised versatility.

The implementation of network slicing using the FlowVisor method for splitting the production network was undertaken by a group of researchers (Sherwood et al., 2010). FlowVisor is a special purpose controller implementation, working as a transparent proxy in SDN OpenFlow (Sherwood et al., 2009). Its main contributions include the possibility of slicing any control plane message format used in OpenFlow (OF), and being the first slicing mechanism that enables a user-defined control plane to exercise control over the production hardware's forwarding behaviour. The exact decisions on slicing so as to optimise use of the network's resources have to take placein the context of the available bandwidth and minimum jitter. In topology, bandwidth, switch central processing unit (CPU) rate, forwarding table quota and the slice controls, a network slice is specified. Slice-handled traffic is specified in packet headers (Flowspace) by bit patterns. The tool FlowVisor uses OpenFlow as a hardware abstraction layer, which logically sits between controlling and forwarding paths on a network

computer, generating virtual slices in wired and wireless networks in isolation. Moreover, when combining NFV and SDN, this offers the controller a transparent proxy in SDN OpenFlow.

1.3. Research Aim and Objectives

The research study reported in this thesis has been driven by the need to address the previously stated problem and by a systematic review of the relevant literature related to SDN. Accordingly, the aim is to establish the concept of bringing SDN together with QoS mechanisms and slicing strategies. This combination opens the possibility of providing new solutions to SDN challenges and is the foundation of the research presented in here. The study is focused on using these three technologies to find new solutions to improving the key characteristics of QoS (delay, throughput and jitter) for delay sensitive media applications. A new architecture, “Traffic Shaping (TS) algorithm in Sliced-SDN and QoSVisor algorithm with a Packet Tagging Prioritisation Agent (PTP Agent) extension framework”, is proposed that depends on the QoS features to filter pre-classified packet data to prioritise those for delivery.

In adition, to achieve the best performance for applications using different queueing techniques in an SDN. It is intended that this framework can markedly improve the QoS for streamed data. To fulfil this aim, the researcher believes that it is possible to propose and evaluate a “new system design" by implementing a hybrid system using the legacy technology within the new technologies to address network performance problems. Thus, deriving a new QoS framework using SDN-based slicing and adopting the legacy QoS mechanisms is a valid hypothesis for research.

The following hypothesis has been determined for this study:

> *“An enhanced quality of service (QoS) framework in software-defined networks (SDN) provides a suitable solution for solving congestion and latency problems, thus leading to measurably augmenting QoS for streamed data in networks”.*

The following objectives have been pursued in order to test this hypothesis:

 # Objective (1): Research the fundamental concepts for implementing QoS in SDN, by classifying QoS protocols, as well as optimising techniques, such as traffic engineering (TE), to fill the knowledge gap.

 # Objective 2: Collect and analyse data for implementing the FIFO, Traffic Shaping and QoSVisor algorithms in sliced-SDN.

 # Objective 3: Design, implement and evaluate a novel approach for evaluating queuing systems in sliced-SDN.

 # Objective 4: Design, implement and evaluate a novel framework for QoS implementation in sliced-SDN.

 # Objective 5: Critically assess and conclude the effectiveness of the proposed framework using a new comparative statistical analysis.

1.4. Research Question

To investigate and find solutions to improving network throughput, reducing end-to-end delay and dealing with traffic issues, like bottlenecks and congestion in network traffic communication, the core research question for this study is as follows.

How to establish a QoS framework that can help solve the congestion, resource allocation and delay problems found in networks, based on slicing technology within SDN?

This question is divided into more focused sub-questions as follows:

1. *What are the available QoS architectures and their limitations?*
2. *What are the available slicing techniques within SDN?*
3. *Can SDN help to reduce the network congestion, resource allocation and delay problems found in networks for delay-sensitive traffic, such as video and/or audio applications?*

1.5. Original Contributions to Knowledge

This research makes a significant contribution to knowledge in the discipline of network traffic communication through innovative solutions for improving the network throughput, reducing end-to-end delay and dealing with traffic issues, like bottlenecks and congestion being proposed.

An issues of research in practical as well as theoretical domains, which is timely given the current interest in big data management and the proliferation of on-line data communication (especially real-time data) which is an important current area of research. The details of the original contributions to knowledge are listed as follows.

1. Comprehensive study and background review of SDN technology.

2. A systematic investigation of quality of service (QoS) frameworks and mechanisms for SDN and slicing studies.

3. A FIFO algorithm is developed and implemented in a sliced-SDN framework as a baseline condition for quantitative performance comparisons.

4. A new Traffic Shaping (TS) algorithm is proposed as a bandwidth management technique to optimise performance in sliced-SDN, Traffic shaping works mainly via the WFQ part-function of TS the queueing mechanism, to reduce congestion and smooth traffic flow.

5. A new methodology of QoSVisor and a Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for video, audio and data over TCP SDN-Slicing networks is developed and tested. QoSVisor works via a local weighting function, which sorts packets by the port and flow policy setup, depending on each packet (flow, packet ID and weighted tag) to provide low-latency queueing (LLQ) for video and audio traffic types.

6. A novel comparative approach for evaluating queueing systems.

1.6. Thesis Structure

The thesis consists of eight chapters and a summarised description of the remainding ones is presented as follows.

- **Chapter Two:** Presents the background review of the SDN reference model, followed by the SDN slicing mechanisms. This chapter also identifies the research challenges and future directions.
- **Chapter Three:** Contains the literature review and related work in the methods and mechanisms of quality of service (QoS) architectures in SDN. This is followed by classification and marking in the Differentiated Services (DiffServ) model. The methodologies of the queueing and traffic congestion management techniques used in different studies in the literature are also reviewed.
- **Chapter Four:** Considers the most relevant methods to be adopted to conduct the research project. It includes an explanation of the research methodology, experimental design and analysis, as well as how the research objectives were achieved, followed by the research design phases.
- **Chapter Five:** Presents the FIFO implementation methodology and a system overview, highlighting the limitations of FIFO algorithm that has been developed and implemented in a sliced-SDN framework, used to establish the baseline condition for quantitative performance comparisons. A complete SDN FIFO model is presented, followed by a description of the testbed and experimental design for the FIFO algorithm module in sliced-SDN.
- **Chapter Six:** Presents the new contribution to implementation of QoS management using the new Traffic Shaping (TS) algorithm in the sliced-SDN.
- **Chapter Seven:** Introduces the new proposed design of the QoSVisor and PTA agent framework, followed by the main components with the workflow of the proposed QoS model.
- **Chapter Eight:** Contains the Data analysis and evaluation, including a novel comparative approach for evaluating queueing systems, a comparison of the FIFO queueing system, Traffic Shaping algorithm and the QoSVisor framework in sliced-SDN using SPSS.

- **Chapter Nine:** Provides an evaluation of the objectives, conclusion of the thesis and proposed directions for future work.

1.7. Summary

After this introductory chapter, Chapter 2 serves as an introduction to traditional networking and SDN. The SDN reference model that has been comprehensively examined in terms of diverse conditions and scenarios, along with the SDN slicing mechanisms. The aim is to provide the reader with a comprehensive background of the SDN reference model, followed by the SDN slicing mechanisms and identifies the research challenges and future directions.

2. Chapter Two: A comprehensive background review

In this chapter, first, traditional networking is introduced and then, the SDN reference model is comprehensively examined in terms of diverse conditions and scenarios, along with the SDN slicing mechanisms being explained. The aim is to give the reader a comprehensive understanding of work relating to SDN technology.

2.1. Introduction to Traditional Networking

Generally, traditional computer networks are implemented using numerous hardware devices, including switches, routers, and different middle boxes that implement several complex algorithms and protocols (Nunes et al., 2014). 'Middle box' is a term concerning a component, which can alter or impact on traffic for a given objective not linked to packet forwarding, including a firewall (Huang et al., 2017). Further, the network administrator is responsible for developing rules and policies that are in accordance with diverse network conditions and situations. For this, low-level configuration requests need to be manually generated from high end policies. Here, the issue is that, typically, administrators need to accomplish their goals using limited essential resources, while ensuring that the devices have sufficient flexibility for addressing inconsistent conditions. Usually, network devices refer to black boxes that are vertically assimilated. Hence, there is a high error rate and the network managers have to face numerous obstacles. Therefore, network maintenance and construction can be regarded as difficult tasks.

It is important to note that the traditional network architecture is also under immense pressure, because of the rapidly increasing popularity of Internet of Things (IoT) as well as cloud servers. Thus, precise regulation of server-to-server networking, improved safety, stability, as well as ease of functionality are required. Further, applications are increasingly spanning across numerous databases as well as servers that are geographically separate and isolated from each other. Hence, the essential user-to-server processes have become overshadowed by the server-to-server interactions' increasing amount of traffic (Lee and Lee, 2015).

With an increase in the number of components of traditional networks as well as requests concerning IoT, it is important for information and communication technology resources to keep evolving (Feamster, Rexford, and Zegura, 2014; Huang et al., 2010). Several researchers have demonstrated that traditional networks do not have the advanced technology required to meet the exponentially increasing demand for network services and the QoS standards. A major factor causing this, as aforementioned, is that network devices tend to be vertically assimilated, which leads to a diverse system that is complicated to maintain. The traditional network devices separate their functionality vertically in three planes, which are the (1) control plane, (2) data plane, and (3) management plane, each communicating horizontally or vertically with the other planes (Serrano, Pasamontes, and Moreno, 2016; Hu, Hao, and Bao, 2014). As noted by Al-Haddad and Sanchez (2018) as well as other studies (Nunes et al., 2014; Pascual Serrano, Vera Pasamontes, and Girón Moreno, 2016), traditional networks can only enable vendor specific protocols.

As an example of traditional networks routing devices and routing algorithms, consider packet forwarding. In traditional networks, once a routing device receives a packet, it implements rules that are built into its firmware for determining its destination device and routing path. Data packets that must be delivered to the same destination tend to be handled in a similar manner and such an activity occurs in inexpensive routing devices. Some routing devices can only perform traditional routing, i.e. static routing, dynamic routing, etc., whilst others can support more complex functionality, such as QoS classification, etc. They are able to treat diverse types of packet using different methods according to their content and nature. For example, in traditional networks, a Cisco router device uses modified local router programming techniques to help a user establish the various flow priorities and hence, direct intervention in the router queue formation is possible. That is, a customised local router setup can help with traffic congestion as well as prioritisation control with enhanced efficiency. However, this approach has the limitation of the current network components' restrictions when there is increased network traffic. This can create severe problems as well as considerably diminish the overall performance. A rapid increase in the network traffic will result in greater challenges concerning security, stability, dependability, efficiency, and scalability (Hu, Hao, and Bao, 2014).

Reconfiguring and reorienting the traditional network processes continues to be difficult. Moreover, traditional networking components do not have the flexibility or dynamic

characteristics for addressing the various types of packets or their different content. As noted by Kim and Feamster (2013), this is as a result of the rigidity of the routing protocols that does not allow for any adaptability. This is a significant restriction, because, as Bakshi (2013) pointed out, this is due to the traditional network operations that cannot be easily reprogrammed or re-tasked.

Today, it is more complex to execute the traditional design of networks from an architectural perspective. As abovementioned, traditional network devices separate their functionality into three planes, which are the (1) control plane, (2) data plane, and (3) management plane and as shown in Figure 2-1, each plane communicates horizontally or vertically with the other planes in switch device.

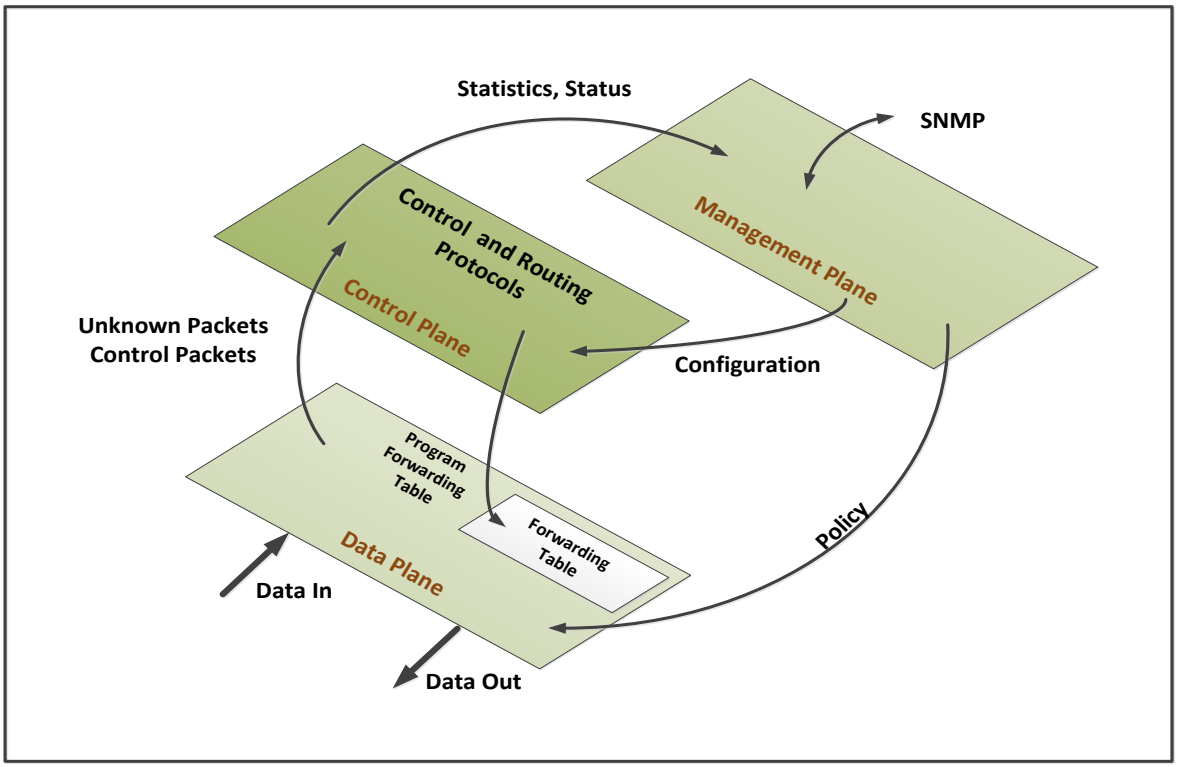


Figure (2-1) Traditional switch architecture (Goransson, Black and Culver, 2016)

The data plane in Figure (2-1) follows a logic and includes various ports that help receive as well as transmit data packets with a forwarding table. This plane is responsible for scheduling as well as buffering packets along with changing the packet headers and forwarding them. Once the forwarding table includes information concerning an arriving packet's header, a packet can be altered and forwarded without the interference of the other two planes.

The control plane in the same figure is involved in processing other control protocols that may affect the forwarding table, because of the switch and the way it is configured. Moreover, this plane keeps the information in the forwarding table, because of which, the data plane is able to route the traffic independently. The control protocols are able to address the network's active topology using microprocessors along with the accompanying software.

The networking device management plane involves the network administrators in charge of overseeing, configuring, monitoring and managing the services with all the other layers of network components of the system. In addition, a type of network management system is used by the network administrators to communicate with the management plane. As shown in the figure above, Simple Network Management Protocol (SNMP) is utilised for managing and modifying the information to change devices' behaviour in IP networks.

Owing to the rising speed of the media applications, the network devices also needed to be speeded up. To ensure improvement in performance, the parallel processing was increasingly distributed over, including numerous 'blades', with every functioning advanced microprocessor being able to access a distributed forwarding table independently. However, with time, the interfaces' speed increased to a point at which it was not possible to execute the header inspection as well as route the table lookup by the software. This, in turn, increased the difficulty and risk of error concerning network performance along with making management control harder (Goransson, Black, and Culver, 2016; Al-Haddad and Velazquez, 2018).

To address the above issue, the data handling protocols needed be implemented as software modules, rather than the hardware embedded elements. It is worth noting that this notion is similar to the SDN concept (Hakiri et al., 2014). This architecture development, as noted by Feamster, Rexford, and Zegura (2014), took place from 2001 to 2007, being undertaken by Nicira Networks, based on Stanford, CMU, and Princeton researchers' tests and experiments (Jarraya, Madi, and Debbabi, 2014). Such an approach can provide significant power to network managers regarding network traffic so that they can deploy numerous network devices as well as protocols. This enabled network efficiency to be considerably improved, ensuring that it was thoroughly utilising the allocated resources, while also implementing QoS, security, as well as other policies, consistently.

2.2 Software Defined Networking

A Software Defined Network (SDN) pertains to programming the networking devices as well as providing higher scalability through distinguishing the control plane and data forwarding plane. The control plane is responsible for controlling the network devices, including switches along with virtual switches-vSwitch concerning data forwarding. As noted by Marschke, Doyle, and Moyer (2015), the management plane is not only simpler than the traditional switch architecture, but also, more programmable and manageable owing to the architecture of SDN. This plane also effectively distinguishes between the control plane and the data plane within the routers and switches, as shown in Figure (2-2).

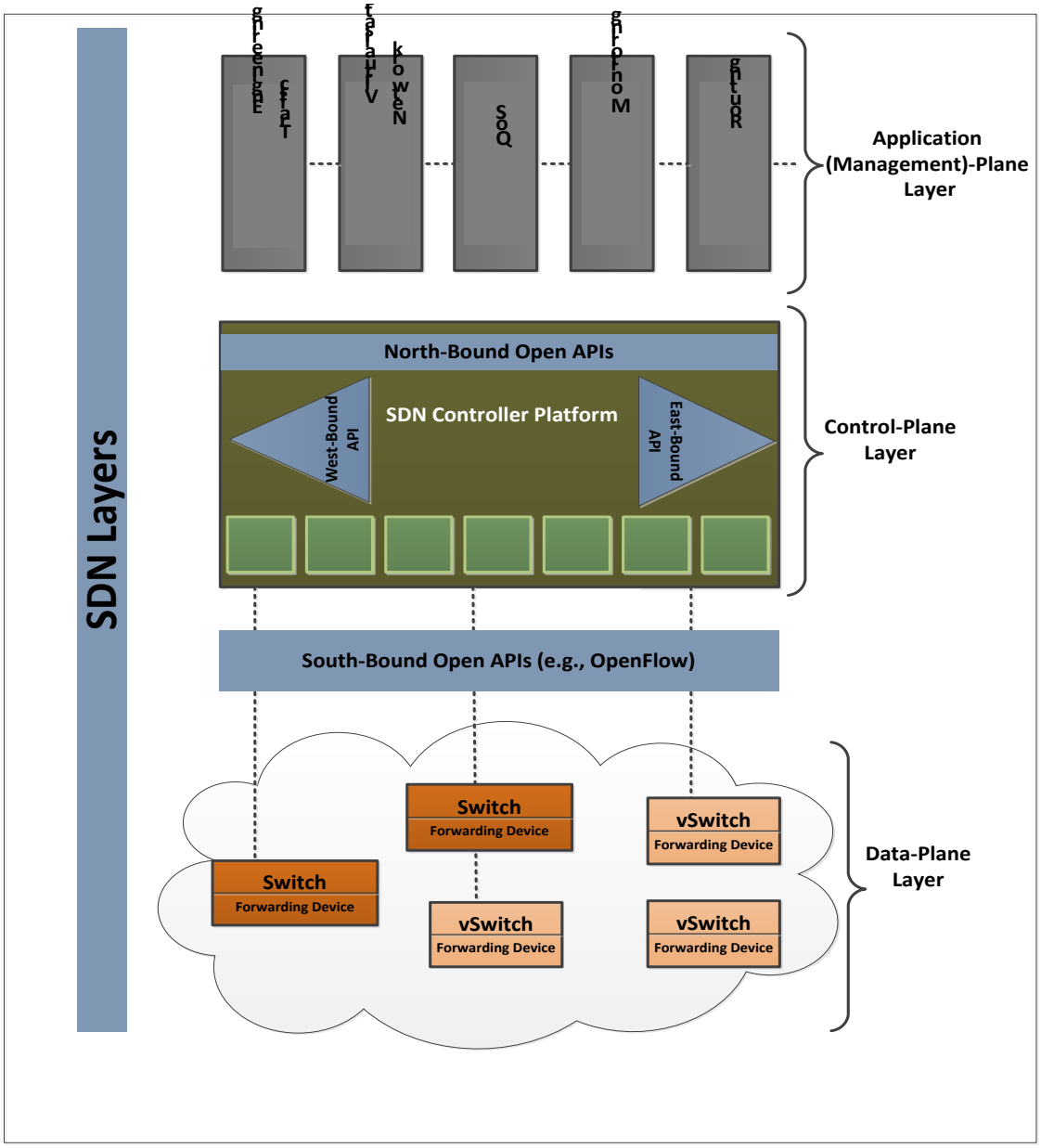


Figure (2-2) SDN architecture

Moreover, it is important to note that, in SDN, the new separation concept means that the control plane can reside outside the networking device and can be developed from one or multiple controllers, with the precise number being established as per the network size. Furthermore, this separation allows for the treatment of network protocols and services as software. Thehe data plane has the role of receiving information and requests from the control plane and implementing them in the hardware as needed (Rowshanrad et al., 2014). The control plane can implement its functionality as software andnetwork services can be considered now as being applications. Such differentiation has led to the development of new protocols as well as applications that are not vendor dependent. It also allows for various

middleboxes to be consolidated into the software control and enabling direct network virtualisation. As shown in Figure (2-3), within the data plane layer, the following aspects are involved in the simple forwarding hardware, as explained by Azodolmolky (2013). First, there is a flow table that includes flow entries containing match rules and actions that address active flows. Second, there is a transport layer protocol, which can conduct communication with a controller in a secure manner concerning new entries that are not presently included in the flow table.

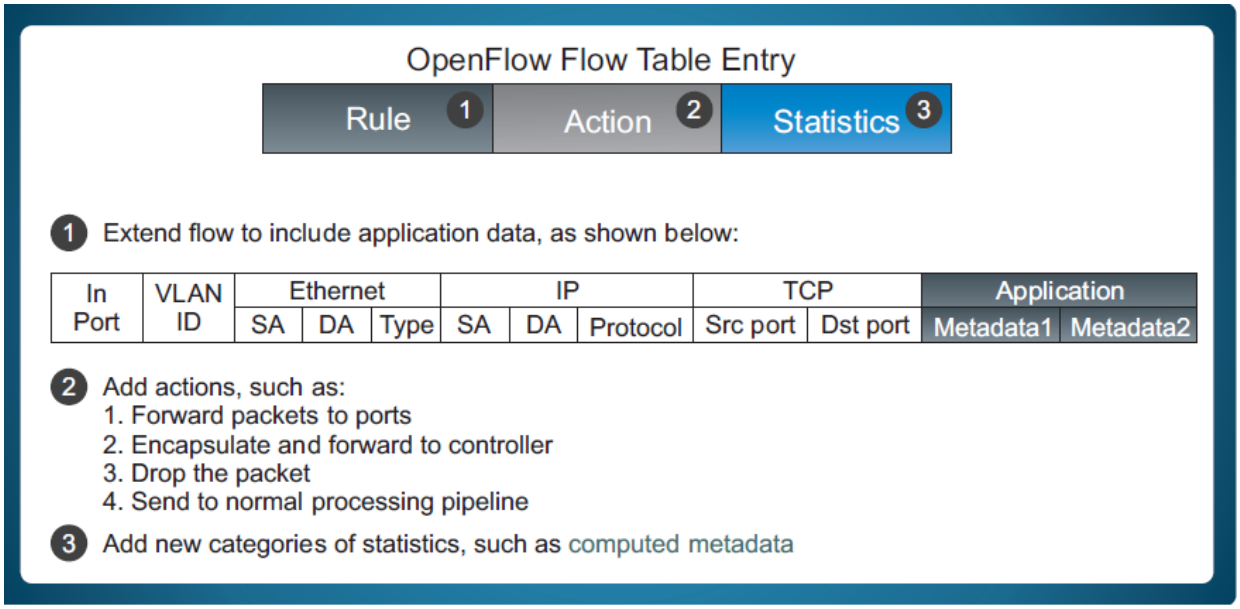


Figure (2-3) OpenFlow flow table components (Azodolmolky, 2013)

The terminology used in this study regarding the various SDN elements, (refer to Figure 2-2), with comprehensive explanations for each, is provided next.

1. The application (Management)-plane layer refers to a set of application programming interfaces (API), including northbound open APIs, which enable the application layer's communication with the control layer for helping common network services, such as traffic engineering, routing, security, multicasting, bandwidth management, access control, energy usage, QoS, as well as various network management methods. It also provides operational access along with monitoring using the graphical user interface (GUI) and functions including the Simple Network Management Protocol (SNMP), XML, SSH, the command line interface (CLI), and NetFlow. These use the controller plane interfaces, including REST API, for interacting and communicating with the SDN controllers (Akyildiz et al., 2014).

2. The Control-Plane layer represents the network intelligence. Through the controller, the information required by the data plane layer for accurately switching the PDUs is provided. It also guides the Routing Information Base (RIB), Multicast Routing Protocol (MRP), Protocol Independent Routing, the Configuration File, and Unicast Routing Protocols (URP) (Kreutz et al., 2014).

3. The Forwarding Devices refer to software-based or hardware-based data plane devices that conduct elementary operations. These devices include comprehensive instruction sets, such as flow rules that help in taking actions concerning the incoming packets, such as forward to specific ports, forward to the controller, drop, and rewrite some header. Southbound interfaces define these instructions, such as OpenFlow (OF) (McKeown et al., 2008), Protocol-Oblivious Forwarding (POF) (Song, 2013) and ForCES (Doria et al., 2010). SDN controllers use southbound protocols to install them in the forwarding devices.

4. The Data-Plane layer includes the ports or interfaces that receive as well as transmit Protocol Data Units (PDUs), physical and virtual switches, routers, QoS, layer 2 Drivers, a Forwarding Information Base (FIB), and access lists (Marschke, Doyle, and Moyer, 2015). This system provides a thorough yet simple perspective concerning the complete network, while enabling users to utilise universal modifications and does not require device-specific programming. It is also expected that the data plane switches will oversee the flow tables that the controllers regulate. However, passing information among themselves is their main responsibility and hence, in having to work with control processes, they are excessively overloaded.

5. South-Bound Open APIs concern the forwarding devices' set of instructions, which the southbound API interface defines and are included in this interface. The interface further defines the forwarding devices' communication protocol with the control plane elements, which stipulates the interaction between the control and data plane elements.

6. North-Bound Open APIs refer to the network operating system that is able to provide the application developers with an API that indicates a northbound interface, such as a common interface to develop applications. A northbound interface usually abstracts the low-level instruction sets that the southbound interfaces use for programming forwarding devices.

7. East-Bound/West-Bound Open APIs, such as HyperFlow, which is logically centralised but physically distributed, localises decision making to the individual controllers, thus minimising the control plane response time to data plane requests (Tootoonchian and Ganjali, 2010) and helping the various controllers exchange control information concerning the data plane flow (Refer to (Tootoonchian and Ganjali [2010] for more details).

2.3. SDN Reference Model

Academia as well as industry have taken a keen interest in SDN (Hu, Hao, and Bao, 2014), as they regard it to be a vital architecture for managing large-scale complex networks that need regular policing or reconfiguration. The University of Westminster, as noted by Gartner (2014), upgraded its network for enhancing efficiency as well as establishing foundations that will allow for future SDN implementation. Lancaster University also began using HP Networking for supporting its SDN research as well as an innovation programme concerning its School of Computing and Communications staff and students. There have been other SDN case studies, including one by a U.S. government agency, which observed a 99% decrease in the network provisioning times, with a 50% increase in monitoring networks and a 50% decrease in the associated capex by large industries, as well as logistic companies enhancing agility, enabling multitenancy, and decreasing opex. Moreover, one regional U.S. bank enhanced its security posture and decreased firewalling costs using SDN.

As mentioned previously, SDN decouples the forwarding functions and network control, because of which, the latter can be programmed directly, such as with the Forwarding & Control Element Separation (ForCES) framework (Doria et al., 2010) and OpenFlow (OF) (Lara, Kolasani, and Ramamurthy, 2013). The underlying infrastructure turns into, as shown in Figure 2-2, simple packet forwarding devices (the data plane) that are programmable. Whilst it is possible to use the SDN control plane as pure software that functions on industry-standard hardware, an SDN agent is necessary for the forwarding plane and this has to be used in specialised hardware. Moreover, based on the SDN networking element's capacity requirements and performance and on the necessity for specialised hardware transport interfaces, it is possible to use the forwarding plane on commodity servers (Taylor, 2014). VMware's NSX platform (Alto, 2015), for example, implements a virtual switch (vSwitch) as

well as controller, both using SDN protocols that do not need specialised hardware. As noted by Nunes et al. (2014), SDN can dramatically simplify network management as well as ensuring innovation and evolution.

The Open Network Foundation (ONF) (ONF, 2015) stated that SDN deals with the conventional networks' static architecture, this being unsuitable for the storage requirements, as well as providing dynamic computing current data centres, campus and carrier environments. Jarschel (2014) pointed out that the SDN architecture is programmable. Moreover, SDN ensures that it is possible to program network control directly as the control and forwarding functions are decoupled. Such programmability can help with automating network configuration so that network administrators are able to execute 'SDN apps' for enhancing specific services, including VoIP, through which phone calls can have a high QoE. When control is abstracted from forwarding, it allows administrators to modify the network-wide traffic flow dynamically as per the evolving requirements. Because of this, the network becomes more agile as software that functions on hardware having shorter release cycles compared to device firmware. Moreover, network intelligence is (logically) centralised within software-based SDN controllers, which ensures the network's global view, thus being perceived by applications as well as policy engines as a logical switch. For encouraging new network protocols as well as tools being developed, well-known SDN architectures follow open standards, including the OF protocol. Regarding the traditional network architecture, packets are involved in processing traffic and forwarding decisions are made by both network routers and switches addressing information within data packets. However, nowadays data tends to be sent as a flow between hosts and SDN utilises the flow as the basic unit to address traffic.

Existing networking paradigms, such as cloud storage (Mell, and Grance, 2011), have certain similarities with the SDN separation concept and hence, approximately 30 years ago, network operators could distinguish between the network operating architecture and the resource availability as well as time-consuming requests. As pointed out by Nadeau and Pan (2012), SDN's value in data centres and clouds is particularly related to the fact that it can provide innovative capabilities, such as automating resource provisioning, network virtualisation, and provisioned network resources represent data, information and hardware that are used in a shared connection as shared resources.

Network virtualisation decouples and isolates virtual networks from the underlying network hardware, whilsthe same physical infrastructure is also observed in the isolated networks. After virtualising is complete, the underlying physical network is only implemented to forward packets. Next, various independent virtual networks are overlaid on the present network hardware, providing the same features as well as ensuring a physical network that has virtual machine (VM) hardware independence and operating benefits (NICIRA, 2012). Network virtualisation benefits are that it:

- Provides enhanced application performance;
- Enhances flexibility and speed, generating SDN structures that are fully functioning becomes considerably easier by developing VMs;
- Dynamically maximises network asset use and decreases the operating requirements;
- Offers network management regarding DCs' interconnecting servers;
- Enables dynamic allocation of the cloud services.

Moreover, SDN ensures an interactive solution for using new capabilities, such as cloud applications and services retrieving network topology, determining and altering network connectivity/tunnelling, as well as overseeing the underlying network conditions, including failures. Despite the fact that the data centre interconnections enabling the current control plane protocols, including spanning tree protocol (STP), to be bypassed, the interconnection of heterogeneous data centres for developing federated clouds becomes a problem for SDN. There have been studies where the controller's northbound interfaces have been extended for developing customised real-time forwarding processes to form cloud plug-ins (e.g. Neutron OpenStack), which ensures bandwidth isolation as well as virtualised address space for every virtual link (Rotsos et al., 2012). It is also possible for third party applications to use the northbound interfaces for monitoring the SLA violation and altering the network resources.

However, SDN-based virtualisation leads to numerous major challenges, such as the performance of the data centre, including the need to address the rising processing and memory overhead, with resource provisioning including operators possibly overprovisioning network links for addressing probable network congestion. In addition, there can be packet drop in the data centres, which not only make congestion unpredictable, but also, expensive in several networking scenarios, QoS management such as various SDN applications which need northbound interfaces for communicating with third party applications that need the Service

Level Agreement (SLA) violation to be monitored for modifying the network resource, and slicing which includes network partitioning.

As SDN is becoming a major player in the evolution of network services, it is essential that a global solution must be offered for dealing with such problems. It is also possible that this architecture can present a modulator SDN layer for overseeing the communication among the applications, data centre infrastructure, and services (Nadeau and Pan, 2012). SDN can be implemented for reconfiguring resource or rapidly routing allocations using a cloud vendor. It also enables cloud users to receive greater value from cloud-based tools as well as resources and conducting tests and trials regarding virtual flow slices. These can be regarded as valid solutions in the case of all routers being used by companies following SDN requirements.

In the future, operators may have to face significant challenges regarding the provision of dynamic, adaptive, and convergent networks in a multi-services, multi-technology, and multi-protocols environment (Lezhepekov, Letenko, and Vladyko, 2017) There is also a higher probability of having to address major problems regarding the required network resources that are needed to provide differentiated as well as measurable QoS. An effective communications network can be considered the foundation for all successful organisations. Such networks provide diverse applications, such as real-time voice, delay-sensitive data, and high-definition video. It is also crucial that networks to provide measurable, predictable and guaranteed services through addressing the throughput, jitter, loss, and delay parameters (Shah et al., 2016).

With SDN, network managers can exert more influence regarding information movement. This, as noted by Tseng et al. (2018), enables them to change the features of routing as well as switching components in a network concerning a principal position of routing, which leads to random modifications becoming possible in routing channels. The managers can also have extra impact on the network information and are able to order objectives according to their urgency (ONF, 2014). Furthermore, they can decide to allow or deny particular data packets based on their intended aims. In the case of control applications being addressed by considering them as software modules, there is no need for a manager to decide for every component (ONF, 2016).

2.3.1 Network applications layer and management plane

As illustrated in Figure (2-4), an SDN architecture can be portrayed as a combination of various layers, with each having its own particular functions. Whilst some of them are always be present in an SDN deployment, such as the southbound API, northbound API, network applications, and network operating systems, others are only included in specific arrangements, such as hypervisor-based or language-based virtualisation. Figure 2-4 shows a tri-fold perspective of SDNs, with the layers illustrated in the middle of Figure (b). Moreover, Figures 2-4 (a) and 2-4 (c) present a plane-oriented view as well as an illustration of the system communication protocol.

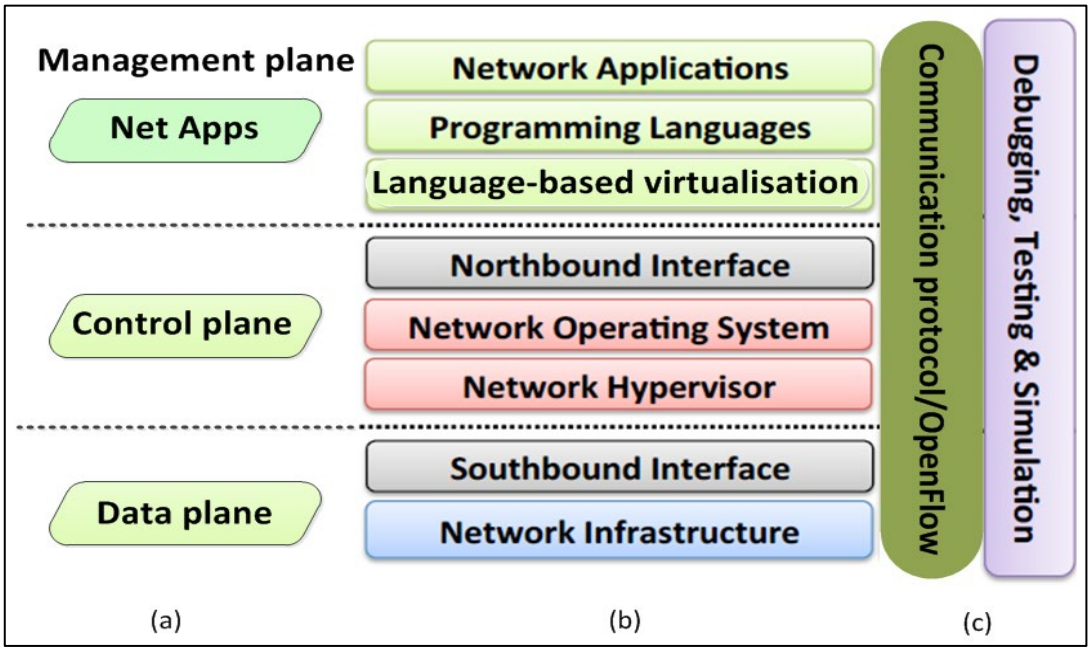


Figure (2-4) SDN in planes, layers and protocol

An SDN architecture's upper layer is the application layer, which establishes the rules as well as providing various services, including a firewall, access control, intrusion detection systems (IDS)/intrusion prevention systems (IPS), QoS, routing, network virtualisation and proxy service. Monitoring balancer explore novel approaches, such as reducing power consumption. It is a crucial layer for SDN and uses the northbound API to abstract the network control management. It is also responsible for regulating the software-based security as well as business requests. Some such areas where SDN has been implemented are network Functions

virtualisation (NFV), cloud solutions, information content networking (ICN), network virtualisation, and mobile networks.

Apowerful reason behind SDN being implemented extensively is its substantial diversity and versatility. Through combining it with reactive 'use case' allocations, network management's advantages become evident. Despite the fact that there are limitless possible networking conditions, most SDN applications can be classified as traffic engineering, mobility and wireless, data centre networking, measurement and monitoring, or security and reliability (Stallings, 2015).

2.3.2. North-bound open APIs

The northbound open API is considered a software API, rather than a protocol. Using the northbound interface ensures a stable and consistent method through which application developers and network operators can use SDN services as well as conduct vital network maintenance (Jarraya, Madi, and Debbabi, 2014). Further, the northbound open API also becomes compatible with diverse types of controllers in some cases as a thoroughly developed interface, which enables developers to create software separate from the data plane as well as the limitations of the specific SDN controller server's limitations.

The northbound open API also includes the relationships among controllers, user applications, and network applications or services. In addition, this layer is in charge of extracting the underlying network functions, such that network operators or developers are able to use or alter applications, while not be concerned with the low-level aspects. Kreutz et al. (2015) stated that the northbound open API obtains the low-level functions as outlines by the southbound open API for programming forwarding devices. Whilst the northbound open API does not have a widely accepted standard, there have been certain initiatives aimed at achieving this. The ONF, for example, has developed a working group to develop prototypes, codify certain patterns, as well as produce artefacts that can verify the development of a northbound open API standard (Kim and Feamster, 2013; Bakshi, 2013). Figures 2-2 and 2-3 present a

common view concerning the northbound and southbound layers. Such a perspective portrays the distinct responsibilities concerning every SDN architecture level. Hence, a northbound standard can help develop various applications as well as execute them in SDN controllers by any vendor by using well-defined APIs.

The northbound interface between the controller and the application plane shares information concerning the application needs, which enables the controller to open the most optimum software network. This includes acceptable QoS, adequate security, as well as the management required for the operations to be conducted without any issues. The Representative State Transfer (REST)-type API forms the essential protocol for such transmissions. This application interface should help with sending the information required to configure, measure, and operate (Pujolle, 2015). This protocol is based on REST and has been embedded in various cloud management systems, especially in the interfaces of the majority of such systems, including Open Source or VMware software, wherein the REST API is integrated, the API Virtual Private Cloud (VPC), and all service provider systems.

In addition, the representations provided by the REST protocol help with passing information to the northbound interface, including properties (Pujolle, 2015), such as each resource getting individually identified and the resources being modified by representations. The messages can themselves explain their nature and are thus self-descriptive, and the present message presents every access to the states of the application that follow.

2.3.3. Control plane and network operating system (NOS)

A set of SDN controllers is present in the control plane to offer control functionality through the Controller-Data Plane Interface (C-DPI) and to monitor the network forwarding behaviour. It also includes interfaces for ensuring communication among control plane controllers (Intermediate-Controller Plane Interface or I-CPI (Lin et al., 2015), optionally secured through the transport layer security (TLS), between network devices and controllers (C-DPI), as well as between applications and controllers (Application-Controller Plane Interface or A-CPI). An A-CPI, commonly known by the SDN community as the ‘northbound interface’, enables communication between the controller(s) and network applications/services with regard to network security or management. Further, there are two major components in a controller,

which are control logic and functional components. There are multiple functional components in controllers, including the coordinator and virtualiser, which help in managing controller behaviours. The SDN control level is responsible for converting the application level's service commands into precise and practical instructions as well as requests. The requests are then transferred to the data plane switches that provide applications with significant information concerning their conditions and behaviour. As stated earlier, the controller determines the optimal route, while ensuring that it functions effectively (Rothenberg et al., 2014).

The controller is central to the network as well as its intelligence, being the network operating system's (NOS) main contributor. An NOS offers fundamental services, common APIs, along with a lower-layer elements abstraction to developers. An SDN NOS has certain major functions, which help developers outline network policies as well as manage networks without having to be worried about the network device characteristics' details that could be dynamic and heterogeneous. The applications and network components are at opposite ends of the process and are connected by the controller. As stated by Rothenberg et al. (2014), the controller detects the optimal route and ensures its successful functioning. Figure 2-5 illustrates this process.

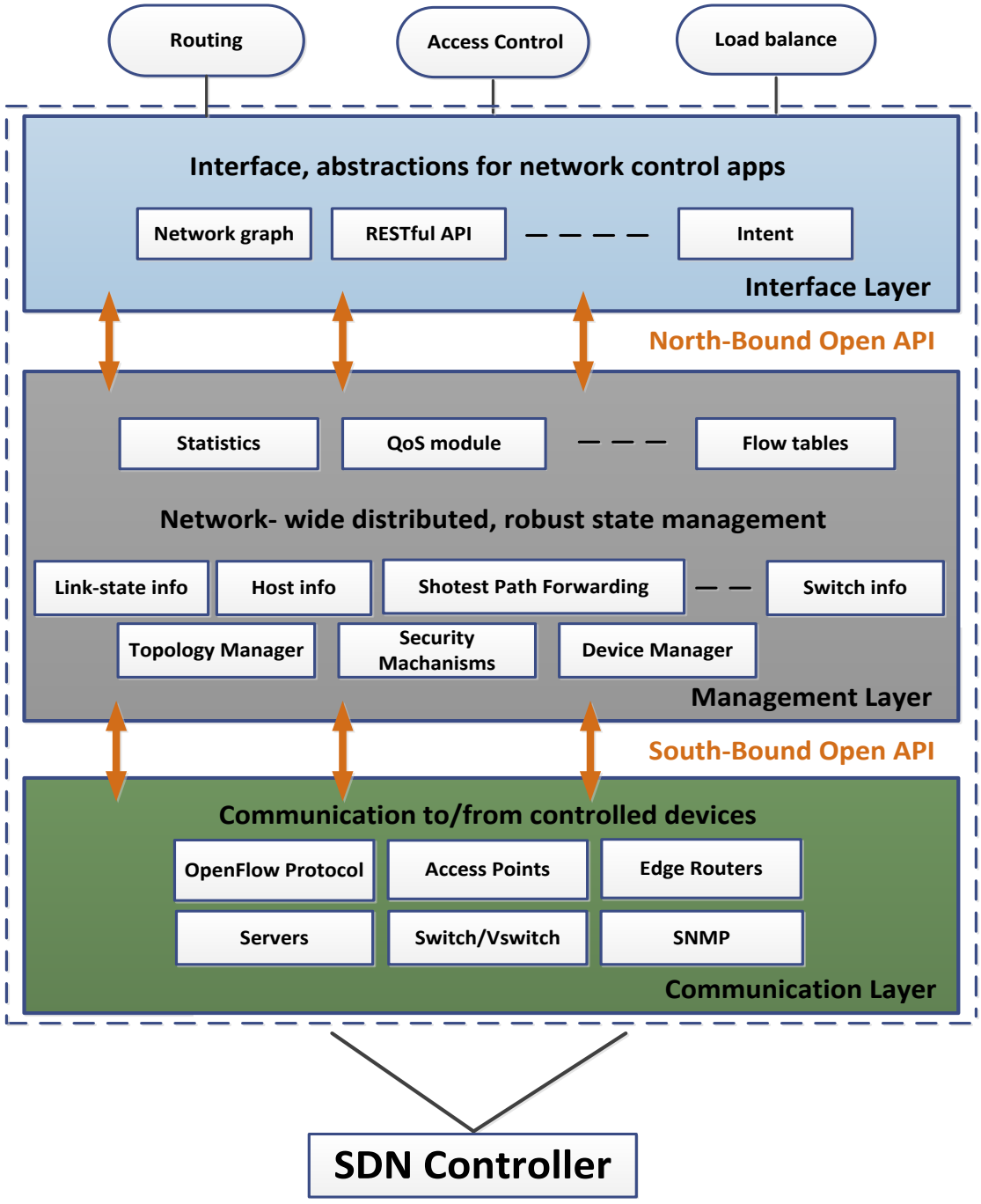


Figure (2-5) Components of an SDN controller

Figure 2-5 illustrates, in detail the major processes controllers are expected to provide, which are described below. While Figure 2-4 (page 26) shows a tri-fold perspective of SDNs, with the layers illustrated in the middle of Figure (b). Moreover, Figures 2-4 (a) and 2-4 (c) present a plane-oriented view as well as an illustration of the system communication protocol.

1. **Shortest path forwarding** involves the controller utilising routing data obtained gathered from switches for determining the quickest channel.
2. **Notification manager** involves the controller interpreting, reconfiguring, as well as transferring details regarding particular conditions to the application, which involve condition changes, security alerts, and other scheduled/pre-set alarms.
3. **Security mechanisms** involve the controller ensuring separation as well as security regarding applications and services in a safe format. Interactions that are common to the switches and controller are ensured safety by using security standards, such as a Secure Sockets Layer (SSL). Further, the forward level has to implement the security feature for binding clients with a Service Level Agreement.
4. **Topology manager** involves the controller developing and regulating data concerning switch interconnection topologies. For implementing the general update operations, every packet or flow is first detected, with the network being updated from an old policy to a new one throughout various switches, after which every packet or flow is handled either by the old or new policy, but not both (Reitblatt et al., 2011).

Consistency can be divided into two: first, there is per-packet consistency, which refers to the processing of every packet passing through the network occurring as per a single network configuration. The second is per-flow consistency, which pertains to the same policy addressing every packet in the same flow and thus, the per-flow abstraction maintains every path property. Such properties are depicted by a set of packets that are a part of the same flow passing through the network.

The statistics manager involves the controller utilising the switches for obtaining traffic information, while the device manager involves the controller altering the switch protocols as well as the features for enhancing flow table regulation. The QoS module refers to controller strength, in terms of whether it can identify the optimal QoS conditions and explicitly manipulate the flow table for differentiating between traffic objectives and priorities that are not similar. In addition, the abovementioned network structure should contain three elements, as described below.

1. **Supervision of Resources** – The forwarders should be capable of consistently overseeing the relevant network resources as regular status updates are required by the controller. The controller may also particularly request such updates.
2. **Signalling of Resources** – The forwarders should be able to use signalling alerts for interacting with the controller. Further, the controller requires data for conducting specific processes, such as upgrading the flow table or altering the QoS policies. The majority of vital information concerns the resource consumption rate.
3. **Allocating Resources** – There are certain circumstances in which the controller requests the forwarder to reserve certain resources so that they can be used in the future. Some well-known requests are allocating CPU calculation time, memory space, and buffer size (Stallings, 2015; Fabiano and Qiu, 2015).

The SDN controller's main role is regarded as a form of NOS. SDN NOS is primarily aimed at easing the process of developers formalising network protocols as well as conducting effective network maintenance, while not being concerned with the individual network devices' complex attributes. This is crucial, as such devices tend to have a rapid pace, are wide ranging, and are heterogeneous. Similar to traditional OS structures, as noted by Nadeau and Gray (2013), a NOS provides developers with low end components' abstraction, diverse main features and processes, as well as certain shared application programming interfaces.

Flows refer to a series of packets regarding every flow throughout its origin until destination which have certain characteristics in common. Initially, the OF specification v1.0-based switches (ONF, 2009) includes a single match table model that is usually developed on the Ternary Content Addressable Memory (TCAM). A flow in this OF concept can be determined through its packet header field, which is similar to a combination of a minimum of 10 tuples, such as ingress port, Ethernet, VLAN id, TCP header fields, and IP. Such aggregate fields are then placed in a single flow table in an OF switch's TCAM. To implement flow rules, the single table develops a major ruleset and can create a limited scale as well as inability concerning large scale deployments as TCAM space is an expensive and restricted resource Table 2-1.

Table (2-1) Overview of hardware memories for implementing flow tables on the OpenFlow switch. Derived from Akyildiz et al. (2014)

	DRAM	SRAM	TCAM
Purpose	Store for all data-plane functions	Store for data-plane functions	Search for data-plane logic
Cost	Low (Costs per Mega-byte)	High (Costs per Mega-byte)	Very High (Costs per Mega-bits)
Scalability (Storage)	Very Large (Millions of flows)	Large (Hundreds of thousand of flows)	Small (A couple of thousands of flows)
Speed (for packet matching and distribution)	Slow	High	Very high
Throughput (Depending on speed)	Slow (A dozen of GbE ports or a couple of 10GbE ports at line rate)	High (A couple of dozens of GbE ports at line rate)	Very High (48GbE + 4 × 10GbE ports or 48 × 10GbE + 4X40GbE ports at line rate)

TCAM-based tables tend to be restricted to a few thousand entries. On the other hand, one switch in the data centre is able to deal with more than 100 million packet flows per second. The adaptive and flexible flow table methods must, thus, be generated to ensure that the new flows that exceed the TCAMs' restricted space are replaced by lower cost DRAM or SRAM spaces, which are larger. Akyildiz et al. (2014) contended that such methods need to be related to a traffic scheduling method concerning various QoS flows.

Flow setup can be divided into three types: (a) proactive, (b) reactive, and (c) hybrid. Regarding reactive setups, the controller establishes flow commands, if the flow tables have no entries in them. Once the flow's initial packet is received at the OF switch, the instructions are immediately delivered. Hence, only this initial packet stimulates the controller's interaction with the switch. After a pre-established stagnation period gets completed, all flow entries turn obsolete and are erased. Whilst reactive setups are restricted because of a relatively slow journey, they are inherently able to use real time responses and accommodate traffic load along with meeting QoS demands. Applications that are naturally inclined towards the reactive model are those that have to respond to the development of new flows. Some of these applications, as noted by Goransson, Black. and Culver (2016), are per-user firewalling as well as security applications, which must determine and process new flows.

Proactive setups, in contrast, follow flow commands that pre-exist in the flow tables and thus, the setup is conducted prior to the OF switch receiving the initial packet. A major

advantage of this system is that it is not necessary to wait for the controller's response. It is, therefore, faster and the controller must be 'activated' only if a flow command is undetectable or in case of a problem. A disadvantage of this is that it may overload the flow tables, which are connected with the switches. The hybrid mode refers to the previous two modes' combination and helps in saving the number of flow entries (Akiyama et al., 2015). The pipeline processing function is used by the hybrid mode for standardising switch specification. The controller can use the hybrid mode that the administrator controls. Furthermore, the administrator is able to install particular flow entries within the switches proactively, reactively altering them by deleting or editing them and adding new flow entries according to the incoming traffic (Karakus and Durresi, 2017).

Whilst fine grained traffic movements, also called miniature or micro flows, tend to inherently be versatile, they cannot be applied to bigger networks. On the other hand, macro flows can be developed by acquiring numerous miniature flows. They can be regarded an appropriate option as using a rougher grained movement enables the controller to develop smaller compromises with less impact between adaptability and scalability. Further, the controller has to guarantee that the compatibility of the flow with the protocols that are connected with the application level to react to a flow setup command. After accomplishing this, further steps can be taken. It is also assumed that an optimal channel concerning the flow will be determined and innovative flow entries applied at each switch along the journey. This is also applicable to the switch that develops the command. Based on the flow entries that are used, a design is determined concerning the regulated granularity flow that leads to a compromise between adaptability and scalability. Moreover, Al-Haddad and Sanchez (2018) noted that the controller interprets the conditions necessary for effective network maintenance.

Table (2-2) presents a list of current controller applications and briefly describes them (Hu, Hao, and Bao, 2014). Every listed controller, except RYU, is compatible with the OF protocol's Version 1.2 (Nunes et al., 2014). A specific function controller application is also included, namely FlowVisor (Sherwood et al., 2010)

Table (2-2) Current open source controller implementations compliant with the OpenFlow standard

Controller	Origin & Programming Language	Description
Floodlight (Belter et al., 2014)	BigSwitch & written in Java	An open source OpenFlow controller, simple to use, build, maintain and run. Works with physical and virtual OpenFlow switches. (Raju, 2018)
FlowVisor (Sherwood et al., 2009)	Stanford University/ Nicira & written in C	Special purpose controller implementation that works as a transparent proxy in SDN OpenFlow. (Sherwood et al., 2009)
MUL (Yazici, Sunay and Ercan, 2014)	Kulcloud & written in C	Is easy to learn and implement, being a highly flexible C based OF controller that has a multi-threaded infrastructure at its core. (Raju, 2018)
IRIS (IRIS, 2016)	ETRI & written in Java	Is a recursive OpenFlow controller that aims to support scalability, high availability, and multi-domain support. (Masoudi and Ghaffari, 2016)
Trema (Carlos, Rothenberg and Maurício, 2010)	NEC & written in Ruby/C	A framework that provides a network emulator and libraries to enable developers to create OpenFlow controllers.
Beacon (Sezer et al., 2013)	Stanford University & written in Java	A controller that supports multiple platforms; Java-based, supporting multi-threads and event handling.

NOX (Fang et al., 2013)	Nicira & written in Python/C++	Wass the first explicit OpenFlow controller platform that provides a high-level programmatic interface for management solutions. (Raju, 2018, page 3)
Jaxon (Raghavendra, Lobo and Lee, 2012)	Independent Developers & written in Java	A NOX-dependent OpenFlow controller
POX (Metzler and Metzler, 2013)	Nicira & written in Python	An open-source SDN controller that uses OpenFlow or the OVSDB protocol for providing a framework for communicating with SDN switches.
OpenDaylight (Medved et al., 2014)	OpenDaylight Independent Developers & written in Java	This controller supports a modular controller framework, providing support for other SDN standards and forthcoming protocols (Raju, 2018, page 3)
Ryu (Jammal et al., 2014)	NTT, OSRG Group & written in Python	Open Source SDN Controller that is used to increase the flexibility of the network by making the task of handling the traffic easier" (Raju, 2018, page 4)
NOX-MT (Tootoonchian, 2012)	Nicira Networks and Big Switch Networks & written in C++	An SDN controller that aims to use multi-core technology and proposes multi-threaded controllers to improve their performance. (Jarraya, Madi, and Debbabi, 2014)

2.3.4. South-bound open APIs

Southbound interfaces refer to the APIs, which ensure that the control plane has strong communication with the data plane. It is also an interface which conceptualises the opposite of a northbound interface. Typically, the southbound interface is located towards an architectural diagram's bottom area. Some examples of southbound interfaces are I2RS, the Command-Line Interface (CLI), the Network Configuration Protocol (NETCONF) SNMP and OF, which are southbound APIs that are primarily applied for SDN implementation (Hakiri et al., 2014).

As noted by McKeown et al. (2008), the OF protocol is a widely used interface for SDN forwarders as well as controllers. The protocol refers to an SDN technology that can help standardise the method as well as define an API regarding the controller's communication with the switches. Further, it offers a specification so that the control logic can be transferred from a switch to the controller. Hence, the controller as well as switches must have a thorough understanding of the OF protocol. OF-based architectures include particular features that researchers can use for experimenting with innovative ideas as well as testing new applications. Some of these capabilities are software-based traffic analysis, dynamic updating of forwarding rules, flow abstraction, and centralised control. Moreover, OF-based applications are suggested for making network configuration easier, simplifying network management, adding security features, deploying mobile systems, and virtualising networks as well as virtualising data centres. According to Masoudi and Ghaffari (2016), OF is one example of several alternative and improved interfaces.

The development of faster and stronger interfaces not only increases the interoperability levels, but also, makes it easier for network managers to implement vendor agnostic components, because of innovative technologies, such as OF. The system is significantly beneficial and is among the most extensively used southbound interfaces in terms of developers who work with SDN. OF is one among several southbound interfaces that are compatible with SDN. Other API structures are the Hardware Abstraction Layer (HAL) (Parniewicz et al., 2014), Protocol-Oblivious Forwarding (POF) (Song, 2013), ForCES (Haleplidis et al., 2015), Programmable Abstraction of the Data Path (PAD) (Belter et al., 2014), OpenState (Bianchi et al., 2014), Revised Open Flow Library (ROFL) (Suñé et al., 2014), OpFlex (Salman et al., 2016) and the OpenvSwitch Database Management protocol

(OVSDB) (Davie et al., 2017). A major benefit of OF is that it provides a shared protocol to incorporate OF-oriented forwarding components as well as maintaining control's interactions with data plane constructs, such as switches and controllers.

The protocol includes three major data sources that are the methods for transmitting important information within the network. The first source includes the statistical information movement between the forwarding components and the controller. The second observes the delivery of the packet data by the forwarding components to the controller, if there is a lack of clarity regarding the handling of urgent flow or if a direct command exists to 'deliver to the controller' connected with the flow table entry. The third source includes the delivery of the event related data to the controller in the case of changes in conditions.

2.3.5. East-bound/west-bound open APIs

A key characteristic of SDN is centralised control over the network, but one controller can only deal with a restricted number of switches. Hence, distributed controllers have become necessary because of an exceptional rise in the network devices in large scale networks. Each controller in this type of distribution has its domain that has underlying forwarding devices. It is important for controllers to share information regarding their own domains to ensure a consistent global view regarding the network. Further, eastbound APIs help with importing and exporting information between the distributed controllers. One such interface is SDNi, which is a message exchange protocol introduced by Yin et al. (2012). Westbound APIs ensure that legacy network devices, such as routers, can communicate with the controllers. An example of this interface is BTSDN, as proposed by Lin, Bi, and Hu (2016).

2.3.6. Data plane and network infrastructure

The SDN data plane is also called the forwarding plane and includes network devices, such as access points, routers, and physical/virtual switches. Such devices can be accessed as well as programmed or managed using SDN controller(s) via interfaces. The interface further ensures that the OF switches are dynamically configured in a consolidated way. SDN is only able to utilise programmable switches, because routing control no longer be included in the devices. Such switches can communicate with the controller, generally through OF and they

include three major components: (a) flow tables, (b) secure channel, and (c) the OpenFlow protocol (Figure 2-6). In addition, devices can include one or more flow tables. As noted by Latif et al. (2019), the secure channel creates a link to the controller, while the OF protocol ensures communication with the external controller.

As shown in Figure 2-6, the flow table includes flow entries depicting the format of actions, counters, and match. Once the packet emerges, actions are taken, and decisions made based on the related table. In the OF component, a channel that passes through the flow tables' arrangement can inform the packets about their processing and transportation. Such actions are stopped once a match is determined in a flow table or when no compatible matches are found and thus, no feasible commands are identified. Rao (2014) pointed out that the command importance is decided according to the established structure of numbers in the row and table positions.

In addition, header matching for every packet is conducted, according to which, a specific action is taken, and the counter is updated. The control plane uses southbound interfaces to install instructions, which are then used by data plane devices for conducting numerous actions, such as forward to port, forward to controller, as well as drop. Once a packet is received by a switch, it matches its packet header to check in the flow tables. Next, if a flow entry is identified regarding this matching, action is taken, and the packet is passed on to a specific port. On the other hand, if no entry is found, the packet could be dropped or passed on to the external controller using a Packet IN message. As noted by Cox et al. (2017), the network's global view is considered for the controller to reply as a Packet OUT message as well as install flow entry regarding this packet.

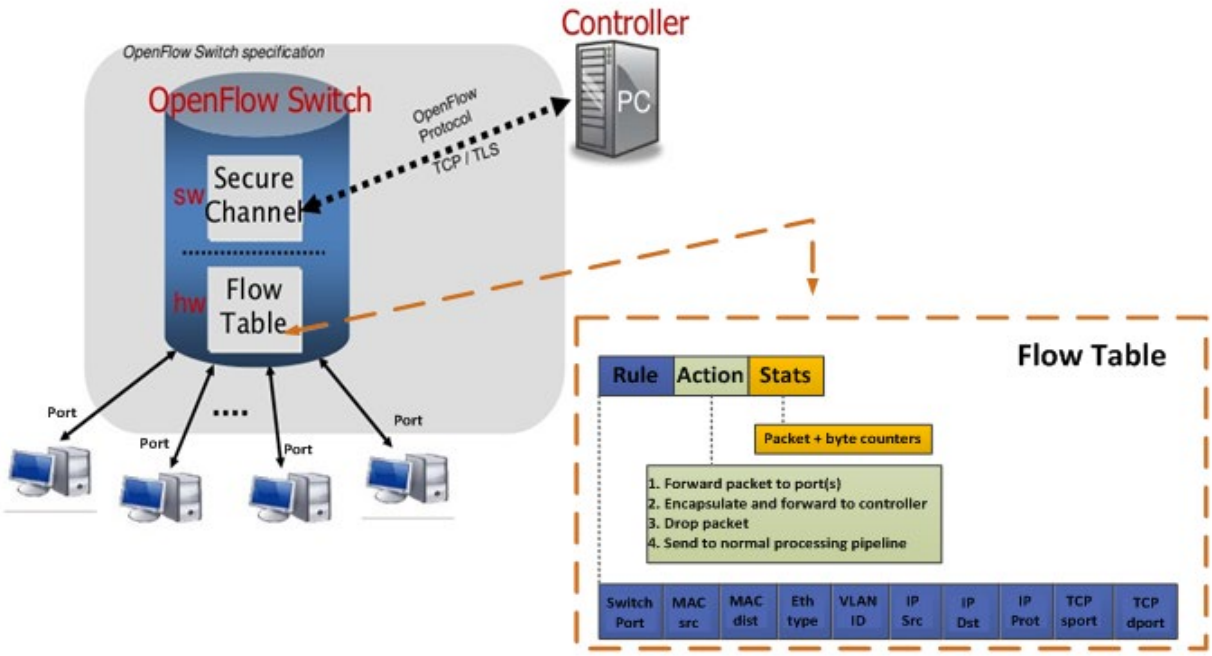


Figure (2-6) OpenFlow-enabled SDN devices and flow table entry for OpenFlow. Derived from McKeown et al. (2008)

The data plane component comprises a particular form of hardware or software resource that is developed to address packet forwarding, whilst controllers constitute a software arrangement type (or a 'network brain' as noted in an earlier analogy) that functions through a commodity hardware foundation. Through open interfaces, controller components can configure heterogeneous components with greater accuracy and at a faster pace. This is a significant advantage, as traditional networks often find this difficult, because of the extensive number of closed and proprietary interfaces and the advanced control plane distribution (Stallings, 2015).

As previously explained, OF was developed for various devices that only had data planes responding to commands that they received from a (logically) centralised controller containing that network's single control plane. The controller was in charge of maintaining the network paths and programming every network device that it controlled. The OF protocol outlines the description of the commands as well as their responses (Lara, Kolasani, and Ramamurthy, 2013). The vital components of an SDN network are the OpenFlow Controller and OpenFlow Logical Switch, which includes ports, the OpenFlow Secure communication Channel (SSL), the flow table with flow instructions, or flow entries, which guide the switch regarding handling of the packets, group table, meter table, and pipeline. The switches move the packets based on

the flow tables, being developed as per the controller's instructions, which can be altered according to the flow pattern changes. The flow table matches source as well as destination addresses to forward a data packet. The controller refers to a software system responsible for altering the switches' flow tables, for which, the OF principles and standards are implemented. A strong interface connects the switches with the controllers, while the packets move within the set parameters. Further, the interface aids in regulating the process packets and switches as well as ensuring the transport of the outbound packets to the correct location. The controller can interact with the switch only if the flow table sanctions it (Lara, Kolasani, and Ramamurthy, 2013). Based on an internal perspective, packets are interpreted by the switches using the ternary content dddressable memory (TCAM) and Random Access Memory (RAM). Those switches showing compatibility with OF must be able to forward packets according to the relevant flow table commands.

There is another benefit of OF, which is that it can lead to fast-paced changes and enhancement of forwarding protocols. It is important to note that human intervention is not required here and that the controller can change the flow table entries at any moment. Accurately configuring the system ensures that it is able to address diverse topological alterations rapidly. Furthermore, actions are taken based on judgements of the software controller, also called the network's 'brain'.

In 2009, OF Version 1.0 was presented to the market (ONF, 2009) and has become the most widely used interface for networks throughout the world. After, OF Version 1.1 was presented in early 2011 (Refer to Table (2-3)) and an updated system known as Version 1.2 was introduced in late 2011 (Jarraya, Madi, and Debbabi, 2014). This new version enables the switches to connect multiple controllers at a given time. This leads to a stronger system with fewer fault risks as well as enhanced load balancing. It also offers additional support for resources, such as extensible matches, IPv6, and type-length-value TLV functions. Introducing TLV enhanced the OF's overall versatility as it means that the fundamental protocols are not restricted by their arrangement and size anymore. Table (2-3) shows details of other enhancements of OpenFlow versions specifications; however, Version 1.0, 1.1, 1.2 is the focus right now.

Table (2-3) OpenFlow version enhancements

OF versions and release date	OF protocol specifications and features
OpenFlow 1.0 release date Dec 2009 (ONF, 2009).	Single table Fixed 12 tuple match field
OpenFLow 1.1 release date Feb 2011 (Specification, O.S., 2011).	Multi-Table Group table Full VLAN and MPLS support Support for virtual ports and tunnels Multipath routing
OpenFlow 1.2 release date Dec 2011 (Jarraya, Madi, and Debbabi, 2014).	First ONF release-fix1.1 More flexibility Flexible match Flexible rewrite IPv6 Role change
OpenFlow 1.3 release date April 2012 (ONF, 2012).	Long term release:1.3.1, 1.3.2, 1.3.3 Flexible capabilities Meters PBB Event filters
OpenFlow 1.4 release date August 2013 (Feamster, Rexford, and Zegura, 2014; (Ren and Xu, 2014)	Synchronised Table Bundles Optical ports Flow monitoring Eviction OF Version 1.4 includes an 'eviction' function at, in case of the table becoming full enough to quire immediate action, enables a switch to

	nore and eliminate entries with lesser urgency. Default Port to 6653
OpenFlow 1.5 release date Jan 2015 (ONF, 2015).	Egress Table Packet Type Aware Pipeline

For the present thesis, in terms of the recommended system, Floodlight controllers will be utilised along with OF Version 1.2, as most SDN controllers, except RYU, show compatibility with this interface (Harkal and Deshmukh, 2016).

2.4 Network Virtualisation

As noted by Lippis III (2007), SDN is important in enterprise networking, particularly because it can provide network virtualisation as well as automated configuration throughout the network fabric. It also delivers rapid deployment of new services and end systems, along with reducing operating cost. Moreover, according to Feamster, Gao, and Rexford (2007), network virtualisation decouples as well as isolates virtual networks from the underlying network hardware. The same physical infrastructure is also observed in the isolated networks. After virtualising is complete, the underlying physical network is only implemented to forward packets. Next, various independent virtual networks are overlain on the present network hardware, providing the same features as well as ensuring a physical network that has the virtual machines' (VMs) hardware independence and operating benefits (NICIRA, 2012). For ensuring next level virtualisation, a network virtualisation platform (NVP) is required so that the physical network components can be transformed into an all-encompassing pool of network capacity, akin to the way in which a server hypervisor changes the physical servers into a pool of compute capacity. In addition, by decoupling virtual networks from physical hardware, the network capacity is allowed to scale with no impact on the virtual network operations (NICIRA, 2012; Lippis III, 2007).

Network virtualisation regards the network hardware as a business platform that can provide various IT services along with corporate value (Turner and Taylor, 2005). Moreover, as noted by Duan, Yan, and Vasilakos (2012), it provides enhanced application performance, as it can dynamically maximise network asset use and decrease operating requirements. SDN has

resulted in network virtualisation becoming involved with cloud computing applications. Further, NV offers network management regarding DCs' interconnecting servers. It also enables dynamic allocation of cloud services while, extending DC limits to reach network infrastructure (Wen, Tiwary and Le-Ngoc, 2013; Jammal et al., 2014).

2.4.1 Machine virtualisation technique

Lantz, Heller, and McKeown (2010) pointed out that processes that follow virtualisation techniques are utilised for the virtualised data plane's first implementation. Mininet refers to the networking virtualisation environment founded on Linux VMs, which function on standard platforms, including Xen, VirtualBox, and VMware. It also enables virtual networks to be developed by establishing as well as interconnecting the network namespace's host processes. Moreover, Virtual Ethernet (veth) is implemented for host interconnection. To implement an SDN-based virtual network, the user may utilise diverse building elements: (a) links formed as veth pairs, (b) hosts acknowledged to be shell processes, (c) switches used as OF software switches, and (d) controllers, which can be executed anywhere in a real as well as simulated network. Ahmed, Huici, and Jahanpanah (2012) put forth the notion of increasing the number of click routers being executed in miniature VMs to scale the click modular router performance. Such miniature VMs are known as ClickOS, and they run on a Xen hypervisor. The Xen hypervisor involves a shared network netback driver that can communicate with hardware as well as use shared memory with the ClickOS netfront driver. The netback driver is responsible for forwarding packets between the network adapter and shared memory across a virtual network interface. The netfront driver; schedules packets to the click router transceiving interfaces from shared memory (FromClickOS and ToClickOS), and vice versa. This can help fill the gap between the click router and NICs, whilst also maintaining all virtualisation gains.

It should be noted that a network/NIC virtualisation technique can help with implementing generic high throughput or connecting VMs in a concrete case. Accordingly, Rizzo (2012) put forth a virtual local ethernet (VALE), which utilises the netmap API for communication and exposes ports to processes and hypervisors. The netmap is fundamentally founded on shared memory that indicates the interface between the network's hardware interface with packet processing applications. In this memory, reception buffers as well as packet transmission are designated to every network interface and to circular arrays known as

netmap rings to store metadata concerning reception buffers and transmission. The software switch mSwitch (Honda et al., 2015) also responds simultaneously to numerous drawbacks observed in the earlier solutions through using techniques concerning network interface virtualisation as well as distinguishing between switching and packet processing. In addition, mSwitch provides a flexible data plane, high throughput, high packet processing intensity, efficient processor utilisation, and high port density. The proposed architecture's core concept is classifying the data plane into switch logic, which is the switch's modular part that processes packets, and the switch fabric that is in charge of packet switching. This enables a high throughput and sustains a high level of packet processing functions' programmability.

Open vSwitch (OVS) is a multilayer virtual switch that Pfaff et al. (2015) introduced, which is aimed at networking within virtual production environments as well as supporting the majority of hypervisor platforms. Packet forwarding in the OVS architecture is conducted through two components. The first, is the user space's ovs-vswitchd daemon that is the same for all operating systems, whilst the second is the high-performance datapath kernel module, which was written particularly for the target operating system. Moreover, the datapath kernel module is in charge of receiving packets from a VM or an NIC and conducts processes as per the ovs-vswitchd module's instructions. If no established processing rules exist concerning the particular kernel module packet, it is passed on to the ovs-vswitchd module that takes a decision regarding further processing, after which it returns it along with the packet. As stated by Kaljic et al. (2019), once the OVS is implemented as an SDN switch, the OF protocols' agent side functions inside the ovs-vswitchd module.

Virtualisation results in the reality becoming the incorporation of comprehensive automation as well as centralised functions occurring at the SDN's core, which effectively enhances productivity, reliability, scalability, and integrity. A major benefit of SDN is that network virtualisation creates a coherent framework throughout each piece of related software, including the switching and routing elements. This leads to the alteration of processes and features at a rapid pace, while maintaining low costs. Further, the network manager does not have to determine a method for gaining access to the service layer, only for quick modification, thus making network use significantly easier (Kaljic et al., 2019). Moreover, as virtualisation enhances flexibility and speed, generating SDN structures that are fully functioning becomes considerably easier by developing VMs (Stallings, 2015). Furthermore,

combining physical and virtual systems results in enhanced versatility, reliability, and scalability. SDN also ensures that network managers can establish multiple networks and not have to change individual networking components manually, according to the altering conditions (Rothenberg, Vidal, and Salvador, 2014; Hoske, 2015). The most optimum quality control strategies are, indeed, capable of assigning bandwidth to move data traffic to the accurate locations. In terms of the present recommended system of this thesis, OVS will be implemented along with OF version 1.2 on VirtualBox standard platforms as well as the networking virtualisation environment, Mininet.

2.5. SDN Slicing Mechanisms

Network virtualisation was first used in the 1990s, such as in the Tempest project (Van der Merwe et al., 1998). Regarding the modern computer network platforms, the research community has used network visualisation as a consolidated technology for introducing virtual resources or an abstract layer to ensure that the same hardware resources are shared, resulting in significant cost reduction and enhanced flexibility compared to physical network equipment. For this, it is vital to use virtual machines, which are software implemented with a completely independent operating system (Kreutz et al., 2015; Hill et al., 2012).

Sherwood et al. (2009) at Stanford University introduced a network virtualisation layer known as 'FlowVisor' for slicing the network and suggested setting up a hypervisor between a PC's hardware and software. Moreover, FlowVisor implements OF as a hardware abstraction layer that is logically placed between network devices' control and forwarding paths. This results in virtual slices in wireless as well as wired networks, thus providing the controller with transparency and assigning control of every slice. It, thus, enables numerous OF controllers to include physical switches that ensure isolation among slices along with a changeable slice policy (Blenk et al., 2015).

As stated earlier, FlowVisor adds a new mechanism functioning as a software slicing layer between the network devices' forwarding and control planes (Sherwood et al., 2010). Thus, FlowVisor's major contributions include the possibility of slicing any control plane message format used in OF as well as being the first slicing mechanism that enables user-defined control plane control over the deployed production hardware's forwarding. The slicing

in network resources occurs as per the bandwidth, forward table entries, topology, as well as the device's CPU. A policy language traces the flows to slices, which provides flexibility to the user for attempting new services that can help them determine the extent of their participation in an experiment. The network users can also send a signal to the network administrator for adding (opt-in) or removing (opt-out) their flows randomly from the flowspace of a slice. This ensures transparency concerning data as well as the control planes, while also providing ease of communication between the sliced and traditional networks. It further helps in blocking and rewriting control messages, if they encounter the slicing layer. Hence, it ensures that slices have strong isolation among them, which enables researchers to conduct their experiments safely and not impact on real production traffic. Moreover, as noted by Sherwood et al. (2010), it has been able to operate efficiently on Stanford University's deployed networks with more than 40 network devices, more than 20 users, four standing experimental slices, and a production traffic slice.

FlowVisor's internal operation as well as the controller's communication with the switch are presented in Figure 2-7. The operation begins with the controller sending the command to the OF switch. These commands are initially received by a slicer element (1), which is in charge of overseeing the commands as well as messages sent to or received by the OF controller. Every virtual network controller has one slicer. Next, the slicer applies flow space (the route traffic of the controlled slices defining flow space) rules to confirm whether the command that is received adheres to the virtual network definition (2) and then, it makes changes to the command, if required. Following this, the output command is passed on to the OF switch (3) through the corresponding classifier element, which manages the commands as well as messages sent to or received by the OF switch, with each including one classifier.

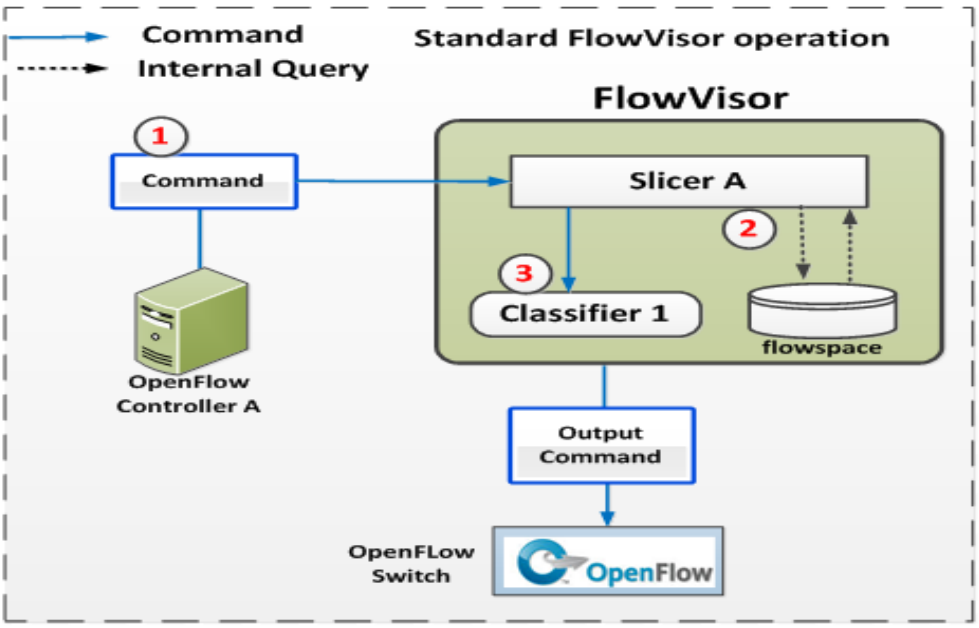


Figure (2-7) The current internal operation of FlowVisor (Costa and Costa, 2015)

2.6. Summary

In this chapter, the SDN reference model has been comprehensively examined in terms of diverse conditions and scenarios, along with the SDN slicing mechanisms. The research challenges have been identified and recommendations for future studies have accordingly been made. Next, Chapter 3 addresses some of the most effective methods for overcoming an SDN's limitations in terms of networks performance and its QoS metrics. A systematic evaluation is performed based on QoS parameters for network-centring multimedia delivery approaches and a thorough explanation of the solutions proposed in this field. Also, additional sections explore SDN applications implementation, such as traffic engineering (TE) mechanisms, evaluation of approaches to network performance in Software-Defined Networks, and QoS mechanisms, among others. To enhance QoS and to define a variety of controls for congestion, traffic management systems, admission control, and policy rules, a classification of virtualisation and slicing techniques and their effect will be provided.

3. Chapter Three: Literature Review and Research Challenges for Assessing SDN Network Performance and Quality of Service (QoS)

3.1. Introduction

The primary purpose of this chapter is to consider some of the most effective methods to get around SDN limitations in terms of network performance and QoS metrics. A systematic evaluation is performed based on QoS parameters for network-centring multimedia delivery approaches and a thorough explanation of the solutions proposed in this field is provided. Also, further sections explore SDN applications implementation, such as traffic Engineering (TE) mechanisms, performance parameters used in traditional networks and SDNs, and QoS mechanisms, among others. To enhance QoS and to define a variety of controls for congestion, traffic management systems, admission control, and policy rules, a classification of virtualisation and slicing techniques and their effects is provided.

3.2. Applications Implementation of QoS and TE in Software-Defined Networks

Through the SDN paradigm, numerous network administrators are able to oversee different network flows using their own applications and services. The following sections review specific services, including QoS and traffic engineering as well as how these are implemented in SDN. They also describe how SDN is able to surpass problems present in traditional networks for every service.

3.2.1. Quality of service (QoS) support in OF

Ensuring QoS has been a significant persistent problem in traditional networks. It has also led to operating expenses as well as risk, because of delivering diverse types of network quality to network end-users (Jarraya, Madi, and Debbabi, 2014). This may have occurred because of the networking equipment's closed interface, the Internet's absolutely distributed hop-by-hop routing architecture, as well as the lack of explanation regarding the overall resources and network. Despite this, the conventional practice and commercial systems, such as Cisco (Szilágyi and Almási, 2012), in general, provide an adequate QoS overall, taking all different types of traffic into account, by implementing higher priority for specific traffic type flows, it is still there is a scalability issue and congestion in these systems.

Theoretically speaking, in various research studies, different congestion-management methods have been used, such as priority queueing (PQ), custom queuing (CQ), weighted fair queueing (WFQ), and class-based weighted fair queueing (CBWFQ), in traditional commercial systems to manage the delivery of packets when there is more bandwidth than the link can handle (Thakur and Shukla, 2013). However, all these methods need re-evaluation and validation regarding the new expanded network systems that are used now and for the new technologies, such as SDN.

It is possible for SDN to address various network QoS issues by providing complete network visibility for collecting and analysing flows of traffic and to ensure that networking devices are programmable. A few papers have examined this subject and there has been some work going on to improve QoS support in OF. Egilmez et al. (2012) suggested OpenQoS, a dynamic QoS routing mechanism, designed particularly to deliver multimedia traffic with high QoS. The Floodlight controller used OpenQoS. However, their research did not show support for traffic shaping. Along with preliminary research initiatives (Sonkoly et al., 2012), the Open Networking Foundation (ONF) (ONF, 2015) has been working actively to improve OF's QoS support. Traditionally, OF Version 0.8 has only included experimental QoS support. Moreover, in OF Version 1.0 (ONF, 2009), it was possible to forward packets to output ports' queues through the optional enqueue action, which was called 'set queue' in Version 1.3 (ONF, 2012). Hence, the queue's behaviour was established beyond the OF scope. Further, OF Version 1.3 added complicated QoS support with meter tables, which included entries that

defined per-flow meters. The meter is a packet indicator that measures the arriving packet rate, which also contains meter bands and meter counters that help the OF to use different simple QoS operations, including rate-limiting. Such meters can also be merged with the optional set queue action that connects a packet and a per-port queue for applying complex QoS frameworks, including DiffServ (Blake et al., 1998) (ONF, 2012). When the present study was being conducted, some researchers had already examined this topic (i.e. complex QoS frameworks), thus making it imperative to conduct in-depth analysis of their work.

3.2.2. Traffic engineering (TE)

The traffic engineering (TE) or traffic management method helps with dynamically assessing, regulating, as well as foreseeing the behaviour of data that flow in networks to enhance performance and fulfil service-level agreements (SLAs). Regarding traditional networks, traffic engineering continues to be a burdensome task as it is founded on implementing infrastructures that are significantly expensive (Jarraya, Madi, and Debbabi, 2014). SDN not only ensures complete visibility regarding the network-wide view, but also, enables external programming of network devices through the forwarding tables' dynamic provisioning. Because of this, there is an option to select the optimally effective path, according to real-time information and application requirements. Google has already implemented centralised traffic engineering in wide area networks (WANs) through SDN (Jain et al., 2013). Wang, Ng, and Shaikh (2012) aimed to achieve traffic engineering using SDN by first examining an SDN-based integrated network control architecture to configure and enhance the network for ensuring bandwidth for big data applications requirements through optical circuits that are dynamically configurable. On the other hand, it is suggested to implement flow-level traffic engineering in big data applications in future studies. Das et al. (2011) examined the SDN/OF's capabilities concerning WAN traffic engineering and showcased them using a network application. TE strategies can be used to address the unwanted inactivity problem and improve flow maintenance as well as efficiency. If such a situation occurs, it is important to organise TE so that it can reduce the load balance and dynamic bandwidth allocation (Akyildiz et al., 2014).

In SDN, TE is one of the most popular network applications for tracking network traffic quantification and maintenance. It is also used to determine the best channel processes and to guide traffic better, such that key resources can be maximised (Sahay, Meng and Jensen,

2019). By and large, the aim is to enhance the QoS by helping to reduce the effects of traffic congestion and overflow and to help the network meet customer demands more easily. When evaluated in comparison to traditional systems, it is clear that SDN is a substantial improvement. To begin with, the fundamental concepts of ***flow management*** within SDNs dictate that the following four-step procedure is followed once a flow enters a switch and no matching commands are found:

1. The ingress switch first delivers the flow packet to the controller;
2. The proper forwarding channel for the flow is decided upon;
3. The controller sends the flow through the appropriate channel to the correct flow tables;
4. All other packets connected with the flow (sometimes from different flows with matching commands) are passed through the data plane without any further contribution from the control plane.

It is worth noting that it may take some time to set up forwarding protocols, so some inactivity will be likely to happen. In the event of a large number of new flows in the cumulative traffic, a major overhead may be produced via the data and the control planes.

By using TE methods to increase the maintenance and performance of the flow, the unwanted inaction issue can be solved. TE must be designed to reduce the tension between the load balance and the dynamism (Akyildiz, et al. 2014). SDN should also be able to conduct failure recovery processes in a simple, transparent, and competent manner to guarantee maximum network integrity. The general rule is that node and connection faults within a fifty-second window should be reacted to and resolved. OF Version 1.1 has allowed for the introduction of a rapid reaction system for node and connection failures, thus improving the overall networking integrity. The switch enables the assignment of a replacement channel so that the transmission route can be modified without first consulting with the controller. However, while changes have been made with the advent of centralised methods, getting adequate reaction times for faults remains a major challenge. To do this, the central controller must define alternate channels and warn all of the required fault switches before any corrective action is undertaken, which takes time. When then an error occurs (whether connected with connections, controls, switches, or something else), the failure

response standard must be extremely high. Not only this, but even all failure reactions should take into account flow table and memory constraints within switches.

The ***topology update*** procedures in the SDN concentrate mainly on expected shifts and improvements (such as minor network protocols and regulatory adjustments). They are much better at managing shifts in situations they are able to plan for, but fault tolerance does not help. With centralised controllers managing each switch of an SDN / OF network, a fair degree of stability and predictability are essential to all of them, through quick adjustment of all universal network principles. This allows the different packages and flows, either by the first or modified rules, to be controlled, whilst also ensuring that both do not overlap and attempt simultaneously to affect the same entity. There is a chance of more flows being ignored as laws and regulations are updated and strengthened. Others are likely to be interrupted and at a later point than expected, which is potentially detrimental to the network's credibility and can lead to poor use of resources. Ideally, the fault response must be conducted in the nearest possible real-time for the device to work and in particular with a significantly big SDN / OF system, this is not easy to achieve. Frequently, some switches cannot be connected directly to the main controller, which is a difficult limitation. This enables the SDN controller to change the network without jeopardising consistency, reliability, productivity or integration, as a major barrier to topology upgrades. Finally, both ***traffic assessment and network monitoring tools*** require acceptable traffic and reporting methods, invariant testing techniques, trend/attribute analysis/extraction, flow/state data collection and debugging functions, amongst other things. Traffic management and recording processes are some of the most critical components of a wider traffic assessment. It is a perfect way that links and network defects are identified and anticipate connection overloads and pile-ups (Mendiola et al., 2016).

Shu et al. (2016) implemented an SDN traffic engineering system consisting of two parts: (a) calculation of traffic and (b) control of traffic. Traffic measurement primarily involves real-time, network status monitoring and analysis for traffic management. Parameters for network calculation include network end-to-end latency, topology link status, bandwidth usage, throughput, packet loss rates, etc. The state of the actual network depends on the QoS parameters. Network traffic may be planned to satisfy user requirements in practice based on traffic calculation knowledge and various QoS parameters.

A framework for offline planning and online traffic routing in SDN networks was proposed by Cao, Kodialam, and Lakshman (2014). The mechanism's SDN controller conducts policy analysis, flow management and route creation. The policy table on the controller specifies the logical traffic sequence. Specifically, the controller maps the physical topology to the logical sequence of middleboxes and deploys the rules in the switch flow table, once the logical physical route has been mapped. The authors proposed an algorithm to maximise resources by boosting the ability of the middleboxes to handle the expected traffic. Also, an online traffic management system is implemented to direct flows upon arrival, in compliance with the policy requirements. For example, incoming traffic from the outside network may be needed to cross the firewall, intrusion detection system (IDS) and network address translator (NAT) on a storage server of the company's network. The required bandwidth for online traffic management is also considered.

Agarwal, Kodialam & Lakshman (2013) suggested an SDN-based mechanism of traffic engineering by leveraging its features, such as logically centralised controllers and the global network view. The objective of this system is to control the network traffic dynamically according to network administrator requirements. To generate the route for traffic management according to optimisation criteria, increase the bandwidth usage and reduce packet losses and latencies, the authors developed a Fully Polynomial Time Approximation Scheme (FPTAS), which takes into account the limited use of SDN switches to move traffic in the network.

3.3. Evaluation of Approaches to Network Performance in Software-Defined Networks

QoS illustrates the network's ability to deliver improved services to selected traffic across a range of IP, LAN and WAN technologies. It points out a network's potential to provide greater service levels over many technologies to chosen network traffic. There are several factors involved in network performance measurement rules that affect QoS, including: bandwidth, network congestion, latency (delay), packet delay variation (PDV)/jitter, route flapping, and error rate. In this study, the congestion of the network, latency, jitter and delay are taken into consideration.

Many researchers have analysed the protocols available and suggested new solutions for the use of traditional network protocols in combination with the latest technologies, such as, virtualisation, SDN's latest paradigms and slicing mechanisms, to support a broad range of applications, such as voice, video and file transfers. A well-defined QoS mechanism is required to address certain network application requirements. However, today's de-facto model of business is not able to support all the above services (i.e. TE and QoS). Assessment of the network performance is also reflected in its capacity and speed to accommodate many simultaneous transactions that involve the transfer of massive data quantities. However, there are also fundamental limitations in SDN. For example, the level of monitoring may, according to Gelberger, Yemini and Giladi (2013), lead to some kind of network bottleneck and to a major disruption of network traffic flows, which is a particular risk to big business using one control unit (Mirchev, 2015). There is a negative impact on performance and control levels. Hence, much research has been geared towards analysing the network efficiency matrix to find solutions.

3.3.1. Bandwidth analysis in SDN

The number of bits that can be sent over the network for a period (Peterson and Davie, 2007) illustrates a network's bandwidth. Effective time network application requires enough bandwidth to provide the end-user with steady network throughput. Specifically, bandwidth refers to the volume of data that an online connection can handle per unit of time. The performance of real-time traffic, such as voice and video, may be affected by a bandwidth bottleneck. Low network throughput results in insufficient bandwidth (Jimson, Nisar and bin Ahmad Hijazi, 2017).

To obtain statistical data to measure traffic flows in SDN networks, Fernández, Villalba, and Kim, (2018) proposed an improved algorithm. The grouping of network switches into clusters is the basis of this algorithm, which focuses on their various ports for different monitoring techniques. Their algorithm aiming to avoid redundancy controlling monitoring queries to OpenFlow switches, especially with common features of characteristics. This reduces the number of OF switches monitoring queries, improves network traffic as well as preventing overload switching and overuse of bandwidth. The data rate is calculated as the difference in the monitoring interval between sending bytes. The

study used the Mininet tool to test video streaming simulation optimisation employing various video types. The framework was based on experiments and comparison with traditional monitoring techniques. It established the feasibility of maintaining the framework with similar values, which reduces the number of requests to the switches. However, their study focused on video flows only and thus, monitored its data rates of the design framework without taking consideration for other traffic types. Their algorithm lacks the measurement of other variables, such as delay, packet loss, jitter or packet duplication.

In regard to the link and path estimation, Singh et al. (2017) used port statistics. The Open Daylight controller uses a network topology to query network switches to get bytes transmitted by each port. The bandwidth consumed by the link between these bytes is the difference between the two neighbouring switches. The maximum connecting capacity for calculating the available bandwidth is then discounted. The path bandwidth available for a flow is the minimum of all link estimates.

Kandoo was suggested by Yeganeh and Ganjali (2012), which appears to be a highly effective way to increase general scalability and to help a network deal with increased traffic volumes. It is essentially a network resource designed to reduce the number of controller events processed at the control plane and works by using two controller levels. The bottom level consists of several controls that are unaware of the network status and have no links. It covers the majority of events to significantly alleviate the pressure at the highest level, thereby maintaining the broad conditions of the network. Other researchers, however, recommend alternative methods to maximise CPU and switch interactions.

Rowshanrad, Namvarasl, and Keshtgari (2017) proposed a system for monitoring queues in each link through SDN, with the extension of the Floodlight controller using OpenFlow as a southbound protocol. It is also built into the network controller that allows device reports to be used by QoS and traffic engineering applications for automated traffic and QoS setup. The bandwidth of the available queue is monitored using network switch polling queue statistics. The difference between two transmitted byte readings is determined by using a queue bandwidth over the time frame. By subtracting the used bandwidth from what the queue is configured as, the available bandwidth is obtained.

3.3.2. Network congestion analysis in SDN

The deterioration of network capacity is an important factor behind congestion (Rahman et al., 2017). It occurs when the node or connection carries more data than its capacity, which means there is zero or negative residual capacity. This is either due to unequal traffic distribution, lack of node hardware or the connection itself, as He et al. (2019) have reported. Congestion typically brings about packet delays, loss, and errors, which damages throughput and increases latency with packet delay variation (PDV). This negatively affects the network's QoS. Contemporary networks use congestion controls and prevention mechanisms to prevent packet congestion and collision. Packet dropping takes place when packets cannot be put in the outgoing queue with the appropriate buffer space (Rahman et al., 2017). Hagag and El-Sayed (2012) wanted to enhance network efficiency through the use and detection, classification and discussion of TCP congestion controls like Tahoe, Sack, Reno, NewReno, Vegas, and Westwood. They compared, described and addressed some of these processes, as well as explaining their differences.

If congestion is disruptive and users encounter severe problems, it being highly unlikely that the fault can be attributed to just one location, then finding effective solutions becomes even more difficult. Yeganeh, Tootoonchian and Ganjali (2013) found that the more versatility the network loses, then the more it is being jeopardised. As it takes time to achieve a feasible solution, this will become increasingly unsustainable and the company will lose capital. According to Metzler and Metzler (2013), when there is inactivity and congestion , these both cause severe network flexibility problems. The overhead expenses and the distribution of flow entries place both constraints on the inherent flexibility of SDN networks. The controller should, thus, be allowed to adjust the header of the core components to regulate and monitor the quick expansion of flow entries. The intended switches are egress and ingress.

Several software-oriented solutions can be used to overcome many of the broader SDN usability limitations. As Kim et al. (2012) suggested, for example, CORONET is a device that can respond to errors extremely fast. Due to the VLAN components attached to its local switches, it is very suitable for large, extensive networks. The main benefit of CORONET is that it can rebound with a minimum downtime from connect and switch faults. It is also

compatible with dynamic networks that can be changed. It uses multi-path routine strategies, if necessary and can be combined with virtually any form of network topology. The CORONET device also can use centralised controllers to direct and deliver packets. CORONET is characterised by a set of modules, which are designed to map routes, undertake traffic control, explore the topology, and to find the best (fastest) packet path. One of the key elements is the use of VLANs, which is an efficient means of standardising packet movement that does not over-complicate the processes. They also help monitor the volume of flow controls and promote the maintenance of a completely adaptable and scalable system.

3.3.3. Latency (delay) analysis in SDN

The latency is the total time taken for data to be transmitted from the receiving host to the destination host via a network and is measured in milliseconds (Comer, 2018). A variety of forms of delay exist:

- Propagation delay: time taken for a signal to pass through a medium;
- Access delay: present in wireless Wi-Fi, using a media connection CSMA/CA method;
- Switching delay: the time taken to locate the next hop and begin the transmission by the system (router or switch);
- Queuing delay: time the packet is held before others that were sent previously are forwarded to the First Input First output (FIFO) queue. The network is labelled congested as queuing delays develop and therefore, FIFO is selected to be used as a baseline condition for quantitative performance comparisons in this thesis. The FIFO queuing algorithm is introduced and discussed in depth in Chapter 5;
- Server delay: time necessary to review, measure and send a reply.

A link latency monitoring mechanism was proposed by Zhang et al. (2020), which has two parts,, namely LLDP discovery and echo monitoring. The first module uses the Link Layer Discovery Protocol (LLDP) packets to measure the communication latency of two switches. The selection of optimal paths based on the connection latency information is suggested by a dynamic routing algorithm. The RYU controller sends the first switch an LLDP packet and directs the controller to forward the second switch. There are no guidelines about how to treat this packet (i.e. how to forward these packets to the controller) and so, it

sends the controller a packetIn message. This means that, when the process is tested in the other direction, the controller records the latency of the connection between the two switches in just one direction. A different process is conducted in the other direction. The second module is used to measure the time spent by sending echo messages time-stamped from the controller to a network switch. When the packet is received, it is returned to the controller by the switch and this calculates the latency of the link between any switches.

The authors Sinha, Haribabu and Balasubramaniam (2015) put a packet in the network, which runs along the test path to test the latency back to the controller, called the Time-to-Live (TTL) loop. As for all sample-based methods, it must insert TTL rules, lower its value, and forward them to the controller when TTL = 0. This method is used in routed environments to reduce the impact of routing loop and avoiding router failures. Altukhov and Chemeritskiy (2014) used the arrival time and Round Trip Time (RTT) values for measuring route delays, whilst Selmchenko et al. (2016) tested the end-to-end delay of the flow. The sample packet travels the path between the first and last switches and returns to the controller. For each tracked flow this procedure is replicated frequently. After the RTT is collected from the controller for both switches the flow delay is calculated upon receipt. Selmchenko et al. (2016) applied a coefficient of proportionality between 0 and 1 and considered the RTT to the controller from first to last. Allakany and Okamura (2017) also suggest that delays are assessed with a probe-based sub-path approach. Link delay is determined by such subpath delays and similarly addressed as with other delays, as previously discussed.

The SLAM system for tracking path latency in SDN-based data centre networks was suggested in Yu et al. (2015) to distinguish high-lapse network segments. The authors used a customised probing packet when a new, unprecedented packet is received in the network switch to trigger PacktInt Notification for the controller. The latency is then determined by the time the controller receives the PacktInt message.

The authors Rowshanrad, Namvarasl, and Keshtgari (2017) adopted the same probe-based method for checking the queue delay. The first switch that is programmed to transfer to the next is sent a special packet. The receiving switch sends the packet to the controller because the controller cannot determine what to do. In the first switch, the difference with the throughput action is selected in the checked queue with the *enqueue* action (i.e. in the later OF versions, converted into the set queue). In this way, the queue delay can be checked from

S1 to S2. The authors Haiyan et al. (2016) derived a model of queue delay from network parameters, including queue buffer size, queue bandwidth, number of flows and the tested Mininet propagation delay, among others. Approximate queue time was then derived from that model and used to monitor the delay of end-to-end traffic. A flow may be shifted into a separate queue when an upper delay limit reaches a specific delay maximum. The most fascinating aspect of this work is that the control can retain or monitor most of the parameters used in that model by switches. Injecting sample packets earlier when no traffic occurs will allow for estimating of the propagation delay in a network connection. Authors believed that the delay in a queue is the primary explanation for the latency of the network, since the time for packet processing is negligible, and that for propagation is constant.

3.3.4. Throughput analysis in SDN

Throughput reflects a device's efficiency assessment (Comer, 2018), being a measure of transmission of data across the network and defined in bits per second (bps). Jimson, Nisar, and bin Ahmad Hijaz (2017) reported that network performance is specified as the product of the likelihood that each connection will succeed and the expected number of simultaneous transmissions.

In OpenNetMon, a POX OpenFlow controller module QoS metrics per-flow enabled fine-grained traffic engineering, as proposed by Van Adrichem, Doerr and Kuipers (2014) for reducing overhead, the monitoring module polls statistics from ingress and egress to measure the byte number transmitted during the flow. Luong, Outtagart and Hebbar (2016) used an OpenFlow statistical collection of bytes and packets that have been switched through. The key goals of the analysis were to cut total costs and increase accuracy. Also, to measure the current transmission rate of each link by counting bytes that move through the link over time. When a new connection is made, these determined values are used by the monitoring module to manage loads. When the connection usage is 80%, a different track is selected to prevent packet loss, as set out in Binsahaq, Sheltami and Salah (2019), instead of the shortest path.

The consequence of implementing flow rules must not be limited to a breach of the policies introduced. The authors Liu et al. (2015) demonstrated that these rules can influence the network output in the sequence inserted. They proposed a scheduling algorithm running

on the SDN controller to determine, first, the impact of new regulation on the existing network, which would lead to network links being congested. This problem can occur, if the routing module takes a reversal decision without taking into account changes in the impact of the flow migration or flow input series.

3.3.5. Packets delay variation (PDV)/jitter analysis in SDN

The arrival delay variances in flow packets were calculated by Karthashovsky and Buranova (2018) and named PDV or jitter. PDV is primarily used in real-time voice and video communications networks (Comer, 2018). Many studies have been conducted to boost QoS performance metrics and to address delay variations. Regarding which, Kleinrouweler, Cabrero, and Cesar, (2016) proposed HTTP (DASH-aware) dynamic adaptable streaming based on SDN to provide stability and DASH-based video, while avoiding the negative effects of TPT and volatile streaming traffic. This design aims to retain an easy and scalable DASH protocol stack, while enhancing the consistency of the viewing experience, it having three layers: SDN network application manager, SDN network management and the network programmable infrastructure. The SDN framework supports the selection of their required bitrate for a range of competing DASH customers, while the network controller management is responsible for controlling and configuring all network components. However, the architecture does not support system heterogeneity and DASH players must collaborate for bitrate selection with the service manager.

The latency and jitter of SDN were analysed by Numan et al. (2019) against traditional IoT communication networks. Mininet simulations showed that the average SDN jitter and latency per packet are three times smaller, which translate into better efficiency under different traffic conditions. The results showed that SDN improves network efficiency by reducing network overhead created from frequent communication attempts between the control and data planes each time a packet is received. However, it cannot guarantee optimal results, because it may represent a single point of failure by using a single control plane design. It is also recommended that the distributed control plane architecture be used, although this also has its specific challenges, including how controllers can be positioned to ensure the best results.

3.4. Quality of Service (QoS) in SDN OpenFlow Queue Management using the Traditional QoS Frameworks, IntServ and DiffServ

Two conventional QoS models are used to support network performance, namely Integrated Services (IntServ) (Braden, Clark and Shenker, 1994) and Differentiated Services (DiffServ) (Blake et al., 1998). The former has to store enough resources along the route used by the service packets. This work has developed into the concept of a signalling protocol called the Resource Reservation Protocol (RSVP) for the application of the IP protocol (Braden et al., 1997). It has a problem with scalability, however, since it needs to retain a location on each traffic flow network node. Packets are labelled in DiffServ to discern or receive from the service level that they need (i.e. VoIP needs up to 150ms (miliseconds) delays), with the packet then being handled differently depending on its label or class, and the QoS policy is configured on every node (Sanaei and Mostafavi, 2019). Different QoS classes for variant services may, therefore, be specified. DiffServ is built to support end-to-end QoS by ensuring QoS on each network intermediary node. This is an issue for network operators since error margins are high and any node needs to be reconfigured in the network to reflect an alteration to QoS policy or new SLA requirements. The use of automated tools may be a solution; however, the variety of network providers and the complexities of the current architecture (i.e. the traditional networks with varying equipment) make the situation even worse.

Frameworks like IntServ and DiffServ built to provide QoS assurances in traditional networking, as argued in Binsahaq, Shelstami and Salah (2019), also struggle to answer end-to-end questions of QoS at a broad scale. With the strain on the internet, service providers have had to choose between adding more capital to their networks or enforcing stringent policies that would not satisfy their customers. They are; however, obliged, as negotiated under the service level agreement (SLA), to provide services of a certain standard. Nevertheless, as the index reports from Cisco in 2017 reveal, the huge rise in the number of internet-connected devices (e.g. mobile devices for users, servers, things) has reaped a lot of monetary reward for providers of services. At this point, it is a challenge for many service providers or network operators to provide services with such a QoS guarantee, while efficiently using network resources (Balakrishnan, 2012).

During the SDN era, these models and frameworks have been tested by several scientists within various contexts. For instance, Wallner and Cannistra (2013) used queue-based classification strategies to provide QoS support for floodlight-controlled SDN networks and demonstrated a QoS method that is defined and handled by the Open vSwitch centralised network controller and virtual switch. To this end, the QoS support in Floodlight-based SDN networks was exploited with traffic shaping (rate limitation) and Differentiated Service Code (DSCP) approaches. The authors identified various groups of services along with rate-limiting routes between switches offered, for example, the only expedited forwarding and best effort types. The main player is the "SDN module", which covers packet matching, classification, and flow operations, such as input, deletion, etc. The QoSPath application allows for the addition of traffic shaping (rate limiting) and DiffServ DSCP policies along a given Dijkstra-based shortest path first routing circuit using the "circuitpusher" based application that ships with the Floodlight controller. To evaluate QoS metrics, their work required a detailed assessment.

Xu, Chen and Qian, (2015) offered an IPv4 ToS based QoS system. It classifies QoS flows and best flows, subsequently assigning queues to them according to their priorities. Then, when the original route cannot provide bandwidth, the high-priority flow will be rerouted by the optimisation algorithm. This system allows queue structures for the transfer of QoS flow until the high priority flow direction is not feasible, with the flows being categorised as two types: QoS flow and best performance flow. The best-effort flow is distinguished from that of QoS, where it is used as the basis of performance without any level of priority requirement. In addition, in this research, QoS flow-1 and QoS flow-2, which represent the different priority levels, were listed as two subtypes. Depending on the particular case, these can also be divided into many sub-types. This work did not; however, define the type of application, i.e. whether video or audio and how to differentiate between them so as to ensure a sufficient priority and bandwidth that would prevent delay and congestion.

Moreover, Wang W et al. (2015) presented an automated QoS management software model. The proposed model covers several QoS features, including package labelling, queue management, and scheduling. It uses Weighted Random Early Detection (WRED), priority queue management (PQ) and Weighted Round-Robin (WRR) algorithms to schedule queues. It was also proposed to increase the utilisation rate of network resources through a

Collaborative Borrowing Based Packet-Marking (CBBPM) algorithm. Nox1.1 Core platform is used for network management coupled with OpenFlow version 1.3.

Al-Haddad, Velazquez and Winckles, (2017) reported that Bari et al. (2013) had put forward an additional approach in PolicyCop. The developers of the QoS policy management system often use a Floodlight controller OpenFlow protocol as it provides an interface to define and execute QoS-based service level agreements (SLAs) through the OpenFlow API. The main objective is to track the network by enforcing data/control/management plans via the autonomous QoS policy implementation mechanism for SDN. The management plan consists of a policy validator and policy enforcer responsible for the validation and implementation of QoS policy; forming the basis for a routing decision in the control layer. They established several software applications which provide different control functions for the management plan. No fine-grained QoS techniques were; however, involved in this work (packet sorting, packet labelling, tail management and queue scheduling).

Huang et al. (2015) suggested a hybrid scheduling system that combines priority queuing with packet sharing to meet various QoS guarantees for SDN multimedia apps. The proposed output framework guarantees heterogeneous data flows, including worst-case delay and the backlog of queues. This is achieved by integrating PGPS with preventive planning algorithms using network calculus theory, which provides heterogeneous flows with deterministically guaranteed QoS.

Furthermore, Egilmez et al. (2012) put forward OpenQoS, a controller design based on OpenFlow aimed at multimedia delivery with end-to-end QoS built upon dynamic QoS routing for multimedia apps using video. The research was focused upon Floodlight controllers and OpenFlow protocols, which contribute to minimising adverse effects, such as latency and packet loss, on other types of flows, while other data remain with OpenQoS. Whilst the incoming traffic has successfully been categorised into data flows and multimedia flows, with the traditional shortest-path for multimedia flows, their case study revealed no involvement in using traffic shaping or slicing methods.

Finally, QoSFlow was proposed by Ishimori et al. (2013) to boost QoS efficiency through FIFO schedules and stochastic fairness queuing (SFQ), with a bandwidth guarantee. To extend the standard OpenFlow 1.0 version of the data route, QoSFlow uses several Linux

kernel packet schedulers to perform routing and traffic engineering. The proposal provides a control for the following: HTB (hierarchical token bucket) by Hubert et al. (2002), RED (randomly early detection) by Floyd and Jacobson (1993) and SFQ stochastic fairness queuing) by McKenney (1990). Their findings showed that the two video flows that were generated and tested are better for SFQ than for FIFO. That is, for Video 1 and 2, the difference between them is around 48.57% and 68.57%, respectively. Hence, SFQ assigns every flow to a hash bucket in the enqueuing phase and deploys a round-robin treatment when sending packet flows.

3.5. Challenges of SDN Controller Scalability and Performance

The weaknesses of traditional architecture, such as those found in Pujolle (2015), are becoming increasingly important: modern networks are currently no longer maximising investment (i.e. Capital Expenses/one time purchase "CAPEX" and Operational Expenses/pay as you go approach "OPEX"). Also, the networks are not versatile, market time is much too long, and delivery strategies are not quick enough. Additionally, QoS solutions proposed are not sufficiently efficient to address the problems of traditional networking paradigms in this regard. SDN aims to minimise costs through virtualisation, automation and simplification. The relation between QoS and SDN is that the former is known as a collection of technologies used to provide overall performance management and resource allocation on a network to guarantee its ability to reliably run high priority traffic flows under a limited network capacity. To that end, SDN makes it possible to configure the networks, to set-up the system fast and to deploy the network in a customised QoS, rather than a general one. While SDN has many network and flow management advantages, it still has several performance limitations, with control plane scalability being one of the key challenges. Hence, the SDN controller is at risk of being congested, if the network significantly increases the number of end-network devices (Abuarqoub, 2020).

The risk of severe traffic congestion increases when a large number of packets are sent to one controller. In reality, the effects of overabundant packets being moved through the channel are unavoidable at some point. Solid, competitive networks ensure efficient congestion management of switches and require no explicit control commands or guidance, as reported by Alsaeedi, Mohamad and Al-Roubaiey (2019). In the end, regardless of what

decisions the network requires, the probability of congestion should be weighed up when increasing the number of nodes. As the number of flows, switches and bandwidth increase, the amount of commands imposed for the controller is continuously increasing. The components can often be overloaded, so careful supervision is necessary.

Enabling the movement of ample streams continues to be very difficult for OpenFlow networks. Indeed, the modification of packets on the control plane takes a long time. Given these vulnerabilities, the issue of using OpenFlow applications to control core network processes remains ambiguous (Badotra and Panda, 2020). This is one of the reasons why OpenFlow is usually used for the network management and controlling parts of a network and not for the core functions. Heller, Sherwood and McKeown's (2012) research clarified how one control unit alone can achieve an acceptable degree of inactivity. The scientists stated that k controllers' implementation decreases inactivity by k. Two main barriers to the scalability of OpenFlow frameworks remain. The first is that the rate at which new flows can be developed is limited by the hardware. The second is that the amount of flows that can be accommodated is relatively limited as a result of hardware and table size capacity.

Controllers must maintain a global view of the network topology graph to achieve centralised, optimised network flow control and setup. Replicating the global link-state view in any distributed controller will ensure that the data plane is managed logically and in a centralised and transparent manner. In addition, it can avoid misrouting triggered by incoherence between two consecutive synchronisation periods by controllers. Maintaining the network-wide replica of controllers in a large dynamic network and topological conditions can lead, however, to a massive volume of synchronisation traffic, which may saturate the controller.

Levin et al. (2012) and Guo et al. (2014) in earlier literature provided solutions to increase the scalability of SDN, by dealing with various consistency and synchronisation issues. Whilst any consistency can ensure a quicker response time for controllers, it can lead to routing problems (for example, routing loops, black holes). Good consistency, on the other hand, can prevent routing issues with increased delay costs, lower availability, and overhead computing complexity. Adapting consistency to the current network state will ensure scalability. However, it takes more technical sophistication to track and quantify traffic measurements in highly dynamic networks to achieve consistent adaptive controllers.

Synchronisation based on events can also minimise consistent traffic by simply syncing the modified controller state with changes in the topology of the data plane and the connection fault events. Controllers are partitioned into clusters, and ultimate continuity in reacting to events involving changing data plans is a realistic and agile solution in large and complex SDN networks. However, packets on the broken connection will be lost and dropped before the assigned controller recalculates and reinstalls the alternative forwarding rules for the network failure to be restored. Therefore, OpenFlow-SDN needs to avoid connection failure during a limited period of recovery by offering alternative routes in advance so as to prevent delays and traffic overhead (Alsaeedi, Mohamad and Al-Roubaiey, 2019).

3.6. Virtualisation and Slicing Mechanism Solutions for QoS in SDN

Over the years, many attempts have been made to produce QoS using SDN virtualisation and slicing mechanisms, including the following.

Bozakov and Papdimitriou (2012) introduced a method called AutoSlice as a special purpose controller, which is able to develop diverse network resource slices as well as ensure that they are isolated from one another. VLAN tags are used by the controller to slice. However, a major disadvantage of this method is that it fails to ensure QoS. It is notable that AutoSlice has only been implemented over the OF protocol. Nevertheless, it has addressed scalability problems concerning network hypervisors through enhancing resource utilisation and reducing the drawbacks of flow-tables through accurately monitoring the flow traffic statistics using a single third party so that the vSDN topologies' mapping can be controlled.

Yamanaka, Kawai, and Shimojo (2017) introduced AutoVFlow as a tool to enable flow space virtualisation in expansive networks, while not requiring a third party. This proposal added minimal overhead to the performance of data transmission, whilst communicating in a wide-area network. Further, AutoVFlow provides a clean-slate network design to different applications and helps infrastructure administrators assist numerous tenants. An innovative network hypervisor prototype is CoVisor, as proposed by Jin et al. (2015), which allows different controllers to manage the same traffic together. It also includes abstracted topologies that offer customised virtual topologies, while enabling administrators to determine access controls overseeing the packets, which can be seen, monitored, rerouted, or modified by a

controller. This method's major benefits are that it introduces efficient algorithms to create composing controller policies, turns virtual networks into concrete OF rules, and processes controller rule updates effectively.

Al-Shabibi et al. (2014) also presented OpenVirteX that offered virtual SDNs using topology, control function virtualisation, and address as a form of proxy work between the forwarding device and network operating system. As noted by Racherla et al. (2014), there are other solutions that are SDN-based for testing on-demand virtual network provisioning. RadioVisor was presented by Gudipati, Li, and Katti (2014) as a system architecture involving three major aspects: (1) a device and application to slice mapping, (2) a slice manager, and (3) a 3D resource grid allocation and isolation function. This method aims to isolate radio resources. RadioVisor also allows dynamic resource sharing and maintains control over the resources using the virtual operators, while slicing the 3D resource grid based on resource requests as well as service agreements in a max-min manner.

Doriguzzi-Corin et al. (2014) introduced a distributed virtualisation architecture to be executed in multi-version OF scenarios. This method was developed to achieve three major goals:

- Avoiding Single Point of Failures (SPoF) using a distributed slicing architecture;
- Providing an OF version agnostic slicing mechanism;
- Reducing the latency overhead resulting from the slicing operations.

This proposal was later investigated by Depaoli et al. (2014), who also showed the way in which the proposed network virtualisation approach based on xDPd and the VA could achieve the following: (a) several OF protocol versions that control the same physical infrastructure, being used simultaneously; (b) IPv4 and IPv6 multicast streaming experiments to be conducted on various virtual networks, while ensuring they do not interfere with one another; and (c) virtual networks for operating, if one VA instance fails (Depaoli et al., 2014).

Salvadori et al. (2011) implemented the ADVisor architecture to provide additional features, such as virtual connexions and virtual ports management, on top of FlowVisor. The main aim of ADVisor is to be able to create complex virtual topologies, with a bandwidth

guaranteed that is separated from the physical topology underlying them, thus providing flexibility to adopt the required header space for L2 to define virtual topologies within the network. Nevertheless, this solution does not adjust the OpenFlow protocol to allow FlowVisor to configure data paths, such as scheduling and queue allocation.

3.7. Summary

The main goal of this chapter was to address some of the core SDN limitations. Performance parameters, QoS mechanisms and adopted methodologies have been evaluated for solving the congestion problem in traditional networks and SDNs. To enhance QoS and define different congestion control, traffic management systems, admission control, as well as policy rules, classification and impact assessment of the virtualisation and slicing techniques deployed is paramount. To summarise, network performance is the most important aspect in the SDN paradigm, and the QoS parameters are crucial. As previously explained, there has been much research aimed at tackling these issues, such as Egilmez et al. (2012), Wang, Ng, and Shaikh (2012), Das et al. (2011) and Cao, Kodialam, and Lakshman (2014). However, there are still have drawbacks in their proposed solutions. Furthermore, in other studies, researchers focused only on improving the data rate for video flows, such as Fernández, Villalba, and Kim, (2018), while Rahman et al. (2017) tried to tackle the buffer problem to avoid network congestion. The authors Rowshanrad, Namvarasl, and Keshtgari (2017) and Haiyan et al. (2016) studied queueing delay. In addition, jitter of SDN was analysed by Numan et al. (2019), which has its specific challenges, including how controllers can be positioned to ensure the best results.

In this chapter, the literature has shown the importance of the FIFO queueing algorithm in several studies, such as Comer (2018) and Ishimori et al. (2013), which has been used as a queueing algorithm by default as a baseline condition for quantitative performance evaluation in SDN. However, in all the related studies, the research design varied depending on the mechanisms used. Nevertheless, using FIFO in the current study as a baseline condition is a powerful methodology for the comparative evaluation approach, in particular, owing to its simplicity in its queueing strategy. A second reason for using FIFO, is that it is expected to show a characteristic qualitative pattern of performance, with which other algorithms can usefully be contrasted, i.e. patterns across a mixture of different traffic types, which can be

separated as slices (audio, video, data). And potentially, also across some other parameters within any typical test run, such as scalability, to provide high throughput, system availability to avoid insufficiency, and high security to prevent any breaches.

As SDN is in its initial development stages and continues to show a promising future for network growth, it requires considerable work to make it more efficient through optimisation across various networks and determining the ideal trade-offs among implementations. It is also important to generate quantitative metrics for assessing SDN performance concerning its scalability, availability and latency. An SDN framework's individual network applications and services also have to be improved, especially as regard to audio, video and data. For this, the best solution is to propose new more advanced queueing algorithms based on the Packet Tagging and Forwarding, Tagging and Queueing, and Packet Scheduling ones, which should be implemented through Sliced-SDN architecture for classifying video, audio and data flows to enhance QoS. Also, there needs to be new algorithms to implement flow forwarding policies and shape the traffic in advanced methods using the existing QoS strategies, such as a class-based differentiated service (DS) using SDN and the OF protocol, as this can define packet-levels or advanced flow-level priorities in sliced-SDN. This could lead to the solution of throughput, delay and jitter problems in one framework. Despite certain IS-oriented OF QoS models having been defined within this chapter, there have only been a few practical implementations. In particular, comparative evaluation methods are required and this is addressed this thesis.

Next, Chapter 4 provides an overview of the most effective methods used to conduct the research project, including an explanation of research quantitative methodology, experimental design and analysis as well as how the research objectives have been achieved, followed by description of the research design phases.

4. Chapter Four: Research Methodology

4.1. Introduction

The purpose of this chapter is to describe the methodology for the research project. Explanation and justification for the experimental design and analysis, how the research objectives have been achieved and description of the research design phases are provided.

4.2. Quantitative Research Methodology

In general, research is a systematic investigation using different methods to address the research questions, find differences and similarities, in techniques interpret the data collected and analysing the data using different research methodologies, such as qualitative, quantitative or a mix of both (Clough and Nutbrown, 2012). A quantitative research methodology generates large scale statistical data using different strategies (Greenfield and Greener, 2016), such as experiments, surveys or structured interviews. A qualitative research methodology generates non-numerical data, such as design and creation, case study or action research (Dawson, 2009). For this research study, an investigation strategy was adopted to solve the problems of "Network congestion" and "Improving the Network QoS" using a quantitative research methodology. A systematic literature review was undertaken to obtain the qualitative (secondary data) presented in Chapters 2 and 3. Then, a comparative experimental design and analysis was planned carefully, and this is presented comprehensively in Chapter 8.

Research objectives have been pursued to test the research hypothesis presented in Chapter 1.

 # Objective (1): Research the fundamental concepts for implementing QoS in SDN, by classifying QoS protocols, as well as optimising techniques, such as traffic engineering (TE), to fill the knowledge gap.

This objective was justified by undertake the research background and a systematic literature review of QoS mechanisms, protocols available in traditional networks and SDN to identify what protocol is suitable with the slicing technique and QoS mechanism.

 # Objective (2): Collect and analyse data for implementing the FIFO, Traffic Shaping and QoSVisor algorithms in sliced-SDN.

This objective is addressed in Chapters 5, 6, and 7. It is achieved by proposing a solution to bridge the gap in knowledge identified in literature. In Chapter 5, the FIFO algorithm pertaining to the standard queueing system is introduced and implemented in sliced-SDN. Its performance is compared with that of the new proposed algorithms, TS and QoSVisor are presented in Chapters 6 and 7, respectively.

The experiments in all the related studies varied depending on the mechanisms used. Regarding which, a comparative experimental design in studies using a Mininet virtual testbed, such as those of Numan, et al. (2019), Fernández, Villalba, and Kim, (2018) and Singh et al. (2017), has shown an effective performance evaluation. First, FIFO was used in the current study as a baseline condition, being selected as a powerful methodology for the comparative evaluation approach. Second, due to the limitations observed in the literature in relation to experiments, a Mininet testbed was used to address the experimental design. The testing plan was maximised to the highest stress test level, with the experiments lasting 1, 5 and 15 minutes in different groups. Then, the large scale results were analysed statistically using SPSS. This was undertaken with a mixture of different traffic types, separated as slices (audio, video, data), further details of which will follow.

The solution presented in Chapters 6 and 7 is a framework that can classify the data according to traffic type (voice, video, and data transfer) and filtering pre-classified packet data to prioritise their delivery. According to this procedure, the framework should handle the isolation of slices created for each type of traffic in a subset pattern known as “flowspace”. The flowspace is defined by a collection of packet headers (n-bit headers) for each slice that does not affect the QoS characteristics for the applications that require high bandwidth. Moreover, the solution manipulates the switches via OpenFlow and reduces the delay, while sending packets by allocating the resources for each slice, thereby solving the congestion problem.

Data collection: Execution of systematic experimental plans. This was achieved by developing a testbed based SDN simulation using Mininet emulation. The testbed enables slicing techniques, algorithms implementation, and delivery of the QoS parameters of throughput, delay and jitter for different types of traffic. Three switching systems are implemented, namely FIFO standard queueing to compare it against the Traffic Shaping algorithm and QoSVisor with a PTA Agent. A custom topology of five nodes of Open vSwitch (OVS) switches is configured with six hosts, with three different flowspaces being created using the FlowVisor slicing tool with the control plane led by Floodlight controllers. To allow for effective evaluation, the data was gathered in three sample duration plans divided between 1, 5 and 15 minutes in total, alongside with bandwidth sample divided between 40, 70, 100 Mbps.

Objective (3): Design, implement and evaluate a novel approach for evaluating queuing systems in sliced-SDN.

Objective (4): Design, implement and evaluate a novel framework for QoS implementation in sliced-SDN.

These two objectives are achieved in Chapter 8, where data management and statistical testing are undertaken, with a novel comparative quantitative analysis approach for the data represent random sampling (replicates) from internet traffic. For data reliability and objectivity between the variables, SPSS software is used to analyse three large groups of primary data for different switching algorithms and slices in order to evaluate the QoS characteristics. The SPSS statistical package Analysis of Variance (ANOVA) (Rutherford, 2001) is used to perform the first level of analysis of the data with various tests of hypotheses. Generally speaking, this involves testing for continuous or categorical relationships between the conditions set up in the study design in relation to the independent variables and performance outcomes (dependent variables).

The analyses involves various ways of collapsing the means of 10 repeated measures in each of the cells in the 243-cell structure of the data (3 traffic types) X (3 algorithms) X (3 durations) X (3 bandwidths) X (3 measures parameters). The advantage of utilising three levels is that this can provide at least a preliminary estimate as to whether the relationships are

linear or not. The analysis provides an overall performance comparison between the three algorithms, and clear identification of the differences between them. This allows for the research questions to be addressed, with highly effective level accuracy being achieved. Chapter 8 covers the scheme of conditions in the experimental design for evaluation and the statistical significances of differences by ANOVA on cell averages are also provided.

#Objective (5) Critically assess and conclude the effectiveness of the proposed framework using a new form of comparative statistical analysis.

This objective is met in Chapter 9, conclusion and future work are presented.

4.3. Research Design

The research design for investigating and meeting the challenges of traffic performance in SDN networks was structured into three phases as follows.

Phase 1: A systematic literature review investigation to obtain secondary data. This phase had two steps:

- ***Step 1***: Introducing the research background and undertaking a systematic literature review of QoS mechanisms as well as the queuing and protocols available in traditional networks and SDN;
- ***Step 2:*** Classifying and investigating slicing techniques and their impact.

Phase 2: Algorithms and SDN reference model design. This phase had five steps:

- ***Step 1:*** Develop and implement an SDN reference model testbed;
- ***Step 2:*** Introduce and implement the standard FIFO queueing algorithm in sliced-SDN;
- ***Step 3:*** Collect and analyse data on the network performance of FIFO queuing based on statistical evidence (primary data collection for FIFO based on the testing scenario (refer to Figure (4-1);
- ***Step 4:*** Design and develop the new traffic shaping (TS) algorithms for QoS in a sliced-SDN testbed to address the knowledge gap;

- ***Step 5:*** Design and develop QoSVisor with a PTP Agent, a more advanced algorithm for delivering QoS in sliced-SDN.

Phase 3: Test the proposed algorithms and evaluate them through comparative statistical analysis using SPSS software. This phase had three steps:

- ***Step 1:*** Execution of the systematic experimental testing plans to collect the primary data for TS and QoSVisor used to evaluate the QoS characteristics;
- ***Step 2:*** A novel statistical strategy for results validation and reliability testing using the SPSS statistical package Analysis of Variance (ANOVA);
- ***Step 3:*** Critical analysis and evaluation of QoS characteristics.

The figure below illustrates the testing scenarios between the servers and clients.

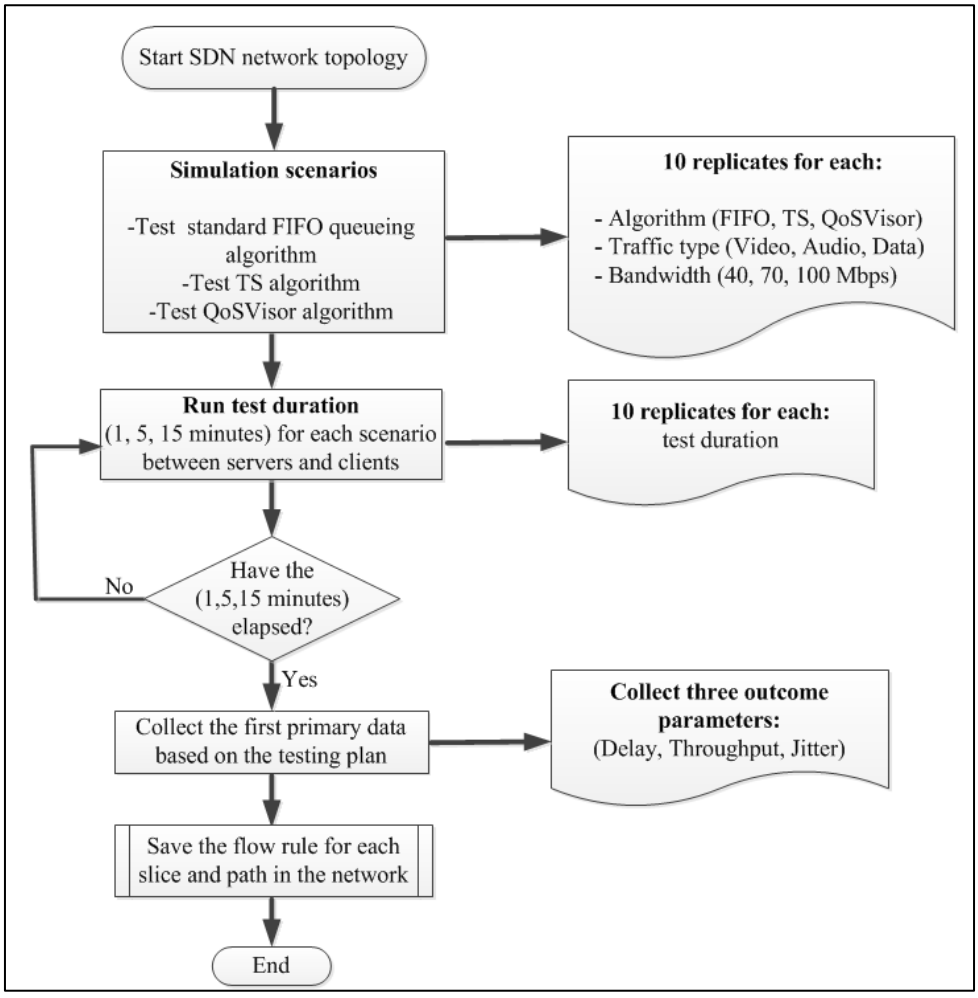


Figure (4-1) Illustration of the testing scenarios between the servers and clients

All the primary data collected based on the illustrated testing scenarios in Figure (4-1) is utilised for data management through the SPSS statistical package Analysis of Variance (ANOVA). Figure (4-2) provides a detailed explanation of the processes involved.

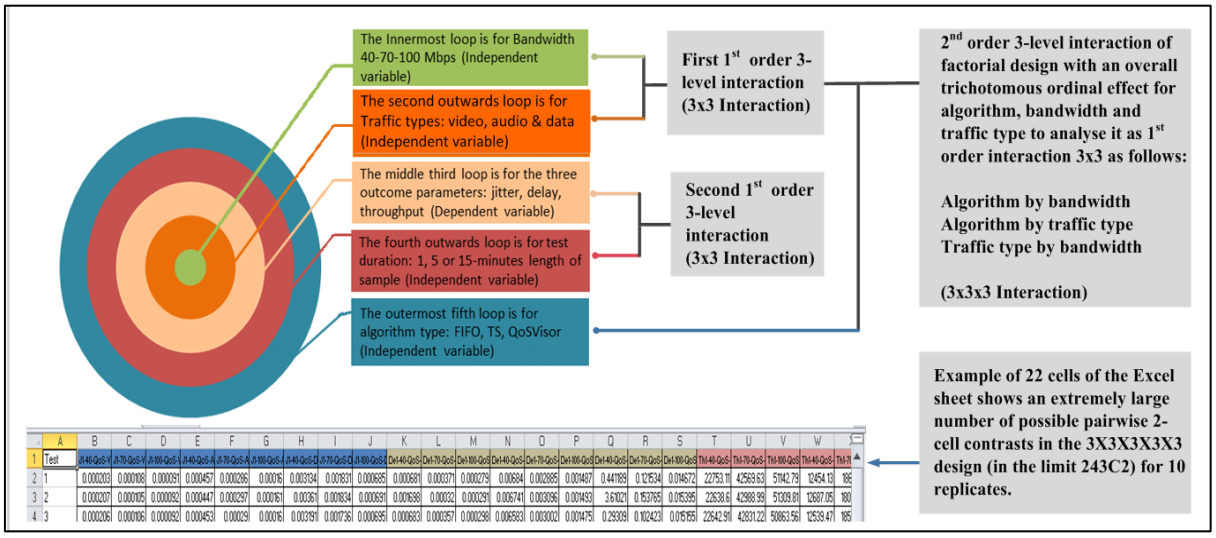


Figure (4-2) Data management for SPSS statistical package Analysis of Variance (ANOVA)

The following Figure (4-3) presents the research design.

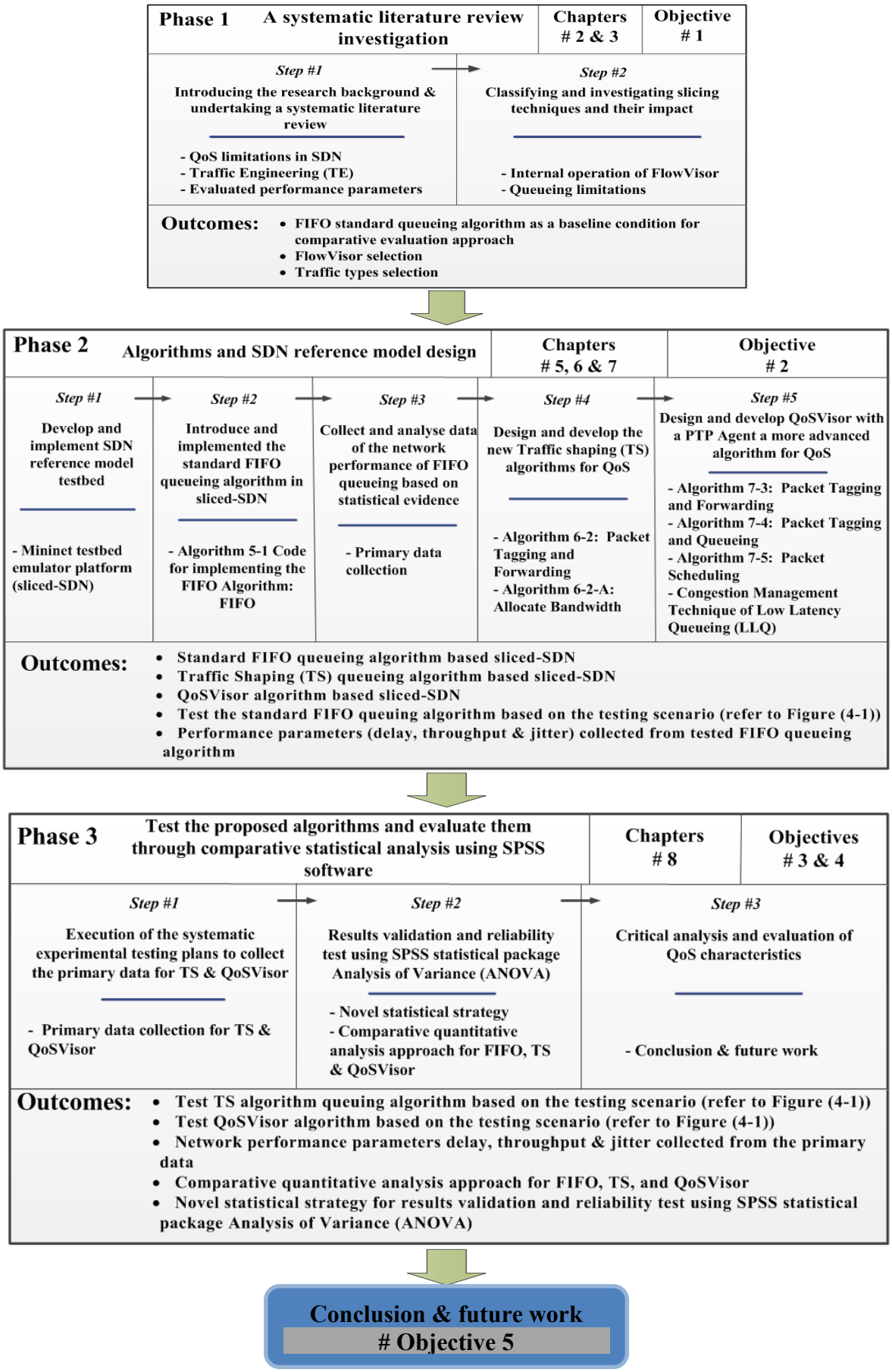


Figure (4-3) Research design

Figure (4-4) shows a comprehensive thesis map, which provides the thesis chapters against the research outcomes. This is in accordance with the research design approach based on three phases (refer to Section 4.3: Research Design)

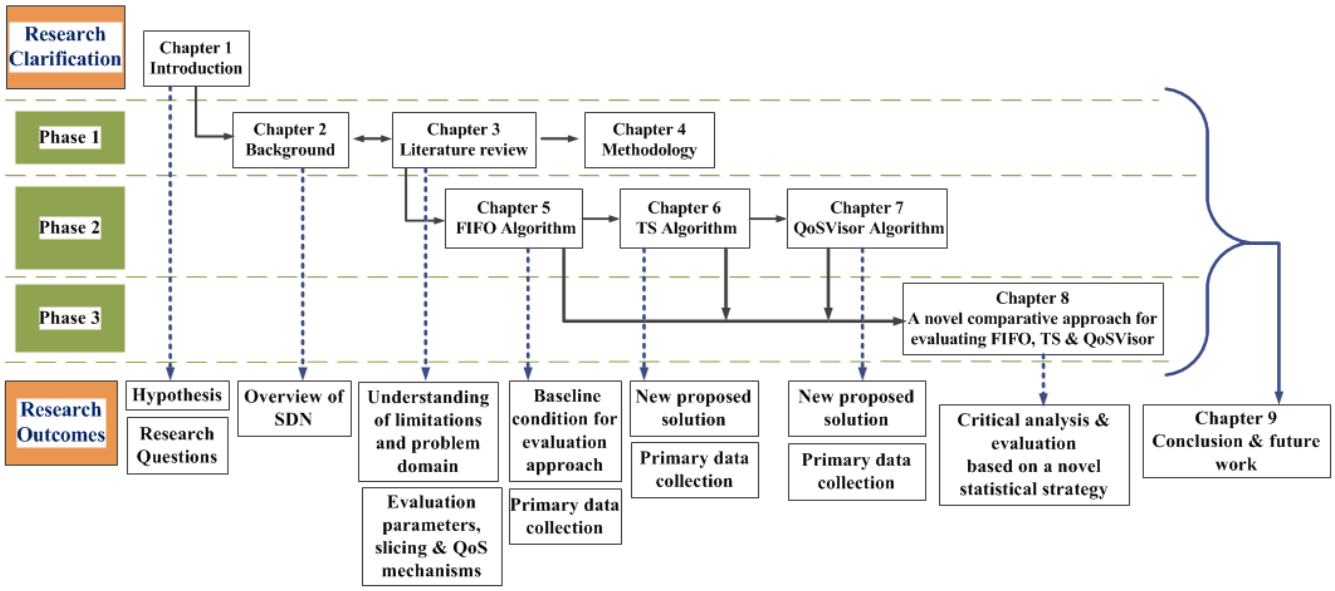


Figure (4-4) Comprehensive thesis map derived from Fatima (2016)

4.4. Summary

In this chapter, an overview of the methods used to conduct the research project has been presented, including an explanation of the research methodology, experimental design and analysis as well as how the research objectives have been achieved, followed by presentation of the research design phases.

The next chapter describes the proposed FIFO algorithm for implementation in a sliced-SDN framework as a baseline condition for quantitative performance comparisons along with detailed explanation of the implementation of the template design for the algorithm in a sliced-SDN testbed being provided. The particular traffic measures (throughput, delay and jitter), which are separate performance indices all contributing to QoS, have been integrated to provide an objectively rooted but overall and subjectively confirmable metric of QoS for each switch. Floodlight and FlowVisor controllers as well as OpenFlow (OF) switches, which consitute characteristic behaviour of SDN, are modelled and simulated via a Mininet testbed emulator platform.

5. Chapter Five: Implementing the FIFO Algorithm in SDN

This chapter introduces the First In First Out (FIFO) queueing system developed and implemented in a sliced-SDN. This is followed by explanation of the methodology for implementation and a system overview, highlighting the limitations of FIFO. FIFO is an ordinary queuing algorithm that has been widely adopted and used by researchers in order to evaluate the network characteristics behaviour of SDN and QoS. The particular traffic measures (throughput, delay and jitter) obtained from SDN FIFO model are separate performance indices all contributing to QoS, and are integrated to provide an objectively rooted, but overall and subjectively confirmable metric of QoS for each switch. The collected data are used for quantitative performance comparisons for this research. Different types of traffic, i.e. data, video and audio, were injected into the network, with each being contained in its own slice, and arriving from different ports for the SDN FIFO model, as explained in detail in Subsection (5.2.2). The purpose of this chapter is to establish the baseline of the current research's main limitations of FIFO queueing.

5.1. Introduction to the FIFO algorithm

The general behaviour expected from the FIFO algorithm in a sliced-SDN framework follows directly from the limited queueing facilities in FIFO (outbound), as described by Englert and Westermann (2009). The first packet to enter is the first to leave, so in other words, there is no prioritisation of traffic and hence, no attention paid to the QoS aims and criteria.

The FIFO system shown in Figure (5-1), is a basic switching system, which does not optimise QoS in video applications and has no algorithms for prioritisation. That is, the figure illustrates a simple description of single queueing; the First packet In is also the First Out. This system offers a baseline for scientifically evaluating other network traffic systems aimed at more ambitious attention to content. In FIFO, the data selector or the multiplexer will queue the data packets according to the arrival order and then, these packets will proceed to

the ports without any differentiation in the types of packet for each flow table. However, with more memory, more computing power and some delay, greater control can be exercised via packet scheduling mechanisms. The various scheduling mechanisms are the detailed means for implementing a policy on packet servicing, for example, to optimise the packet delay in the total queue and to avoid congestion when approaching system peak capacity. Packet scheduling can, in the extreme, even apply some packet dropping tactics, such as eliminating late arriving packets, eliminating the first packet that is waiting in the queue, or discarding packets randomly from the queue within a node (Mustafa, 2015).

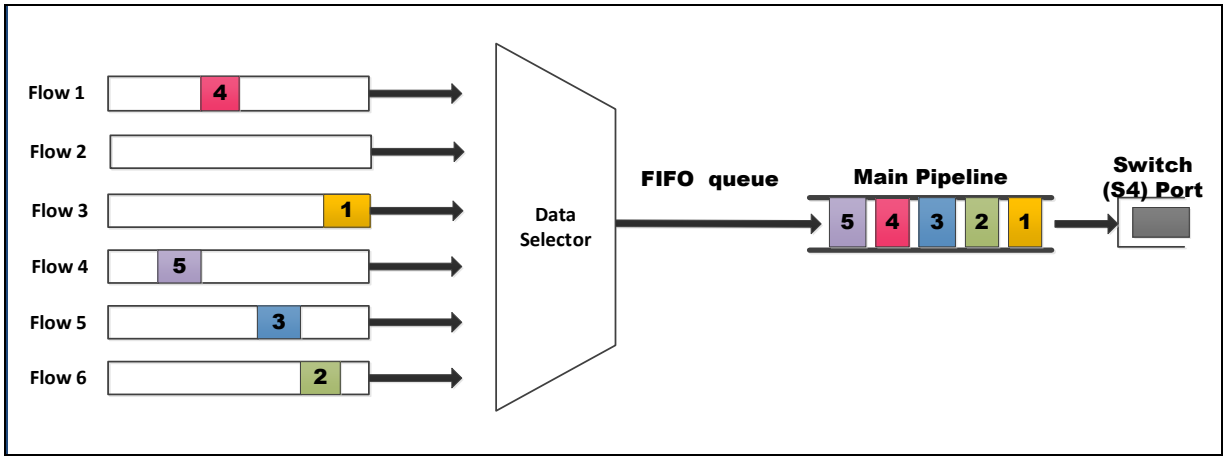


Figure (5-1) Schematic representation of the FIFO queuing methodology for packets arriving from different flows

Derived from Mustafa and Talab (2016)

Whilst only a baseline condition for quantitative performance comparisons is considered, the FIFO algorithm in a sliced-SDN framework is expected to show a characteristic qualitative pattern of performance, with which other algorithms (refer to Chapters 6 and 7) can usefully be contrasted, i.e. patterns across a mixture of different traffic types, which can be separated as slices (audio, video, data), and potentially also across some other parameters within any typical test run, such as scalability, to provide high throughput, system availability to avoid insufficiency, and high security to prevent any breaches. This pattern will become more distinct, the greater the average delay due to the length of the queue and level of congestion. These expectations, owing to the basic characteristics of the algorithm as conceived in terms of queueing theory, can be given the status of formal predictions from theory as capturing the limited sophistication of this algorithm. All aspects of performance

may be poor for some packet types within the different slices, because of the arriving order. For example, when three hosts (h1 h2 h5, which present different traffic types) within a designed system topology are sending traffic to h6 concurrently and in the case where the data class packets arrive first at the pipeline, then it will use most of the available resources. Other classes will be achieving a similar throughput, balanced and not showing guaranteed timings, no matter whether this is a more demanding application, such as video (and to some extent also audio). This convention of "classes" of information will consequently be used throughout this chapter. Any time "traffic types" are mentioned, this refers to the type of application (packet type) in the system. In a given case scenario, the data slice should, under FIFO, access the maximum of the available bandwidth only if it arrives first. Here, "available" reflects the fact that, because of fluctuating traffic demand, the available bandwidth is not a constant between networks, nor over time. To evaluate the algorithms across the situations they can encounter, three bandwidths of 40, 70, 100 mbps are sampled. Similarly, under FIFO, if the audio arrives first then it will take the maximum bandwidth, while the others (i.e. the video and data) will then share the remaining bandwidth equally. The limitation is obvious: as FIFO does not distinguish the importance of the traffic flows from the hosts/sources that are queueing, delays in entering the pipeline will occur for delay-sensitive traffic as the applied traffic load builds up.

The overall pattern of the performance parameters obtained from FIFO illustrates its limitations, especially for video applications. When the metrics for QoS are more specifically considered (delay, jitter and throughput), if the focus is on the critical signal class of video, FIFO will fail to deliver the desirable constant delay and throughput within a specified finite time that is desirable for video slices. This form of failure should be alleviated at high bandwidth, with FIFO achieving occasional acceptable performance levels for video, but this is not always guaranteed, as it depends on arbitrary precedence (i.e. which source accesses the pipeline first). The concept behind this first prediction is set out in some detail for the more general reader as an illustration in Chapter 8.

Further predictions regarding the details can henceforth be expressed in a more schematic and succinct way (i.e. differences in behaviour between the different queuing algorithms). This is achieved in qualitative terms, because the predictions of the difference between the algorithms is essentially ordinal, based on the nature of each algorithm. The research question – magnitude of benefit – is quantitative and the measures potentially permit

quantitative engineering trade-offs, e.g. with systems performance measures, cost or performance metrics. However, in the absence of a comprehensive mathematical theory of such networks the quantitative estimates of the differences in the predicted directions chiefly enhance the power for ordinal predictions, as will be revealed in depth in Chapter 8. The search to find studies for quantitative engineering trade-offs, specifically a trade-off analysis (also called a trade study) (Daniels, Werner, and Bahill., 2001) was a challenging task. Trade-off analysis is an analytical method for evaluating performance and comparing system designs based on user-defined criteria precisely in SDN publications.

In the search undertaken between 2016-2018 before implementing the proposed framework for this research in 2018, the researcher could not find such a case study to compare this analysis methodology. Despite this, the analysis method presented in this thesis is claimed to be an important novel contribution to the field.

5.2. Implementation Methodology of FIFO and Sliced-SDN Testbed System's Overview

5.2.1. Queueing limitations of FIFO in SDN

It is not quite correct to say that queueing is totally absent from FIFO, but rather, it exists singly so that it often affects the QoS when there is more than one type of traffic involved. SDN systems (control and data planes) are an independent aspect from FIFO. They manage the FIFO queueing between the controllers and the switches via the communications protocol, OpenFlow (OF) (i.e. same queuing incoming request at the logical ports for both switch and controller) (McKeown et al., 2008). The controller contains APIs for queue management on switch interfaces (data plane) of OF forwarding devices and through these APIs, other applications can perform traffic shaping, policing and scheduling features provided by Linux-traffic control on the controller (Pfaff and Davie, 2013). This is formally defined as a protocol that handles the SDN system by managing the forwarding information of the switch(es) (Lara, Kolasani and Ramamurthy, 2013). It is recognised (Ishimori et al., 2013) that in OpenFlow (OF) in SDN, a good QoS level is only achieved with bandwidth guarantees and through well-known FIFO queueing discipline on packet transmission and

delay. As a consequence of this limitation, OF with switching might not give good QoS in some applications, like multimedia and more specifically, in applications that are delay sensitive. Current versions of OF support simple mechanisms for achieving reasonable QoS, which can be controlled through the OpenFlow command-line utility software. However, the packets will cross from source to destination through the OF protocol under the FIFO queuing methodology. This affects the adequacy of QoS and traffic-shaping, whilst also having a negative impact on the order of packets in a queue for packet handling with higher priority. This constraint will clearly affect the Quality of Experience (QoE) of the network users.

5.2.2. SDN FIFO model

The particular model used for this study is illustrated in Figure (5-2), this being the SDN FIFO model. A Mininet testbed emulator was used to control the virtual environment represented by the virtual machine with virtual switches (Maugendre, 2015). The input flows from each Host (H1, H2, H5) are aggregated after every packet is transmitted through the switch (S1), as shown below in Figure (5-2). This switch is configured specifically to decide the packets routing towards the switches S2, S3 and S5 in predefined paths in the sliced custom topology. The different flow types (represented as X1, X2, X3) are as a result of the aggregation and propagation functions of the traffic flows through the SDN system used for this study. S1 and S4 function as pipeline-based switches (Sun et al., 2015), while S2, S3 and S5 function as forwarding plane-based OpenFlow protocol specifications. S1 performs the routing management as an ingress bound interface to decide what to do with the arriving packets by looking up in the flow table the information needed to determine the routing path, which it then sends to the outbound interface (S4) through S2, S3 and S5 (Al-Haddad et al., 2021).

In this model, the FIFO algorithm can be configured in either of two ways to set the queue length. That is, this can be done by choosing between “Packet FIFO” (which is based on the number of packets) or “Byte FIFO”, based on the number of bytes in the FIFO scheduler. To implement the FIFO algorithm in a sliced-template SDN, “Byte FIFO” is adopted here as the FIFO scheduler, because it work compatibly and with less complexity with the slice configuration (Ishimori et al., 2013; Al-Haddad et al., 2021).

As mentioned earlier, in Figure (5-2), the general queuing module is defined as FIFO queue (FQ), which describes the conditions used in this evaluation study, where the output traffic flows is converted into capacity units (bytes) and stored in an FQ with a capacity of k bytes, providing various capacities for output channels for three different test scenarios. The server(s) process(es) the queued data at a rate of 40, 70 and 100 (b/s) as the parameters chosen for the designed study, representing the limiting speed of the outbound interface. In the experimental system used for the study, the flow (Y) leaving the queue system can be divided into m output traffic flows, based on the testing scripts created for this purpose, to allow for configuring the proportion of traffic to be forwarded to every single output. Flow splitting is computed according to (a) the designed slicing mechanism in the SDN system, (b) the measured magnitude of every input flow and (c) the output configuration. Finally, the queuing modules store traces with time granularity of the traffic flows, measured at every single input and output in the database, for video, audio and data files, separately. Additionally, the performance is continuously monitored for analysis purposes using the D-ITG tool (Ruiz et al., 2018; Botta et al., 2013; Al-Haddad et al., 2021).

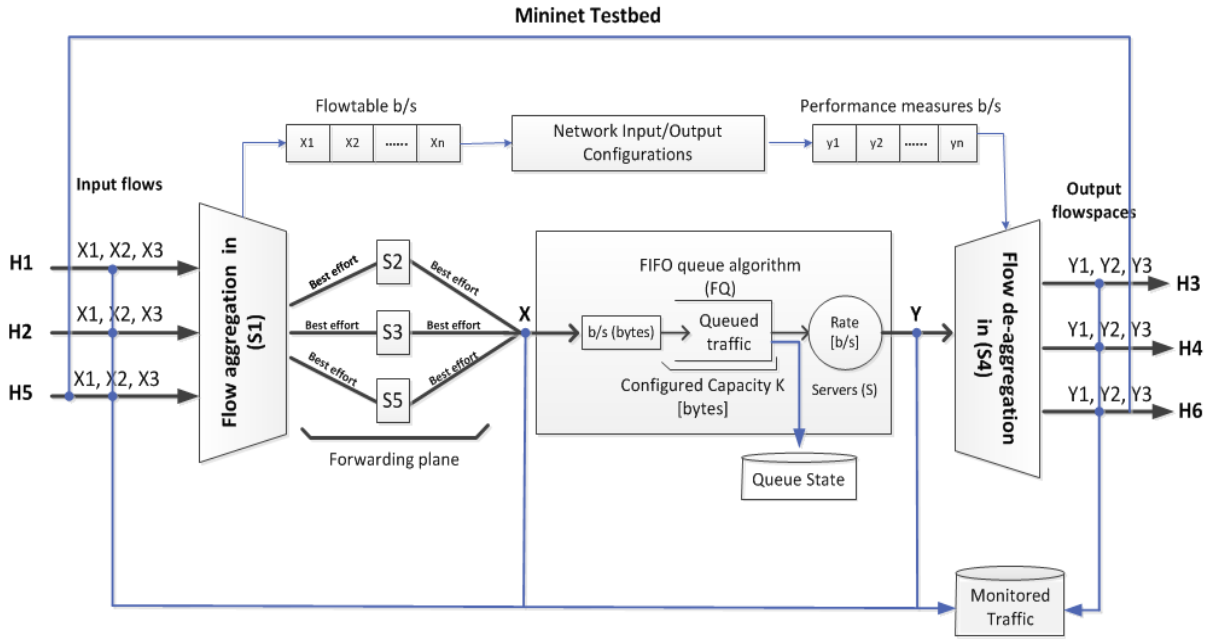


Figure (5-2) The data plane for the FIFO SDN model
(Al-Haddad et al., 2021)

The differentiated service code point protocol (DiffServ) (DSCP) (Baker et al., (1998)) has been chosen, because it can provide classes that are assigned to each queue within the slices and to provide soft and dynamic QoS guarantees by the use of queueing, thus enabling the routers to classify packets. The classification is made using the Differentiated Services Code Point (DSCP) to assign the value of the best effort for the packet headers for the network traffic. Settings as a default traffic treatment for the slice-types were set as follows in Table (5-1).

Table (5-1) Schematic representation of the DiffServ protocol for optimising performance under FIFO

Slice Configurations	**The Differentiated Traffic**
Video configured as the default class of service (S3)	Best effort DSCP level
Audio configured as the default class of service (S5)	Best effort DSCP level
Data configured as the default class of service (S2)	Best effort DSCP level

These three flows (video, audio and data) presented in Table (5-1) are directed to switch S4 (refer to Figure 5-2), where they compete for the maximum resources available in the bottleneck link between this switch and the hosts, due to the limited queueing under FIFO. Video, which requires the highest channel capacity (bandwidth), will suffer most degradation over the long term, although exceptions are seen where it has arrived first. So, control over the switches (S2, S3, S5) is manipulated to ensure that the principles of limited FIFO queueing are followed (Al-Haddad et al., 2021). The route traffic of the controlled slices defining flow space properties is illustrated schematically in Figure (5-3). The top half represents the properties for the network flow for each outbound and inbound packet during the routing process.

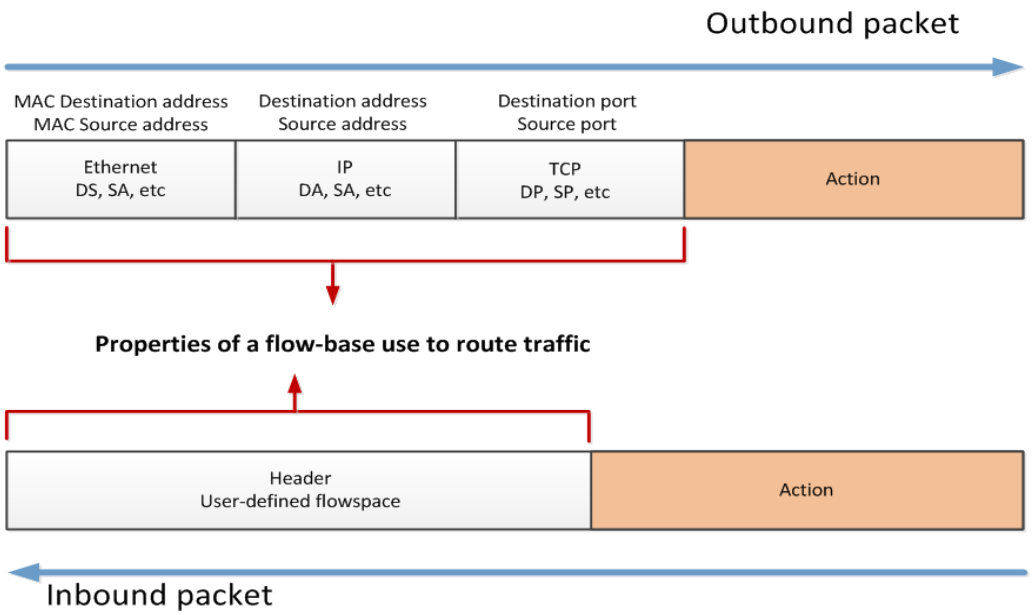


Figure (5-3) Flow space properties

The following code illustrates the methodology for implementing the FIFO algorithm in the SDN template design on the software platform mentioned in Subsection 5.2.2. (Al-Haddad et al., 2021)

Algorithm 5-1 Code for implementing the FIFO Algorithm: FIFO

Input:

P: packets received from the hosts

Bandwidth: maximum bandwidth for the queue

Output:

SP: sorted list of packets

BE_q: list of BE packets

while P ≠ ∅

for each packet pi in P **do**

SP→weight:= 0

SP→port:= 0

I_p←getPacketInfo(pi)

SP→ID = I_p→ID

```
            if  Ip→type == video then
                    SP→weight = 0
                    SP→port = 9999
                    SP→tag = video
                    SP→device = S3
                    SP→length = Ip→length
                    add_to_queue (pi, BE_q)
                    sort_queue (BE_q, DSC)
                    SP→bandwidth  =  allocate_bandwidth(Bandwidth, Ip→type)

            else if Ip→type == audio then
                    SP→weight = 0
                    SP→port = 8888
                    SP→tag = audio
                    SP→device = S5
                    SP→length = Ip→length
                    add_to_queue (pi, BE_q)
                    sort_queue (BE_q, DSC)
                    SP→bandwidth  =  allocate_bandwidth(Bandwidth, Ip→type)
            else
                    SP→weight = 0
                    SP→port = 1111
                    SP→tag = data
                    SP→device = S2
                    SP→length = Ip→length
                    add_to_queue (pi, BE_q)
                    sort_queue (BE_q, DSC)
                    SP→bandwidth  =  allocate_bandwidth(Bandwidth, Ip→type)
            end if
                    store_in_DB(SP)
                    forward_packet(SP, Bandwidth)
        end for
end while
------------------------------------------------------------
```

When a slice is created, bandwidth is normally set as fraction of it on each link (Sherwood et al., 2010). The reason for this is that in the Linux operating system, this is the kernel default packet-scheduling algorithm, which uses a class-blind queueing discipline and supports no internal subdivisions within classes. In other words, the forwarding of a packet is based on the arrival order. In contrast, class-based queueing discipline needs to determine which type of traffic must be forwarded into queues. This is done by using a flow table for the switches.

5.2.3. A testbed and experimental design for the FIFO algorithm module in sliced-SDN

The FIFO algorithm was implemented in the sliced-SDN system, by configuring the buffering to implement queue management. For systematic evaluation purposes, the Byte FIFO settings were configured for the limited queueing with maximum bandwidth settings at 40, 70, and 100 Mbps. The traffic was set up to contain a mixture of slices assumed to arise from different ports, as is explained in detail in Subsection 5.2.2. for the SDN FIFO model.

The control plane was built using three Floodlight controllers (Marschke, Doyle, and Moyer, 2015) to control the slices, by using a script developed for this purpose. Floodlight is a high-performing OpenFlow controller able to handle a large amount of equipment, while maintaining a high level of availability. It has a modular architecture to which modules can be easily added and fitted to those within the basic system. The default (basic) modules of Floodlight do not provide enough functionality to apply the proposed solution, for example, it has no monitoring module, so it was necessary to write an application to monitor the latency in the network. Floodlight was loaded with the monitoring application, which retrieved the latency measurements stored in a separate file or accessed via a REST API (Phemius and Bouet, 2013).

Floodlight1 is used to control only data slices via switch 1 to switch 2 and to the switch 4 path (S1-S2-S4), as shown in Figure (5-2). It does so by listening in port number 6635, which represents any TCP port number that could be available for use. In the internet protocol (these port numbers can be any number less than 65535), Floodlight 2 controls the video slices via switch 1 to switch 3 and to switch 4 (S1-S3-S4) listening in port number

6634, while Floodlight controller 3 controls the audio slice via switch 1 to switch 5 and to switch 4 (S1-S5-S4) listening in port number 6636. Each controller will see only these switches, i.e. S1, S2, S3, S5, then they are all routed to (S4). In addition, using the FlowVisor tool (Sherwood et al., 2010), the three control slices needed to control the different flows are created. Different flow spaces (Sherwood et al., 2009) created using FlowVisor are used to map packets to the slices in the topology shown in Figure (5-2). In addition, a general queuing module is defined that relates the aggregator that collects all the packets from all flows, the topology system and dis-aggregator, the model in Figure (5-2) allows a number X of input traffic flows that will be aggregated, queued, and finally de-aggregated into Y output traffic flows. FlowVisor creates a per-slice queue on each port (Sherwood et al., 2010). All this is a classless queueing discipline (best effort, i.e. there are no guarantees regarding the delivery of packets to their destinations that refer to a specific network).

The video control slice has to listen in port 9999, whilst the audio application slice has to listen in port 8888 and the other data or http application, i.e. the protocol used to send data over the network, such as emails, will listen in any other port. The control slices eventually look like other transmitted slices in most respects, but they have to be based on listening in different ports to enable the proposed limited queueing approach available under FIFO.

5.2.4. Timescale parameters

It should be noted that the time parameters are different in this research. Figure (5-4) shows a general explanatory diagram of all the time parameters, both those inherent to the timescales of the operation within the algorithm and those in the sample lengths and the scheme of rotation between the defined conditions. These timescale parameters are applied throughout all three switching algorithms of the simulation techniques, i.e. for FIFO, TS and QoSVisor.

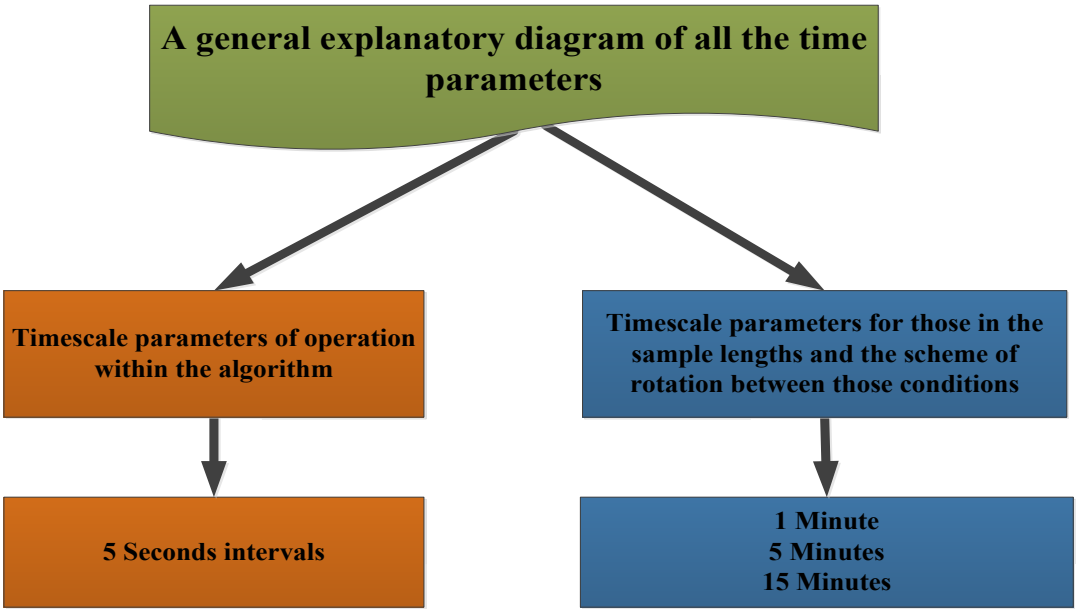


Figure (5-4) Timescale parameters

For 40, 70, and 100 Mbps, the 1-minute test run duration is not long enough in relation to typical traffic statistics to give reliable results, especially at low bandwidth. This generality is evident there is always going to be a dependence of pattern of performance on the flow characteristics, and the mixture of the material type of (video, audio and data) and brevity of sample will widen the variance to be expected. Whilst the expectations for this duration of the test run will, on average, be the same for all traffic types, there is good reason to include a short run duration in an evaluation design, as an illustrative reminder of the need for longer-term performance measures. The expected wider variation in the measures to be observed would reflect the chance properties of incident traffic, including educative extreme examples of success and failure of FIFO.

For the longer test run durations of 5 and 15 minutes for all the bandwidth conditions, as the packets in the queue build up, the performance data will reflect the lack of discrimination under FIFO regarding traffic type. Whilst this will lead to a sense of fairness in relation to queueing and transmission, it will not maximise efficiency from a range of perspectives. When the queue becomes full, bottlenecks will occur in the queue and so, some incoming packets will drop out. The consequences of this system with no prioritisation will affect all the performance characteristics, in particular, highly sensitive packet data, i.e. video. This summarises the differences in delay, jitter, and throughput, which aggregate to constitute

general quality of service. The ‘fairness’ of the FIFO algorithm is expected to be seen as only becoming a major limitation when network resources are limited. That is, when there is substantial bandwidth limitation, jitter-sensitivity is present, especially when there are different traffic types.

5.3. Summary

In this chapter, a FIFO algorithm has been developed and implemented in a sliced-SDN framework as a baseline condition for quantitative performance comparisons, with detailed implementation of the template design for the FIFO algorithm module in a sliced-SDN testbed also being provided. The particular traffic measures (throughput, delay and jitter), which are separate performance indices all contributing to QoS, and are integrated to provide an objectively rooted but overall and subjectively confirmable metric of QoS for each switch. Floodlight and FlowVisor controllers as well as OpenFlow (OF) switches, which are characteristic behaviour of SDN, have been modelled and simulated via a Mininet testbed emulator platform. A custom topology has been used with five switches, all the topology being connected to Floodlight and FlowVisor controllers, with the ingress bound interface switch (S1) being connected to three hosts. At the same time, the outbound interface switch (S4) is also connected to three hosts. The server(s) processed the queued data at a rate of 40, 70 and 100 b/s, as the parameters chosen for the designed study, representing the limiting speed of the outbound interface and ten replicate measurements were taken.

The simulation was run for different timescale parameters, both those inherent to the timescales of the operation within the algorithm for five seconds intervals for each test and those in the sample lengths as well as the scheme of rotation across the defined conditions for 1, 5 and 15 minutes (Al-Haddad et al., 2021). These timescale parameters were applied throughout all three switching algorithms of the simulation techniques: FIFO, TS and QoSVisor. The results will be presented in full in Chapter 8.

The next chapter describes the TS algorithm proposed to work as a bandwidth management technique to optimise performance in a sliced-SDN network, and to overcome the key limitation observed in FIFO queuing, namely buffer overflow. Traffic shaping works mainly via the WFQ part-function of the TS queueing mechanism, to reduce congestion and smooth traffic flow. This methodology is used for QoS does two things simultaneously: makes the

traffic conform to an individual rate using WFQ to decide the appropriate queue for each packet; and combines the methodology with buffer management that decides whether to put the packet into the queue according to the proposed algorithm defined for this purpose. In addition, a code used to implement the components of traffic shaping algorithms (packet tagging, queueing, forwarding to queues and allocation of bandwidth) was introduced and during this tagging, the bandwidth is managed based on Algorithm 6-2-A: Allocate Bandwidth (Al-Haddad et al., 2021).

6. Chapter Six: Traffic Shaping (TS) Algorithms in SDN

In this chapter, traffic shaping algorithms are proposed as a new contribution for the implementation of a Quality of Service (QoS) bandwidth management technique to optimise performance in a SDN-Sliced network. Two algorithms, namely "packet tagging, queueing, forwarding to queues" and "allocating bandwidth" are proposed and developed for implementing a weighted fair queuing (WFQ) technique as a new methodology in an SDN-sliced testbed. This methodology is used in QoS and does two things simultaneously, first, making traffic conform to an individual rate using WFQ to make the appropriate queue for each packet. Second, there is combining of the methodology with buffer management, which decides whether to put the packet into the queue according to the proposed algorithm defined for this purpose. Other sections present the system components and implementation overview, experimental system, WFQ components, the objective of the TS algorithm for sliced-SDN, testbed experimental tools, measured parameters, control plane development, managing the queueing time at network nodes and the code used to implement the components of the traffic shaping algorithms (packet tagging, queueing, forwarding to queues and allocation of bandwidth) in detail. This is followed by the chapter summary.

6.1 Introduction to Traffic Shaping for SDN-Sliced (TS) Algorithms

The Internet traffic flow in data networks is characterised by bursts of activity; the traffic arrives at not uniform rates and the overall traffic rates vary (Tannenbaum and Wetherall, 2011). Bursts in network traffic are more difficult to handle than constant-rate traffic, because they can fill the buffers, and with unmodified FIFO the consequences of this will be packet loss. The next algorithm explains the proposal, which includes "packet tagging, queueing, forwarding to queues" and "allocation of bandwidth" by implementing a weighted fair queuing (WFQ) technique as a new methodology in a sliced-SDN testbed. For distinctiveness of name and to convey this perspective, the author denotes this as 'Traffic Shaping for Sliced-SDN'. This algorithm involves a related set of sub-goals to be followed by sub-algorithms addressing the flow problems that arise when there is a high average rate and 'burstiness' of flow in the data that enters the network. The flow is identified and can be quantified as a

stream of a specified numbers of packets, based on the source and destination addresses in the header fields (Tanenbaum and Wetherall, 2011). Traffic shaping works as a bandwidth management technique for optimising performance in a sliced-SDN network, and to overcome the limitation observed in FIFO queuing, i.e. buffer overflow. Traffic shaping works via the queueing mechanism, thereby reducing congestion and smoothing traffic flow (Al-Haddad et al., 2021).

The relation between traffic shaping, and WFQ the part-function of traffic shaping for sliced-SDN is the new methodology implemented in the sliced-SDN testbed. This methodology, used in QoS, does two things simultaneously: making traffic conform to an individual rate using WFQ to decide the appropriate queue for each packet. It also combines with buffer management to decide whether to put the packet into the queue, according to the proposed algorithm defined for this purpose. In this way, the latency and congestion remain in check, thus meeting the requirements of real-time services. The main subsidiary principle of its implementation is the Differentiated Service (DiffServ) protocol, which is used to define classes (Al-Haddad et al., 2021). This is explained in more detail later in Subsection 6.2.2 (Experimental system). In a sense, QoSVisor (Chapter 6) is also involved in shaping of traffic, but the name is kept for the second TS algorithm to represent the step up in sophistication from FIFO, which is clearly a more descriptive name.

6.2. System Components and Implementation Overview

6.2.1. Weighted fair queuing (WFQ) for the traffic shaping algorithm

The general idea of traffic shaping is implemented by the next algorithm to be compared, namely weighted fair queuing (WFQ). WFQ is a technique used to accommodate the requirements of the most vulnerable classes of traffic type, while providing enough bandwidth for other traffic. This technique works by giving a ratio of bandwidth for each queue to tag the traffic according to different classes of traffic. This makes it possible to tweak the ratio, thus bringing variable prioritisation to the network traffic, and delivering a range of detailed forms of service (Shankar and Ambe, 2003), all of which could correctly be

called WFQ. WFQ gives the higher priority queue a higher weight so as to allow high priority packets through first. This priority allocation can even be done dynamically, e.g. if it is known that more video is to be expected in the evening, or in response to short-term demand. The weight of a flow represents the number of bytes per round. Here, the round is referred to as the one-way round (OWR) or round trip time (RTT) of a packet of data, as explained in more depth by Mirkovic, Armitage, and Branch (2018). It refers to the latency measures between two endpoints in order to measure the performance of many networked applications. In this way, the video flow can be given more bytes to give it more weight, so as to be served first, while other queues with different weights are transmitted during the same period, but more slowly, This technique treats the hosts with equal priority, but with different weights, i.e. WFQ only includes the bit ratio and has no other basis for prioritising.

In WFQ, the scheduling is done by assigning a bit weight to each flow, where each flow x is given a weight Wx as cost per bit of flow x in total line channel rate Cx (Stiliadis. and Varma, 1998), as in Equation (6.1):

$$W_x = \frac{1}{c_x} \text{ for } c_x \qquad \text{Where Cx>1.0} \qquad (6.1)$$

Then, using these bit weights (B), each queue will be transmitting packets in a correlated fair share of total line channel rate C (Stiliadis and Varma, 1998), as in Equation (6.2):

$$\text{WFQ/Queue}_x = \frac{B_x}{B_1 + B_2 + B_3 + \cdots + B_n} \times C \qquad (6.2)$$

6.2.2. Experimental system and WFQ components

In this study, WFQ can be seen as an element common to both the traffic shaping and QoSVisor (refer to Chapter 7) algorithms by coordination of processes operating under the principles of traffic flow classification EF, AF, and BE, in both of which there has to be a mechanism for queueing. However, only QoSVisor implements the next level of sophistication, that of prioritisation according to the type of traffic to handle the situation of network congestion, specifically, by permitting more sophisticated allocation of bandwidth, and that it reduces critical delays of video traffic and to some extent, audio as well.

To avoid or manage this situation, the focus is on designing and testing the WFQ method given its simplicity in implementation with the DiffServ protocol of both traffic shaping and QoSVisor in the new SDN software environment to handle congestion, thus finding out whether the TCP protocol can be generalised to optimise QoS dynamically to reach scalable network systems.

This algorithm TS is proposed to tackle the single queueing limitation in the FIFO algorithm, and to give multimedia applications a reasonable QoS level. Using WFQ disciplines in a sliced-SDN context to solve congestion problems is proposed, especially when large flows fill the buffer quickly and cause packet dropping in other flows. The research involved developing the experimental system for the TS algorithm using multiple weighted queues. In fact, three different queues were created, with a specific weight given for each one, such that delay-sensitive classes of traffic are taken into consideration. The Differentiated Service (DiffServ) protocol is another important element, which is used to define classes in order to make network traffic patterns more sensitive to the traffic class, by specifying precedence for each traffic type (Al-Haddad et al., 2021). This is done in the IP packet header IPv4 (Layer 3 in the OSI reference model) (Day and Zimmermann, 1983), in a field known as Type of Service (TOS) or TOS Byte, which is used to assign the QoS level, according to the delay, throughput and the reliability level of the network. The header includes an 8-bit IP Type of Service (TOS) field, which is composed of 3-bit precedence for TOS, and also, two unused bits, which are always zero. This precedence pertains to the Diffrenciated services (DS) the priority for the traffic, whilst the TOS bits are reserved for the preference for throughput, delay and loss (Wang, 2001).

The flows are controlled by the class selector per-hop-behaviour (PHB) to elicit the forwarding behaviours in the switch (S1 the network edge) (Kulhari and Pandey, 2016) to apply the QoS treatment, such as queuing, TS or traffic policing in the switch setup. This is used to design the SDN system topology which is defined as shown in Figure (6-1/ a, b, c and d). The figures illustrate schematically the structure of the processes used by this algorithm.

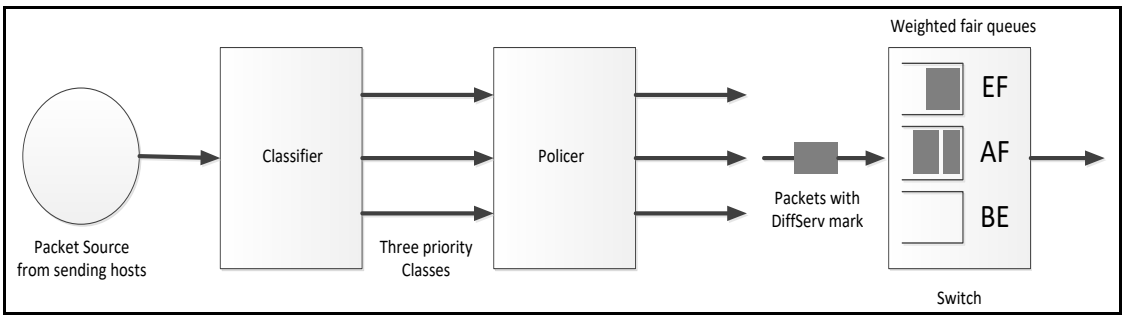


Figure (6-1) (a) The implementation of the DiffServ protocol

The Figure 6-1(a) shows that the packets from flows are classified and marked in the DS field in the IP header. The DS field contains the 6-bit DSCP value, which is used to replace the ToS field in IPv4. The protocol DiffServ is able to classify the flows into three service statuses, whilst the classifier and policer enforces an assignment to each flow and queues the resource managements under WFQ by a coordination of processes operating under the principles of BE, EF and AF.

The classifications made for the first ingress switch (network edge) allow for more information to be made available about each packet (to define management between flows), such as which sending host it comes from and what packets belong to which flows. The packets marked with expedited forwarding (EF), as shown in Figure 6-1 (b), will receive special forwarding treatment within a whole system of traffic flow and this special treatment is defined in terms of a minimum departure rate of the aggregate packet from any DS node that the actual rate must equal or exceed. This happens independently from what happens to any other traffic attempting to transit the node (Wang, 2001), thus allowing for the packets to incur low delay, low loss and low jitter. The recommended DSCP for EF is 101110B (46 or 2EH), which is selected for video (Acharya, Dutta and Bhoi, 2013). To unpack this code, this DSCP means that it is backward compatible with an IP precedence value of six digits, as seen in the binary pattern: 101110 = DSCP 46, or binary values-32 16 8 4 2 1 (in the figures below, the packet traces were obtained using Wireshark).

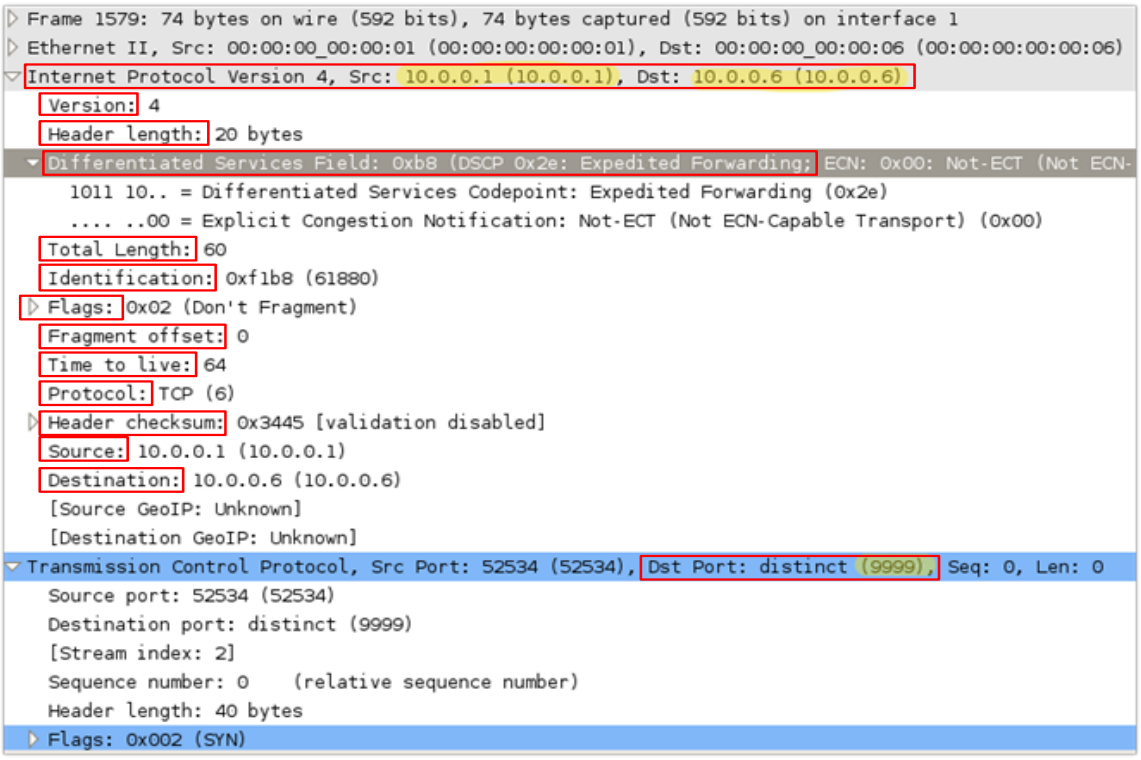


Figure (6-1) (b) Expedited forwarding (EF) implementation in the SDN system

The key decision steps for assured forwarding (AF), as shown in Figure 6-1 (c), are made in the ingress switch to define management between flows, which this is done according to four priority classes, each having its own resources. They might be called gold, silver, bronze, and standard. This technique also defines three discard levels for packets that congestion may present concurrently: low, medium and high. In this system, Class 2 is defined as shown in Table (6-1). In this example, AF21 (DSCP18) (see Table 6-1) is selected as having low drop probability only for packets and belonging to audio flow and DSCP 18 = 010010 in binary, while BE(00) for other data is default forwarding = DSCP 0 = 000000.

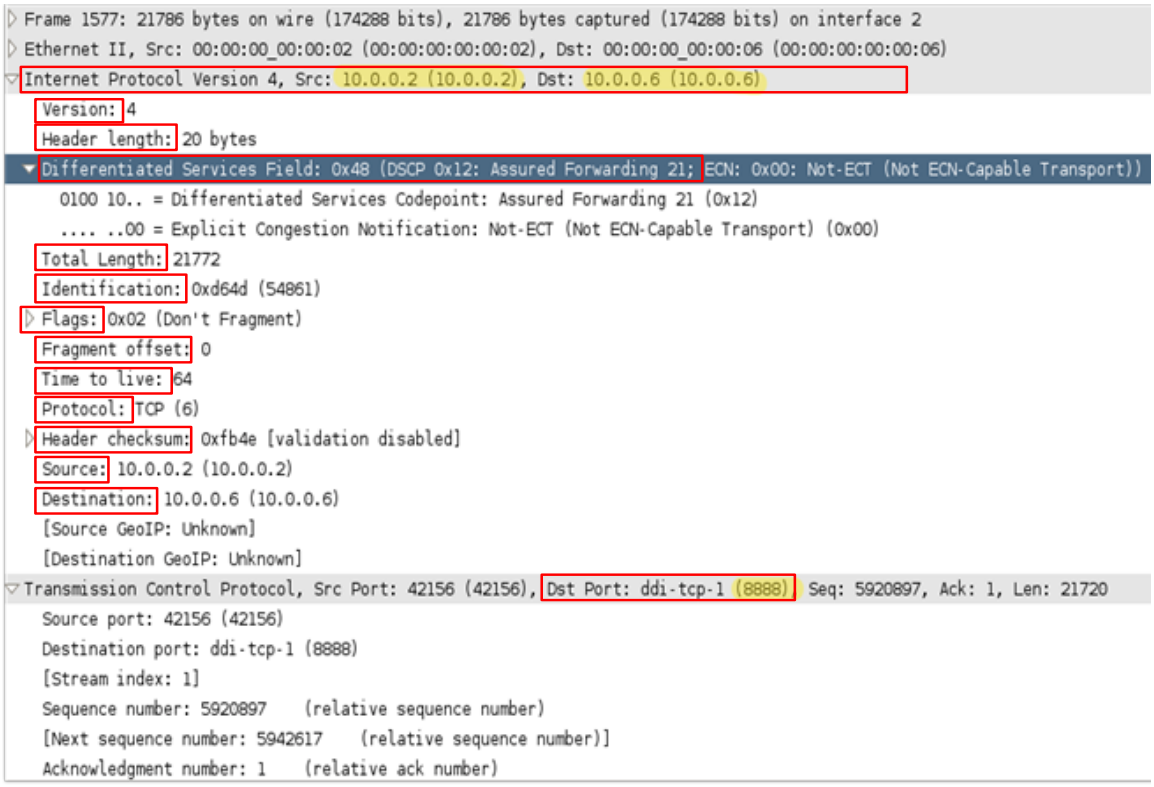


Figure (6-1) (c) Assured forwarding (AF) implementation in the SDN system design

Table (6-1) shows an example of standard priority levels classification for assured forwarding (AF) (Acharya, Dutta and Bhoi, 2013)

Discard level	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Low drop probability	AF11 (DSCP 10)	AF21 (DSCP 18)	AF31 (DSCP 26)	AF41 (DSCP 34)
Medium drop probability	AF12 (DSCP 12)	AF22 (DSCP 20)	AF32 (DSCP 28)	AF42 (DSCP 36)
High drop probability	AF13 (DSCP 14)	AF23 (DSCP 22)	AF33 (DSCP 30)	AF43 (DSCP 38)

While Figure 6-1 (d) shows thedefault forwarding (BE) classified as the best effort forwarding, where none of the listed classes above is used and the DSCP for the default forwarding is 0.

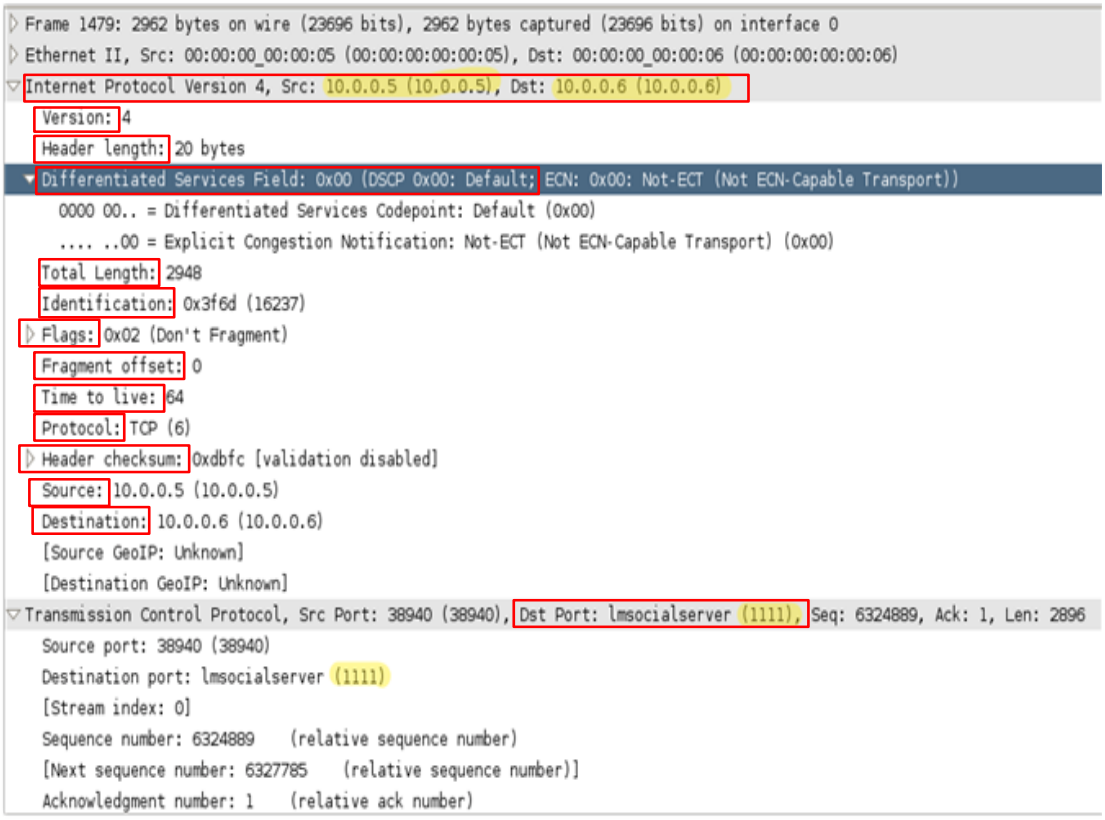


Figure (6-1) (d) Best effort forwarding (BE) implementation in the SDN system

6.2.3. Implementation of the WFQ method

In order to provide high quality of service, TS enforces an assignment to each flow and queues the resource managements under WFQ by a coordination of processes operating under the principles of BE, EF and AF. The way in which this is done is by having five switches, as shown in Figure (6-2).

The figure represents the specific setup of the SDN system implementation and development of the TS algorithm for this study. The algorithm template design consists of: topology links, matching, buffer and access permission, forwarding, routing, and queuing. Below, a detailed explanation is provided for each of these.

- Topology Links: The links in the three different colours are intended to show the three flows that have been previously configured. The figure represents the different packets arriving from different ports to the switch (S1), which will handle the flow rule actions and hence, the flow statistics, to match the packets (Al-Haddad et al., 2021).

- Matching: The matching and routing process is based on the port numbers. Port 9999 is for video flow, as shown in Figure 6-1 (b), and port 8888 is for audio, as shown in Figure 6-1 (c), whilst the data flow listens in port number 1111, as shown in Figure 6-1 (d). Each host (H1, H2, H5) in the network edge works as a client, sending a mixture of flow packets of video, audio and data. When they arrive at the switch (S1), they are forwarded by the class selector (Kulhari and Pandey, 2016), according to per-hop-behaviour (PHB) (Al-Haddad et al., 2021).

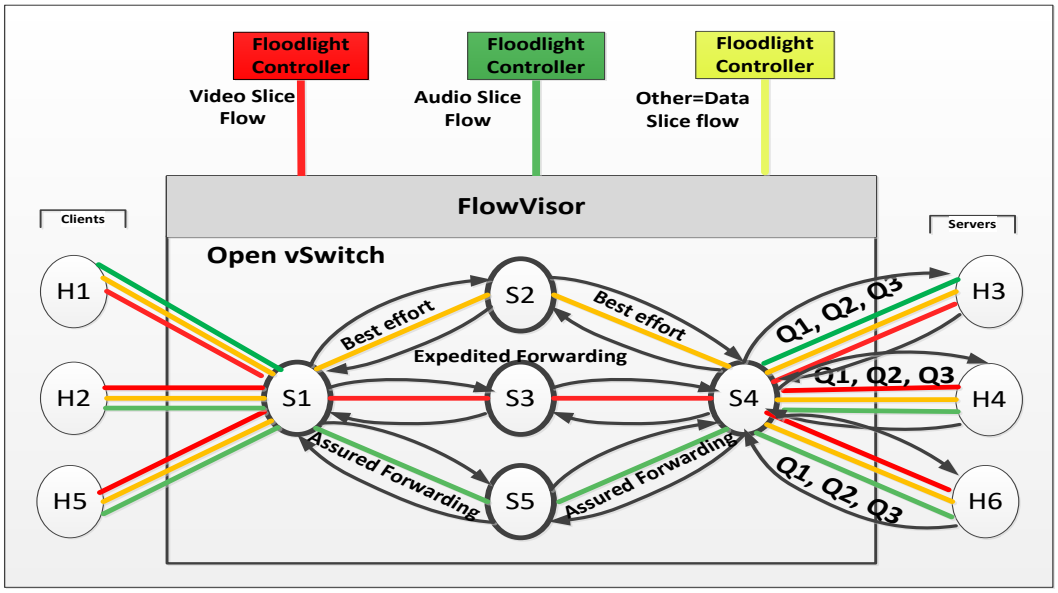


Figure (6-2) Traffic-shaping algorithm: template design linking the data and control planes (Al-Haddad et al., 2021)

- Buffer and access permission: The arriving packet is buffered and the packet header is checked each time against the rules in the flow table for matching purposes. The access permission for each slice in the flow space is assigned (Appendix 1) based on the permission number (7-bitmask value), set for this purpose between FlowVisor and the switches, whilst the bitmasked-set is used to select metadata updates (Bosshart et al., 2013). The individual permissions are READ, WRITE, AND DELEGATE= (2+4+1=7, respectively), to allow for all the slices to read, write and delegate in the flow space, according to the configuration between the switches and the controllers (Al-Haddad et al., 2021)

- Forwarding: Packet forwarding can be done according to these rules for the slices that belong to the flow. If it not matching, then, the packet will either be dropped or it will be sent to the Floodlight controller for processing, according to the flow rules (Al-Haddad et al., 2021). Subsequntly, it will be sent to the switch, where it will be stored to use again on any similar occasion. It, thus, implements the network policy without needing to go back to the Floodlight controller for a decision (Kuźniar et al., 2015).

- Routing: The switch decides the routing, depending upon the predefined routing paths for the port number for each arriving flow, and this is based on their OpenFlow specification. Data flows between S1-S2-S4 as the routing path, while the video flow is assigned between S1-S3-S4, and finally, the audio flow between S1-S5-S4 (Al-Haddad et al., 2021). Table (6-2) shows the detailed topology configurations.

Table (6-2) Predefined routing paths

Switch number	**Switch port configurations**	**Routing traffic path**	**Traffic classes**
S2	s2-eth1 & s2-eth2	S1-S2-S4	00 Best Effort (BE)
S3	s3-eth1 & s3-eth2	S1-S3-S4	46 Expedited Forwarding (EF) PHB
S5	s5-eth1 & s5-eth2	S1-S5-S4	18 Assured Forwarding (AF) PHB

- Queuing: To implement minimal queue management, a minimum buffering rate is defined for each queue. In switch S4 three queues for each output interface between it and the three hosts H3, H4 and H6 are configured (Al-Haddad et al., 2021). The detailed notations for the elements of WFQ in an SDN network system are provided in Table (6-3), which shows these queues.

Table (6-3) Weighted fair queueing for the traffic shaping algorithm

Maximum Rate	**No. of Queues**	**Minimum Rate per Queue**	**Ports configurations**	**Flow type**
40 Mbps	Q1, Q2, Q3 No priority for any queue	Q1= 20 Mega Q2= 2 Mega Q3= 200 Kilo	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data
70 Mbps	Q1, Q2, Q3 No priority for any queue	Q1= 45 Mega Q2= 4.5 Mega Q3= 450 Kilo	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data
100 Mbps	Q1, Q2, Q3 No priority for any queue	Q1= 50 Mega Q2= 5 Mega Q3= 500 Kilo	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data

A variant of WFQ is used on the three slices, with the weight configured proportional to the dynamic allocated bandwidth for each slice (parameters values have been set up by the researcher for this study and they are not standard). EF traffic receives higher bit weight by giving the highest bandwidth allocation to video and then, AF receives the second highest bit weight, with the BE flow being allocated the third. Hence, the scheme works according to a ratio principle, as shown in the table above, with the ratio of 20 Mega /2 Mega /200 Kilo being used for a 40 Mbps link capacity. Similarly, for 70 Mbps the ratio is 45 Mega /4.5 Mega /450 Kilo, while for 100 Mbps, it is 50 Mega /5Mega /500 Kilo (Al-Haddad et al., 2021).

6.2.3.1. The objective of the TS algorithm for sliced-SDN

The objective of TS algorithm is to measure the delay for a video slice (EF flow) without specifying any particular priority scheme. In other words, the EF flow will experience less delay than AF and BE, with the queuing delay being distributed, while the bandwidth allocation remains fixed. This scheme works as a conservation (sparing) scheme, because the AF flow can use the remaining bandwidth that is left from the EF flow, so long as the traffic is within the maximum capacity of the link (Al-Haddad et al., 2021).

6.2.4. Testbed experiment tools and the parameters measured

The technical contribution of this study is the devised set of measurements (parameters) for each algorithm to evaluate them. Developing a testbed that enables implementation of the algorithms and allows for uniformly getting measurements is necessary. The testbed is important for enabling the statistical strategy for constraining the tests of reliability of effects and contrasts (See Chapter 8 data analysis). A novel analysis plan has been proposed for the(short, middling and longest duration. The longest duration, in general, tends to have the most reliable results for the effects of bandwidth, algorithm etc. There is some reason to believe that the middling and longest durations could provide some patterns of results, but also, there is a possibility that he shorter duration might reveal more critically. Therefore, more sensitively, the algorithm, or algorithm by traffic type interactions effects predicted, and testbed system to prove that, is therefore developed. (More to follow in Chapter 8)

Tables (6-4) (A and B) show the testbed details, including the hardware and software specifications used to build the testbed, respectively.

Table (6-4) A. Hardware

A. Hardware	**Specifications**
PC/ Processor	Intel® Core (TM) i7-3770 CPU @ 3.40 GHz
RAM	16 GB
O.S	64-bit Operating System

Table (6-4) B. Software specifications of the testbed development

B. Software/Tool	**Specifications**
VirtualBox	Version 4.2.16
VM Mininet 2.2	O.S Ubuntu Linux version 14.04.4 LTS
FlowVisor (special purpose controller)	Version 1.4.0
Floodlight controller	Version 1.2
Mininet	Version 2.2.1 Ubuntu 14.04.4 LTS /64-bit
MobaXterm toolbox	Version 10.4
Open vSwitch	2.0.2
OpenFlow	Version 1.2
Wireshark tool	Version 1.10.6
Iperf tool	Version 2.0.5
D-ITG tool	Version 2.8.1
VLC media player tool	Version 2.1.6

The testbed comprises a PC with processor Intel® Core (TM) i7-3770 CPU @ 3.40GHz with RAM 16 GB, and a 64-bit Operating System. VirtualBox (Mohan and Damodaran., 2012) is used as a kernel-based virtual machine (KVM), which provides a virtualisation infrastructure for the Linux kernel that turns it into a hypervisor (White and Pilbeam, 2010; Desai, et al., 2013) to install and create the virtual switches. In addition, the remote desktop MobaXterm (Available at: http://mobaxterm.mobatek.net/) version 10.4 toolbox (lightweight application) was used to provide features for remote computing, including an embedded X server fully configured through SSH for secure transport and a tabbed terminal with an SSH client based on PuTTY, which is a free software application that can make an SSH connection to your server; the latter can provide network tools (Hung et al., 2016).

The Iperf tool is used to generate network traffic and to test throughput and jitter using TCP traffic. The D-ITG tool **(**Distributed Internet Traffic Generator) (Botta et al., 2007) is used as a platform, which can also produce IPv4 and IPv6 traffic by accurately replicating the workload of current Internet applications. At the same time, D-ITG is also a network measurement tool able to measure the most common performance metrics (i.e. throughput, delay, jitter). It does this by using two different separate components called ITG-Send and

ITG-Receive. D-ITG is designed to run on both Linux and Windows platforms (Kolahi et al., 2011) at the packet level. To create the network topology, open source open flow virtual switches (Open vSwitches-OVS) (Pfaff et al., 2015) are used as the main switching type for the measurements (in the Control plane). To emulate the network topology, a Mininet testbed emulator was used to control the virtual environment represented by the virtual switches mentioned above that were designed for this project (Maugendre, 2015). To test the delay, average jitter and throughput the following formulas are used.

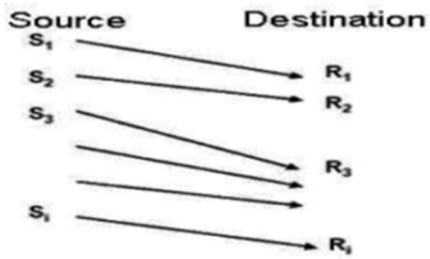
To calculate the delay: The values are computed according to the timeline records (Botta et al., 2013), as in Equation (6.3):

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(di - \hat{d})^2} \qquad (6.3)$$

Equation (6.3) Delay calculation formula

where, N is the number of packets being considered, "di" is the delay of packet i and $\hat{d}$ is the average delay of the packets.

The Jitter is computed according to the formula below (Equation 6.4), where S_i and R_i correspond to txTime and rxTime, respectively.



$$D_i = (R_i - S_i) - (R_{i-1} - S_{i-1})$$
$$D_i = (R_i - R_{i-1}) - (S_i - S_{i-1})$$

$$AVGJitter = \frac{\sum_{i}^{n}|D_i|}{n} \qquad (6.4)$$

Equation (6.4) Jitter calculation formula (Botta et al., 2013)

The Throughput is computed according to the formula below (Equation 6.5), where w_n is the weight assigned to flow n.

$$Throughput\ _n = \frac{Available\ Capacity}{\sum_{i=2}^{n} Wi} . w_n \qquad (6.5)$$

Equation (6.5) Throughput calculation formula (Balogh and Medvecký, 2010)

6.2.5. Implementation of the FlowVisor slicing tool and floodlight controllers in the control plane

Any SDN network requires having both a control and a data plane, the latter being represented by the controllers and switches. In this set-up, the control plane comprises the Floodlight controller and FlowVisor special purpose controller. The intended interfaces of the data plane are represented by the configured switches in the customised (i.e. adapted specially for the context) topology, using the OpenFlow communication protocol to handle the packets (Kuźniar et al., 2015). The OpenFlow protocol (McKeown et al., 2008) works in a reactive mode (Fernandez, 2013); version 1.2 was selected in preference over the other four versions (1.1), (1.2), (1.3), and (1.4), because this version is compatible with the SDN controllers used in this research. FlowVisor is used to create the slices and Floodlight is set to control them.

The FlowVisor tool (Sherwood et al., 2009) is used to create controlled slices, only three in total, because the aim is to test the QoS based on three types of traffic (for audio, video and data) and to implement the queuing system for these. Sherwood et al. (2009) defined the slices as "a set of flows that make up a data path which can be thought of comprising a well-defined isolated space of the entire geometric space of feasible packet headers". Also, slicing can be defined as specifying the six-way relationship between: (a) the packet contents, (b) the switch port number assignments to match the physical port (layer 1), (c) the source/destination Ethernet protocol type (layer 2), (d) the source/destination IP addresses,

(e) the Ethernet type (layer 3), and (f) the source/destination assignments, so as to match the transport layer for the TCP/UDP port (layer 4), as shown in Figure (6-3) (Azodolmolky, 2013).

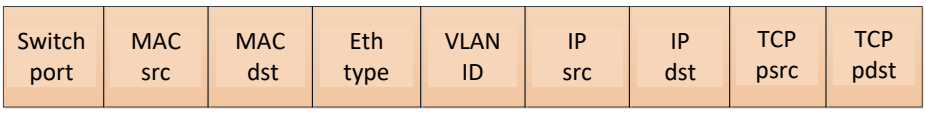


Figure (6-3) The packet header fields (Azodolmolky, 2013)

The above described system uses FlowVisor and Floodlight controllers to build the control plane (Belter et al., 2014). A custom topology is then built, based on separate scripts, consisting of five nodes, each occupied by an OVS switch (S1, S2, S3, S4, S5), with a specific role. The six hosts necessary to define a network of sufficient complexity for generating the problems that the present set of switching algorithms address are H1, H2, and H5 (the upstream 'clients') and H3, H4 and H6 (the downstream servers). Moreover, three different flow spaces are created, with three slices being created to control the flows of each type of traffic. The video application slice has to listen in port 9999, whilst the audio application slice has to listen in port 8888 and the other data or http application will listen in port 1111. Because they are slices, they have to listen in different ports and this is what enables the queueing approach proposed in this study.

In the implementation, three Floodlight controllers are used to control the slices, with full isolation from the network traffic, by using a set of scripts developed for this purpose. As shown in Table (6-5), Floodlight 1 controls only data slices passing through S1-S2-S4, by listening in port number 6635, whilst Floodlight 2 controls only the video slice through S1-S3-S4 by listening in port 6634 and floodlight 3 controls only the audio slice through S1-S5-S4 by listening in port 6636. Each controller will see only these switches when the flows come together again in S4. Thus, S4 can be considered as the output interface and it can be seen how it performs the TS by having three queues in this interface. However, it would be possible to configure up to eight queues at switch port level, according to different required QoS levels.

As for the main pipe, the egress port is set to three different levels (40, 70, and 100 Mbps) for the purpose of the present evaluative comparisons, whilst performance under

conditions of traffic congestion are assessed by simulation with three hosts directing traffic towards a single host. Whilst other arrangements are feasible, this one is the simplest most general for implementing and evaluating the three types of queueing system of this study.

Controllers	Ports	Slices
FlowVisor	6633	Slicer
Floodlight	6634	Video
Floodlight	6635	Data
Floodlight	6636	Audio

Table (6-5) Controller plane configurations

Table (6-5) above shows the allocated assignments between the control plane (Floodlight controller and FlowVisor controller) and the intended interfaces of the data plane, represented by the configured switches in the custom topology.

The switch configures the flow according to the sets of flow rules in the flow table and then, forwards the packets, according to the header fields defined to assign the DSCP classes for the packets, as explained earlier (Subsection 5.2.3. A testbed and experimental design for the FIFO algorithm module in sliced-SDN). The Differentiated Service (DiffServ) (DSCP) protocol (Baker et al., 1998; Blake et al., 1998) has been chosen, because it can provide classes, these classes used to assign a class for each queue within the slices, also, to provide soft and dynamic QoS guarantees by the use of queueing.

6.2.6. Managing queueing time at the network nodes

For all classes of traffic, as previously discussed, the average delay time in the switch, throughput for performance and jitter are the most important considerations, for this will result in minimising the queue length at the network nodes. Equations are used to express the functional relationships over time for the queueing approach implemented in the TS algorithm using WFQ (Al-Haddad et al., 2021). Figure 6-4 below, shows schematically how queues are handled as they arrive at a downstream server host. The scheduling involved underlies the equations, which express the delay times inevitably accruing and this more sophisticated algorithm avoids delays and jitter, thus raising QoS.

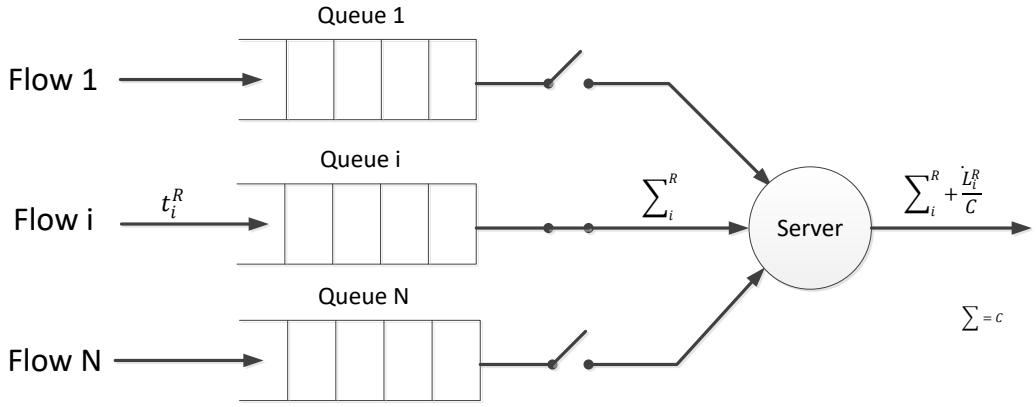


Figure (6-4) Model of the network nodes used to manage queuing time in the SDN system design (Dekeris et al., 2006)

This model was developed in 2006 by a group of software engineering researchers (Dekeris et al., 2006) in order to ensure QoS for real time services. It is generally used as a method to describe delay time in the queues for network nodes using WFQ for video and other types of applications.

The theory description for the WFQ queuing method is expressed as follows.

t_i^R is the time in seconds when the last bit of a packet pRi (R packet of i flow) arrives at the queue; L_i^R is the length of packet P_i^R; and $\sum_i^R$. is the time in seconds, when the last served bit of packet P_i^R has been sent (Dekeris et al., 2006, Al-Haddad et al., 2021).

In this case, the queueing time of the packet P_i^R can be expressed as:

$$P_i^R = \sum_i^R . t_i^R \qquad (6.6)$$

Depending on the queueing scheduling discipline, the delay time $P_i^R(t)$ in the whole system of packets from flow i $\sum_{R=0}^{\infty} L_i^R$ for time and length of queue $\int(t - L_i^R)$ for different classifications [EF, AF, BE] can be expressed by:

$$P_i^R(t) = \sum_{R=0}^{\infty} L_i^R \cdot \int (t - L_i^R) \qquad (6.7)$$

here: $\delta(t)$ – is Dirac's delta function (Khuri, 2004). Then, the function of the common queueing time in the network node is:

$$Pi(s;t) = \int_s^t pi(t)\delta(t) \qquad (6.8)$$

Code used to implement the components of traffic shaping algorithms: (packet tagging, queueing, forwarding to queues and allocation of bandwidth) (Al-Haddad et al., 2021)

Algorithm 6-2: Packet Tagging and Forwarding

Input:

P: packets received from the hosts

Bandwidth: maximum bandwidth for the queue

Output:

SP: sorted list of packets

EF_q: list of EF packets

AF_q: list of AF packets

BE_q: list of BE packets

while P ≠ ∅

for each packet pi in P **do**

SP→weight:= 0

SP→port:= 0

I_p←getPacketInfo(pi)

SP→ID = I_p→ID

if I_p→type == video **then**

SP→weight = 46

SP→port = 9999

SP→tag = video

SP→device = S3

SP→length = I_p→length

add_to_queue (pi, EF_q)

sort_queue (EF_q, DSC)

else if I_p→type == audio **then**

SP→weight = 18

SP→port = 8888

SP→tag = audio

SP→device = S5

SP→length = I_p→length

add_to_queue (pi, AF_q)

sort_queue (AF_q, DSC)

else

SP→weight = 0

SP→port = 1111

SP→tag = data

SP→device = S2

SP→length = I_p→length

add_to_queue (pi, BE_q)

sort_queue (BE_q, DSC)

end if

store_in_DB(SP)

forward_packet(SP, Bandwidth)

end for

end while

During this tagging, the bandwidth is managed in S4 based on algorithm 6-2-A below (Al-Haddad et al., 2021).

Algorithm 6-2-A: Allocate Bandwidth

Input:

QT: queue type

Bandwidth: maximum bandwidth for the queue

Output:

Allocated_bandwidth: bandwidth in Bs (bits)

```
if Bandwidth == 40 then
    if  QT == video then
        Allocated_bandwidth = 20000000
    else if QT == audio then
        Allocated_bandwidth = 2000000
    else if QT == data then
        Allocated_bandwidth = 200
    end if

else if Bandwidth == 70 then
    if  QT == video then
        Allocated_bandwidth = 45000000
    else if QT == audio then
        Allocated_bandwidth = 4500000
    else if QT == data then
        Allocated_bandwidth = 450000
    end if

else
    if  QT == video then
        Allocated_bandwidth = 50000000
    else if QT == audio then
```

```
            Allocated_bandwidth = 5000000
        else if QT == data then
            Allocated_bandwidth = 500000
        end if

end if

return Allocated_bandwidth
------------------------------------------------------------
```

6.3. Summary

In this chapter, the TS algorithm has been proposed as new contribution to work as a bandwidth management technique to optimise performance in a sliced-SDN network, and to overcome the key limitation observed in FIFO queuing, namely buffer overflow. Traffic shaping works mainly via the WFQ part-function of the TS queueing mechanism, to reduce congestion and smooth traffic flow. This methodology is used for QoS and does two things simultaneously: making traffic conform to an individual rate using WFQ to decide the appropriate queue for each packet; and combining the methodology with buffer management that decides whether to put the packet into the queue according to the proposed algorithm defined for this purpose. Also, a code used to implement the components of the TS algorithms (packet tagging, queueing, forwarding to queues and allocation of bandwidth) was introduced and during this tagging, the bandwidth is managed based on Algorithm 6-2-A: Allocate Bandwidth. In this way, the latency and congestion remain in check, thus meeting the requirements of real-time services. All the experimental results conducted will be presented in depth with the corresponding statistical analysis in Chapter 8.

Chapter 7 that follows describes a new methodology of QoSVisor and a Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for video, audio and data over TCP SDN-Slicing networks has been developed, implemented and tested. New more advanced queueing algorithms are developed based on the packet tagging and forwarding, tagging and queueing, and packet scheduling algorithms in order to overcome the limitations that exist with the traditional FIFO queueing method. QoSVisor is aimed at reducing the delay and

jitter for the packet transmission process when compared with the standard algorithm FIFO, whilst at the same time maintaining or even improving the results obtained from the TS algorithm introduced here in Chapter 6.

7. Chapter Seven: QoSVisor Algorithm in SDN

In this chapter, a new methodology of QoSVisor and a Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for video, audio and data over TCP SDN-Slicing networks has been developed, implemented and tested. New more advanced queueing algorithms are developed based on the Packet Tagging and Forwarding, Tagging and Queueing, and Packet Scheduling algorithms in order to overcome the limitations that exist with the traditional FIFO queueing method. QoSVisor is aimed at reducing the delay and jitter for the packet transmission process when compared with the standard FIFO algorithm, whilst at the same time maintaining or even improving the results obtained from the traffic shaping algorithm introduced in Chapter 6. Subsequent sections present the implementation of these algorithms via a local weighting function, which sorts packets by the port and flow policy setup. The Action Filter Classifier and the Action QoS Manager with Congestion Management technique LLQ (Low Latency Queuing) are introduced throughout this chapter, which represent the further detailed stages of the QoSVisor framework. The chapter concludes with a summary.

7.1. Development Overview of the QoSVisor Algorithm

The development and testing of QoSVisor and the Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for different multimedia applications over TCP SDN-Slicing networks is aimed at enabling the deployment of more advancd queuing systems to meet the QoS of critical applications by filtering pre-classified packet data and prioritising those packets for delivery. OK? New more advancd queueing algorithms are developed based on the Packet Tagging and Forwarding, Tagging and Queueing, and Packet Scheduling algorithms, which have to meet the QoS requirements of video traffic and to some extent, audio as well, whilst also being suitable for network devices. The expected benefits of a 2-stage VISOR algorithm over a 1-stage prioritisation algorithm are explained. This research is focused on the internal development and enhancement of FlowVisor, with the aim of ensuring slicing in the production network. This would mean guaranteed and more precise QoS for different applications to continue running, even during peak congestion times, and

with priorities able to be agreed for specific requirements. The proposed design is based on gathering four main technologies together: traffic engineering (TE), SDN, network hypervisors and network slicing. A new improved model is proposed for enhancing the current FlowVisor, thus meeting the requirements of improving the QoS classification within a sliced-SDN. The objective of the proposed QoS model, which is a "Packet Tagging Prioritisation Agent (PTP Agent)", is to extend the current internal operation of FlowVisor. The algorithm QoSVisor is aimed at reducing the delay and jitter for the packet transmission process when compared to the standard algorithm FIFO, whilst at the same time maintaining or even improving upon the results obtained from the traffic shaping (TS) algorithm. The proposed algorithm employs similar techniques to those used in the TS algorithm, with some changes and enhancements that take into account the limitations of network devices.

The methodology of the QoSVisor algorithm and Packet Tagging Prioritisation Agent (PTP Agent) extension is proposed to measure to what extent the intended benefits of using slicing and queueing discipline established in WFQ in SDN are realised in practice as well as to implement and show adherence to the strict priority policy added to this algorithm. There is no great mystery about the testing of hypotheses concerning what the new algorithm(s) should achieve, because they have been proposed and developed as a result of clear understanding of the problems met with simpler systems.

7.2. Local Weighting Function for QoSVisor Workflow

In this section, the essential ingredient of QoSVisor workflow is described, which is the labelling (tagging) of some packets as delay-sensitive (for example, video and to some extent, audio), whilst others are not. The implementation of this distinction is via a local weighting function, which sorts packets by port and flow policy setup. Clearly, this enables the use of more advanced queueing systems, for which more than one stage is necessary, plus extra memory capacity to handle long queues. The implementation of this sophistication has to start from standard transmission data about each packet (flow, packet ID and weighted tag) and the enhancement of a priority scheme, which governs placement in subsequent queues. The weighted tag is the weighting assigned to the queue based on the traffic type, which is derived from the elements in WFQ (refer to Section 6.2: System components and implementation overview.

Most of the WFQ traffic-shaping for the sliced-SDN algorithm operates within a local weighting function in the data plane and is independent of the FlowVisor controller, being incorporated into its request validation mechanisms, which will be verified internally according to the permissions database. The new architecture of the QoSVisor algorithm has added features beyond those in the WFQ traffic-shaping for the sliced-SDN algorithm. Crucially, separate queues are supported for the three content-types (video, audio, and data) and the available bandwidth is allocated dynamically according to priority. Thus, at the expense of some actual overall delay, an apparent 'look-ahead' capability is created, in which the performance aims can be respected and hopefully, achieved via systematic prioritisation. Figure (7-1) shows the first stage operations underlying the QoSVisor workflow. Detailed explanations on how QoSVisor works are presented in Subsections 7.3.1, 7.3.2 and 7.3.3.

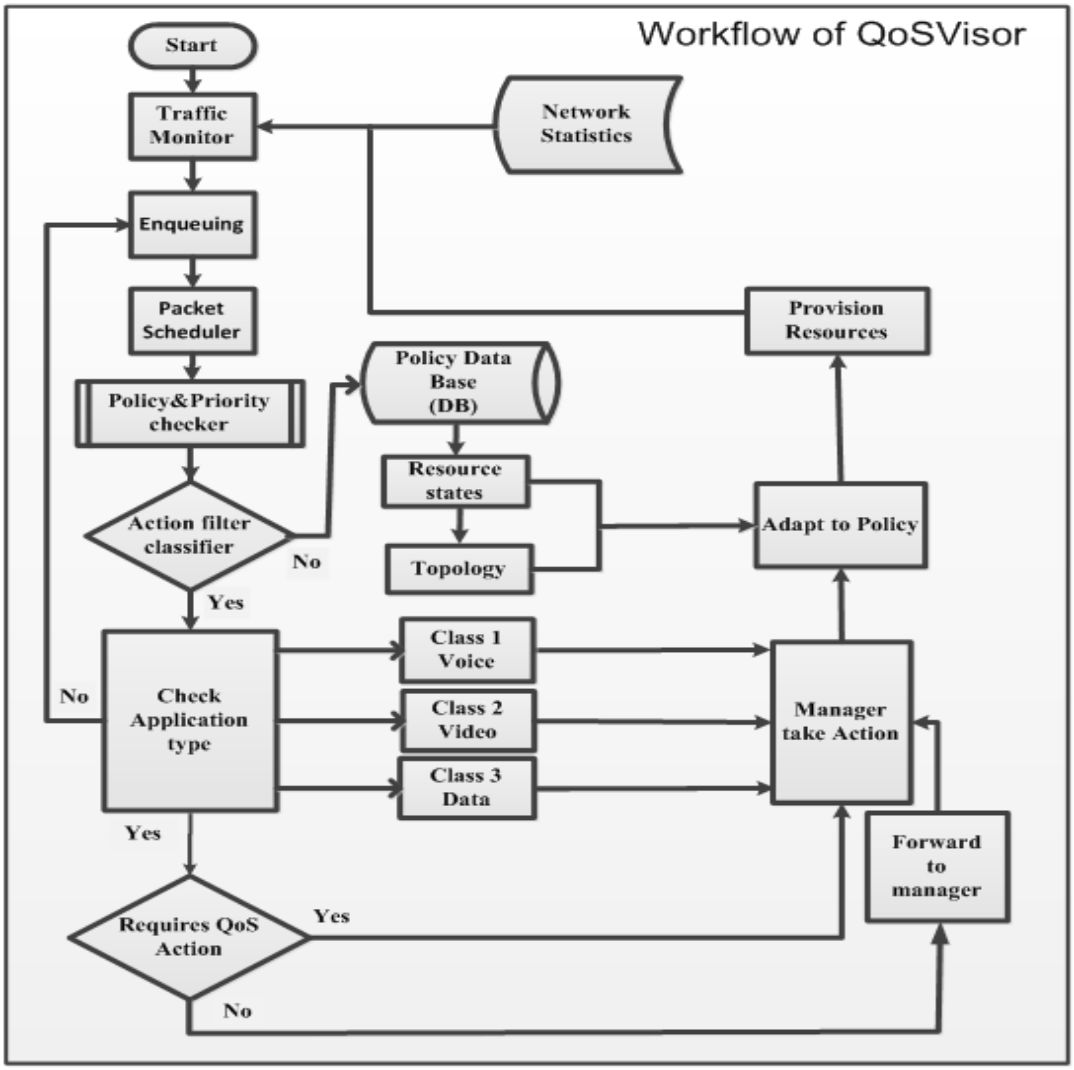


Figure (7-1) Operations underpinning the QoSVisor workflow

To address the limitations of WFQ a new improvement is proposed to enhance the current FlowVisor function. The objective of this algorithm is to add flexibility to the network operation and architecture, in a way that allows for definition and customisation of the required tasks for each active controller in the network flows. Also, add flexibility to the flow table (that includes flow entries containing match rules and actions that address active flows) to determine the actions allowed and not allowed by the controller. In case of no matching for these permissions, the algorithm can be used to modify those requests and keep only allowed actions in each request, or to return an error message to the controller (floodlight) promptly.

The proposed method adds an extension to existing processes within FlowVisor, as described in Section 7.3.1 (First Stage Packet Tagging Prioritisation Agent (PTP Agent) description) on page (121). The new process within the operation of FlowVisor is called the Packet Tagging Prioritisation Agent (PTP Agent) and it will verify the components of traffic shaping mainly WFQ internally according to the permissions database. Otherwise, any unpermitted actions will be removed or an error message will be sent to the controller as a command, which is then forwarded to the respective classifier element to the intended OpenFlow switch, via the FlowVisor flow space that interfaces with switches. The proposed design of QoSVisor is shown in Figure (7-2).

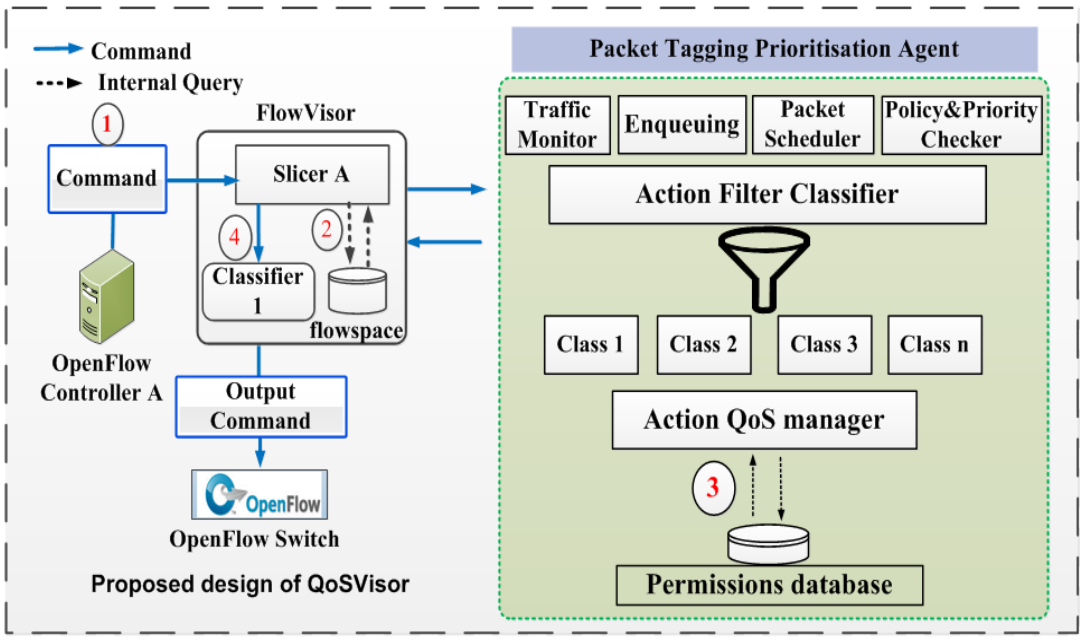


Figure (7-2) Proposed design of QoSVisor

7.3. The Operational Stages of the Proposed Architecture

The new architecture of QoSVisor comprises three operational stages, with each having its main elements and features. Throughout these stages, new algorithms are developed and implemented. These are the Packet Tagging and Forwarding algorithm (Algorihm 7-3), which is expresses the logic of packet identification, Algorithm (7-4), Tagging and Queueing, which performs the tagging and queueing, while keeping the scheduling algorithm separate from Algorithm 7-4. Lastly, there is Algorithm (7-5), the Packet Scheduling algorithm, which is proposed to enforce resource allocation for each queue, while keeping the actions for managing the entire switching and queueing mechanism separate. The objective of the architecture is to extend the current internal operation of FlowVisor with an enhanced QoS model. This will mean guaranteed and more precise QoS for different applications continuing to run, even during peak congestion times, and with priorities able to be agreed for specific requirements. Figure (7-3) illustrates the overall operation of QoSVisor.

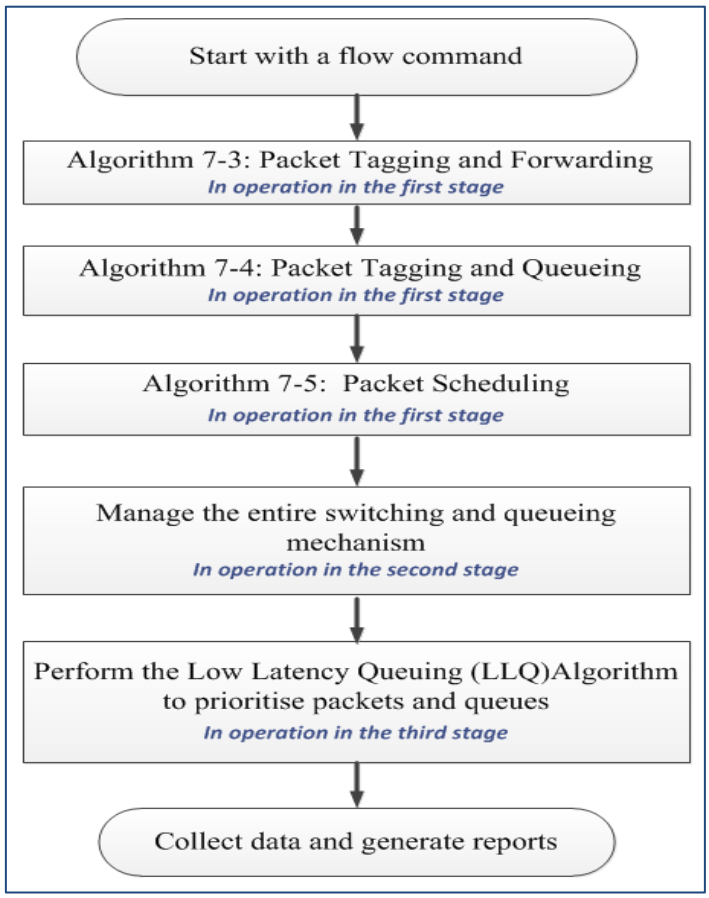


Figure (7-3) Overall operation of QoSVisor

7.3.1. First stage: Packet Tagging Prioritisation Agent (PTP Agent) description

The Packet Tagging Prioritisation Agent (PTP Agent) (Al-Haddad, Sanchez & Winckles, 2017) is an extension of the current internal operation of FlowVisor, designed to get the benefit of the isolation of any data plane/control plane communication channel from the production network that FlowVisor offers as well as the decoupling concept inherent to SDN. The Action QoS Slicer was previously proposed in Al-Haddad, Sanchez & Winckles (2017) and has now been change to PTP Agent, which represent a better description of its function. The proposed framework adds the following four extensions to the current internal operation of FlowVisor: traffic monitoring, enqueueing, packet scheduler, and a policy & priority checker.

A- Traffic Monitoring

This module is essentially a gate to monitor intervals of the network traffic, which collects all the primary data on traffic characteristics. It also analyses, reviews and manages network traffic for any type of failure, such as hardware and software failure, database failure, bandwidth congestion or network switch configuration failure (Benzekki et al., 2016) that can affect network performance, availability and/or security from specific switches. This is done using port and flow policies that are installed as software in specific switches in the data base; working as active policies these examine the computer network-based communication as well as data and packet traffic. The network traffic monitoring makes practical and effective use of the D-ITG tool embedded in the traffic monitoring function, and this also collects the metrics of network performance that are used to specify and evaluate QoS.

B- Packet Tagging and Forwarding

The forwarding mechanism uses preliminary QoS measurement to help sort various network applications, especially the latency-sensitive traffic type. The Packet Tagging and Forwarding algorithm sorts the various packets received from the hosts, with the mechanism

within this algorithm being focused on defining the priorities for different traffic classes, as explained in Table (7-1).

Table (7-1) The DSCP polices assigned to the selected ports

Protocol type	**IP packet type**	**Selected Application port number**	**Applied rule**	**DSCP**	**Priority**	**Defined class**
TCP	IPv6	9999	DSCP	46 (EF)	1	46
TCP	IPv6	8888	DSCP	18 (AF)	2	21
TCP	IPv6	1111	DSCP	00 (BE)	3	00

The packets will be tagged and forwarded in a list giving each type a packet weight with the allocated bandwidth, with this specific forwarded information then being stored in the flow table and subsequently, in the data base. This algorithm is the first major step towards packet enqueueing. Algorithm (7-3) expresses the logic of packet identification (Al-Haddad et al., 2021).

Algorithm 7-3: Packet Tagging and Forwarding

Input:

P: packets received from the hosts

Bandwidth: = maximum

Output:

SP: sorted list of packets

while $P \neq \emptyset$

for each packet pi in P **do**

SP→weight:= 0

SP→port:= 0

I_p←getPacketInfo(pi)

SP→ID = I_p→ID

if I_p→type == video **then**

SP→weight = 46

SP→port = 9999

SP→tag = video

SP→device = S3

SP→length = I_p→length

else if I_p→type == audio **then**

SP→weight = 18

SP→port = 8888

SP→tag = audio

SP→device = S5

SP→length = I_p→length

else

SP→weight = 0

SP→port = 1111

SP→tag = data

SP→device = S2

SP→length = I_p→length

end if

store_in_DB(SP)

forward_packet(SP, Bandwidth)

end for

end while

C- Enqueueing

This component is responsible for creating queues. The enqueueing action is an optional action in OF 1.2 (Lara et al., 2013) that forwards a packet through a queue attached to a port to provide certain QoS for each slice-based class of traffic type. It maps the priority levels that are necessary to operationalise the statements in the OpenFlow protocol, and to modify the state of the flow table; any flow entry can specify a meter in its instruction set. In this stage, each packet in the flow table meter entry contains a header field, with particular actions

being taken and counters, accordingly, are updated for matching flow packets, which are processed by a meter. Here, a meter table consists of meter entries, defining per-flow meters. Per-flow meters enable OpenFlow to implement various simple QoS operations, such as rate-limiting, which can be combined with per-port queues to implement complex QoS frameworks (Fernandez and Munoz, 2015), such as the DiffServ protocol. Algorithm (7-4) below implements the enqueueing of the incoming packets. Packets belonging to a single flow within the custom topology are enqueued in a sorted list in a particular queue of each destination port (Dst port) to apply the queuing policy. The queues are configured through the standard protocol Command Line Interface (CLI) of Mininet, the testbed emulator. Floodlight controllers can query the configuration and statistics parameters from the pre-defined queues. These switches rewrite the IP ToS/DSCP field in the IP header used to drive parts of the DiffServ protocol and to ensure the incoming packet forwarding behaviour, whilst also accommodating the Behaviour Aggregate (BA) classification by the core switch. The following component mechanisms map flows to the queues of the structure of the kernels (between controllers and switches) to define the data structures before the next component, which is the policy & priority checker.

Tagging and queueing while keeping the scheduling algorithm separate from Algorithm (7-4).

Algorithm 7-4: Packet Tagging and Queueing

Input:

P: packets received from the hosts

Bandwidth: = maximum

Output:

SP: sorted list of packets

EF_q: list of EF packets

AF_q: list of AF packets

BE_q: list of BE packets

while $P \neq \emptyset$

for each packet pi in P **do**

```
SP→weight:= 0
SP→port:= 0
Ip←getPacketInfo(pi)
SP→ID = Ip→ID
if  Ip→type == video then
        SP→weight = 46
        SP→port = 9999
        SP→tag = video
        SP→device = S3
        SP→length = Ip→length
        add_to_queue (pi, EF_q)
        sort_queue (EF_q, DSC)

else if Ip→type == audio then
        SP→weight = 18
        SP→port = 8888
        SP→tag = audio
        SP→device = S5
        SP→length = Ip→length
        add_to_queue (pi, AF_q)
        sort_queue (AF_q, DSC)

else
        SP→weight = 0
        SP→port = 1111
        SP→tag = data
        SP→device = S2
        SP→length = Ip→length
        add_to_queue (pi, BE_q)
        sort_queue (BE_q, DSC)

end if
        store_in_DB(SP)
        forward_packet(SP, Bandwidth)
```

end for

end while

Below, the packet scheduling to enforce the resource allocation for each queue, while managing the entire switching and queueing mechanism separate from the policy checker (in part *E below)* to perform dynamic maintenance of preliminary QoS is described.

D- Packet Schedulers

These components are responsible for enforcing resource allocation to individual flows. When queues start to build up in the switches the packet scheduler will decide which packets should get access to the resources. This is done according to three priority levels defined based on low latency queuing (LLQ) and the latest congestion management technique, as implemented by Algorithm (6-5).

Packet scheduling to enforce resource allocation for each queue, while keeping the actions for managing the entire switching and queueing mechanism separate

---

Algorithm 7-5: Packet Scheduling

Input:

EF_q: list of EF packets

AF_q: list of AF packets

BE_q: list of BE packets

Priority_list: [EF_q, AF_q, BE_q, null]

Output:

Log: list of status messages

While priority p in Priority_list ≠ null **do**

Packets = de_queue (p)

packet_type = p→type

for each packet pi in Packets **do**

```
            status = send_packet (pi)
            if status == success then
                Log→write (success_message, status, packet_type)
            else
                Log→write (failure_message, status, packet_type)
            end if
        end for
end while
return Log
```

E- Policy checker

The Policy checker deals with the traffic monitoring and the policy database (DB) for each queue, collects data and identifies any policy violations.

7.3.2. Second stage: the action filter classifier

In the switch (data path), the classifier divides an incoming packet stream into multiple queues, according to predefined rules. In QoSVisor, Behaviour Aggregate (BA), the Differentiated Service classifier, is used, as with the Traffic shaping algorithm (see Subsection 6.2.2: Experimental system and WFQ components), because it is simple. This will select packets based solely on the DSCP values in the packet header, in order to check the class of traffic type.

The classifier reads the packet headers and divides the traffic into its three classes: Class 1 for video; Class 2 for voice; and Class 3 for data. Each class will then have the suggested action needed for enhancing the QoS forwarded to the Action Quality Manager (AQM). This module checks the suggested action individually against the designed-in policies and then, forwards messages to the data base (DB) for user access, according to the control rules in the permissions data base.

The network traffic in this Packet Tagging Prioritisation Agent (PTP Agent) is handled based on the class of traffic that is defined by the DiffServ protocol. Each class is sent to FlowVisor's flow space as slices (video, audio and data), so as to be ready to be prioritised and sent as an output command for the OpenFlow Switch, as in Figure (7-2).

7.3.3. Third stage: action QoS manager

This part relates to the management of the entire switching and queueing mechanism, which is able to perform dynamic QoS enhancement based on the following five steps. The steps represent the logical sequence, which is run through repeatedly at 5 second intervals in order to be dynamic; recording the length of the SP (sorted list of packets) to be => 100 packets, as defined in Algorithm (7-4) above. This enables the current preliminary estimate of QoS status being achieved to be integrated into the network through REST API. Also, it allows for the installation of the data specifying the policy path for this module, by configuring minimum/ maximum rate limits and the DiffServ Type of Service (ToS) protocol. These steps are as follows.

1. Performing DSCP marking (traffic classification) and the IP header's ToS value bits within a TCP header as an input parameter to configure network parameters as well as mapping traffic to existing traffic queues created on the basis of Algorithm (7-4) (i.e. Packet Tagging and Forwarding).

2. Creating an overall QoS policy by accepting the source and destination Internet Protocol version 4 (IPv6) and Media Access Control (MAC) addresses.

3. Configuring the control module services to use software components using JavaScript Object Notation (JSON) based on RESTful API and the Python modelling language.

4. Scheduling the queue into different traffic classes and priority, as per Algorithm (7-5) (Packet Scheduling).

5. Collecting data and generating reports on the network traffic via the graphical user interface of the monitoring tools.

7.4. Congestion Management Technique of Low Latency Queuing (LLQ)

Low Latency Queuing (LLQ) (Balogh and Medvecký, 2010) is the latest priority queuing scheduling algorithm, usable as a module in the QoSVisor framework. It is aimed at supporting dedication of bandwidth, improving loss characteristics, avoiding and managing network congestion, shaping network traffic and setting traffic priorities across the network (Brundavathy et al., 2014). This is intended to enhance how arriving packets in a network flow table are picked out for scheduling for onward transmission, with less waiting time. This has a major impact on improving the QoS achieved, especially for video traffic.

From the functioning of Algorithm (7-5), the SDN controller is able to regulate traffic by allocating traffic type to expedited forwarding (EF), assured forwarding (AF), and best effort (BE) levels of priority, by modifying the DSCP bits of the packet (refer to Table 7-1) and storing it in Open vSwitch within the network system. Table (7-2) summarises the DSCP polices to the selected port, where each output port in a switch has its own algorithm to prioritise the packet forwarding, based on this information. This will dynamically allow for lower priority (e.g. BE) traffic to achieve maximum speed, when there is no higher priority (e.g. EF, AF) data flow. Table (6-2) shows the weighted fair queueing and prioritisation scheme for the LLQ algorithm.

Table (7-2) The WFQ simulation parameters for the LLQ algorithm

Maximum Rate	**No. of Queues**	**Maximum Rate per Queue**	**Port configurations**	**Flow type**	**Strict Priority Queues**
40 Mbps	Q1, Q2, Q3 With DSCP marking	Q1= 40 Mbps Q2= 40 Mbps Q3= 40 Mbps	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data	Q1=video=priority**1** Q2=audio=priority**2** Q3=data= priority**3**
70 Mbps	Q1, Q2, Q3 With DSCP marking	Q1= 70 Mbps Q2= 70 Mbps Q3= 70 Mbps	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data	Q1=video=priority**1** Q2=audio=priority**2** Q3=data=priority**3**
100 Mbps	Q1, Q2, Q3 With DSCP marking	Q1= 100 Mbps Q2= 100 Mbps Q3= 100 Mbps	port s4-eth4 port s4-eth5 port s4-eth6	Q1=video Q2=audio Q3=data	Q1=video=priority**1** Q2=audio=priority**2** Q3=data=priority**3**

WFQ, as presented in Formula (7.1), is implemented in the Open vSwitch, based on the strict priority used in Algorithm (6-5) within the system, by allowing delay-sensitive data to

be prioritised first (de_queue) and expressing a low-latency preference for a class of traffic type. The policies to perform traffic classification and queues, as defined in the SDN controllers, are based on the available criteria shown in Figure (7-4), which represent the control bits in the packet.

Weighted fair queuing (WFQ) extracted by SN (Sequence Number) in Formula (7.1) and (7.2):

SN = Previous_SN + (Weight * New_Packet_Length) (7.1) (Bahaweres et al., 2015)

Then,

Weight = 32384 / (IP Precedence +1) (7.2) (Bahaweres et al., 2015)

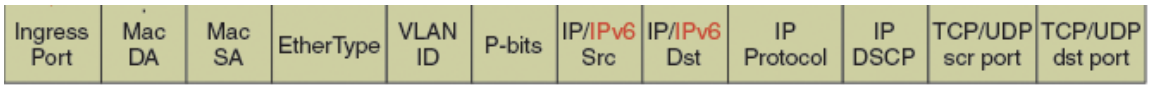


Figure (7-4) The control bits in the packet

The “qos” term has been created as a valid logical ID for implementing QoS for the queues within Linux-htb, which are standard in most Linux distributions and have been used in all the algorithms in this research. The different priorities within the queue commands are configured as follows.

- Best effort queueing:
set port s2-eth1 qos=@qos and set port s2-eth2 qos=@qos configured as best effort for created queue with dscp value=0 000 000 , with the Equivalent IP precedence value (000 – Routine) given to other data types, except video and audio.

- Expedited forwarding queueing:
set port s3-eth1 qos=@qos and set port s3-eth2 qos=@qos configured as expedited forwarding for the created queue with dscp value=46 (101 110), with the Equivalent IP precedence value (101-Critical).

- Assured forwarding queuing:

  set port s5-eth1 qos=@qos and set port s5-eth2 qos=@qos configured as assured forwarding for the created queue with dscp value=18 (010 010), with the Equivalent IP precedence value (010-Immediate).

The information in Table (7-3) represents the custom topology used within the working network system.

Table (7-3) Information of network topology and routing protocol

Number of nodes (switches)	5
Packet size	512 bytes
Routing protocol	RTP
Packets interval	1, 5, 15 minutes

Figure (7-5) represents the topology design for the QoSVisor framework, which can be compared with Figure (6-2) in Chapter 6. Each system is developed in order to govern the network traffic as well as to measure this and other specific network performance parameters. Each design uses the same controllers, switch numbers, port configurations and traffic classes, with each having the same number of queues. The difference between Figures (6-2) and (7-5), is that the queues in the former are configured without any priority, thus giving limited traffic shaping, while in Figure (7-5), the traffic is mapped to three queue priority levels (QPLs) based class of traffic type to implement and execute the LLQ algorithms (Algorithm (7-4): Packet Tagging and Forwarding and Algorithm (7-5): Packet Scheduling).

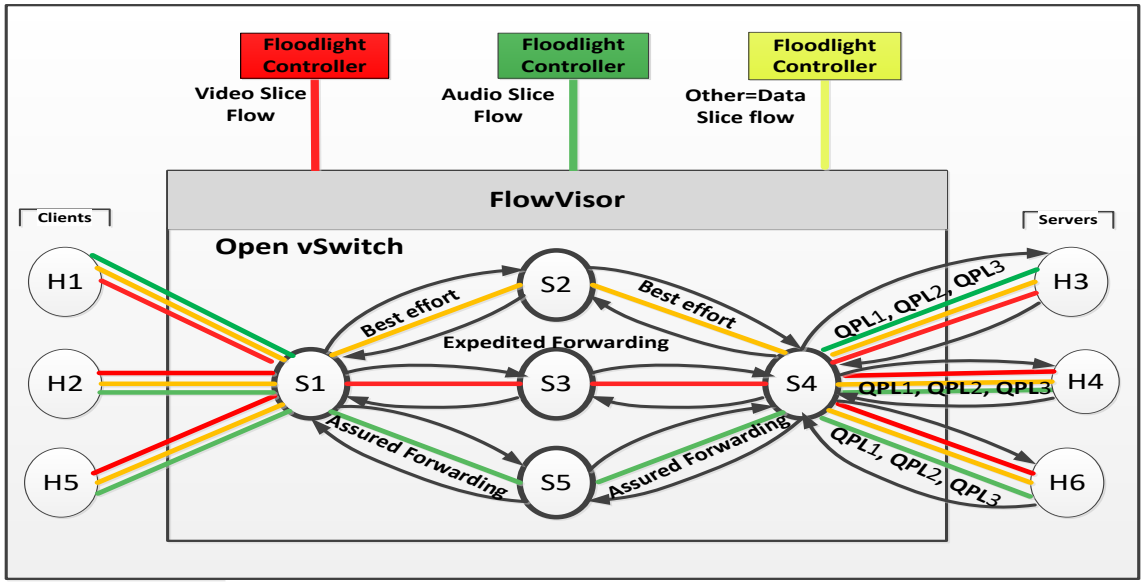


Figure (7-5) QoSVisor topology

In the QoSVisor topology, the S4 switch is configured to perform the LLQ priority algorithm for the three queues listed above for three different max-rates: 40, 70 and 100 Mbps. The way in which traffic is mapped to three queue priority levels (QPLs) is shown in Figure (7-5) and done based on the class of traffic EF, AF, and BE, as shown in Figure (7-6), EFQPL1, AFQPL2, and BEQPL3.

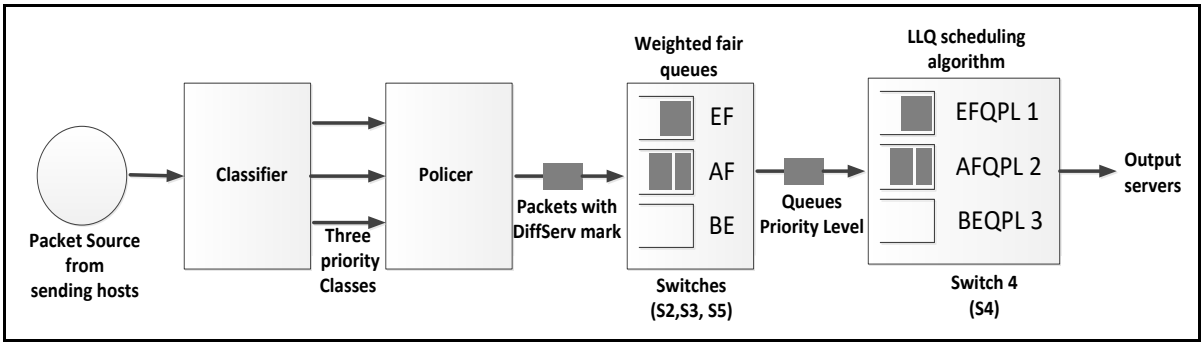


Figure (7-6) Algorithm for LLQ scheduling based on the QoSVisor framework

The code below shows the function to allocate priority for queues.

The function to allocate priority for queues

```
sudo ovs-vsctl -- set port s4-eth4 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3
```

The same is done for setting port s4-eth5 qos=@qos and port s4-eth6 qos=@qos.

The script configures the switches port on the custom topology for a maximum rating of 40 Mbps, as presented in Appendix 2. This is followed, likewise, for the 70Mbps, and 100 Mbps configurations (refer to Appendix 3 and Appendix 4).

7.5. Summary

In this chapter, a new methodology of QoSVisor and a Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for video, audio and data over TCP SDN-Slicing networks has been developed and tested. This was developed to overcome the limitations that exist with the traditional FIFO queueing method. The algorithm is aimed at reducing the delay and jitter for the packet transmission process, when compared with the standard algorithm FIFO, whilst at the same time maintaining or even improving the results obtained from the TS algorithm.

The implementation of this distinction was via a local weighting function, which sorts packets by the port and flow policy setup. This enables the use of more advanced queueing systems, for which more than one stage is necessary, along with extra memory capacity for handling long queues. Depending on each packet (flow, packet ID and weighted tag), the eeighted tag is the weighting assigned to the queue based on the traffic type, which is derived

from the element in WFQ (refer to Section 6.2: System components and implementation overview, on page (96)) the more advanced queueing algorithms have developed based on the following proposed algorithms: Packet Tagging and Forwarding, Tagging and queueing, and Packet Scheduling algorithms. These have been presented throughout this chapter, to meet the QoS in order to filter pre-classified packet data, to prioritise those packets for delivery, and to achieve the best performance for applications using different queueing techniques in an SDN.

The objective of the proposed PTP Agent is to extend the current internal operation of FlowVisor with an enhanced QoS model. This will mean guaranteed and more precise QoS for different applications continuously running, even during peak congestion times, and with priorities able to be agreed for specific requirements. In general, the proposed design is based on gathering four main technologies together: traffic engineering (TE), SDN, network hypervisors and network slicing. A new modification model is proposed for enhancing the current FlowVisor to meet the requirements of improving the QoS classification within a sliced-SDN. The proposed algorithm employs similar techniques to those used in the TS algorithm, with some changes and enhancements that take into consideration the limitations of network devices. The experimental results are presented with the corresponding statistical analyses in Chapter 8.

Next, in Chapter 8 a novel comparative approach for evaluating queueing systems is introduced. The chapter includes a detailed explanation of how the data were analysed. The SPSS statistical package Analysis of Variance (ANOVA) was used to perform the first level of analysis of the data, comprising various tests of hypotheses relating to the benefits to quality. Generally speaking, this involves testing for continuous or categorical relationships between conditions set up in the study design as independent variables and performance outcomes (dependent variables). A novel statistical strategy for results validation and reliability testing is proposed along with a detailed presentation of comparative results for the three systems that were implemented and tested previously in Chapters 5, 6, and 7.

8. Chapter Eight: A Novel Comparative Approach for Evaluating Queueing Systems

8.1. Introduction

This chapter includes a detailed explanation of how the data were analysed. The SPSS statistical package Analysis of Variance (ANOVA) was used to perform the first level of analysis of the data regarding various tests of hypotheses in relation to the benefits to quality. Generally speaking, this involves testing for continuous or categorical relationships between conditions set up in the study design as independent variables and performance outcomes (dependent variables), as shown in Figure (8-1).

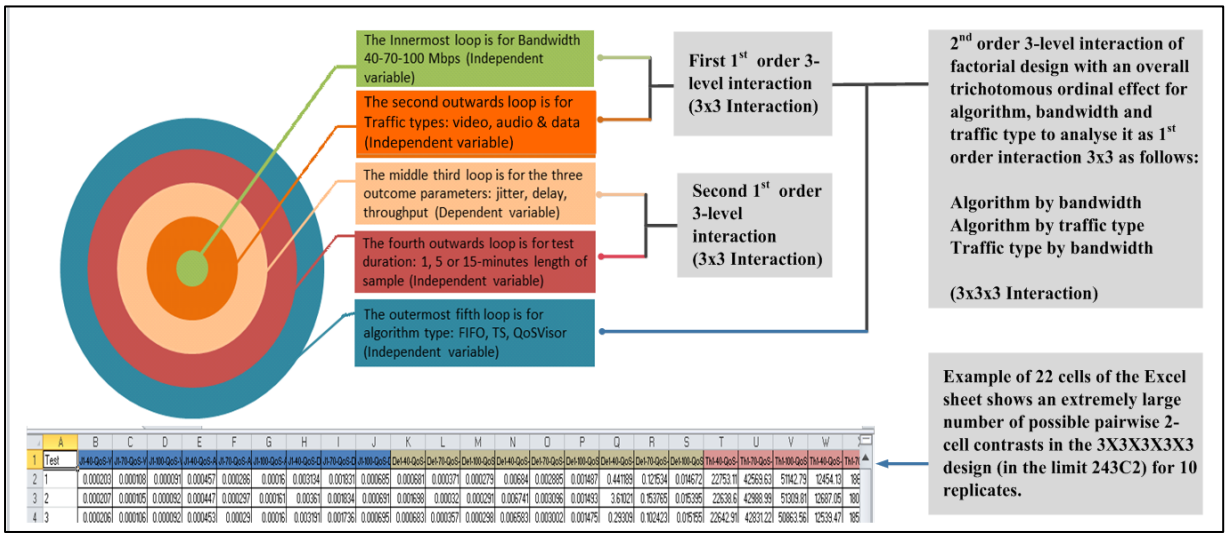


Figure (8-1) Data management for SPSS statistical package Analysis of Variance (ANOVA)

The analysis involved various ways of collapsing the means of 10 repeated measures in each of the cells in the 243-cell structure of data (3 traffic types) X (3 algorithms) X (3 durations) X (3 bandwidths) X (3 measures parameters). The advantage of three levels of categorical independent variable is that it provides at least a preliminary estimate of whether relationships may be linear or not. This chapter provides an overall performance comparison between the three algorithms FIFO, TS, and QoSVisor as well as some in-depth identification of the differences between them. The most general first analysis of such data structures for

the last 80 years has been the Analysis of Variance (ANOVA) or F-test (Rutherford, A., 2001), which can handle a 3-level categorical independent variable. Often, such analysis proceeds to comparisons of two conditions, using the Independent Samples t-Test. Rather than applying formal rules for conservative thresholds when there are three possible comparisons between the two levels of a categorical design variable, for this study, the researcher has adopted the convention of avoiding describing single results as categorically 'significant' or not.

But handle results near the conventional margin of significance as what they are – marginal, and requiring further analysis to resolve the dangling issue (i.e. the complexity of the data) has also taken an overview of the parallel contrasts in other parts of the data-structure and avoided interpreting isolated contrasts, which can be confusing in terms of what the dataset as a whole is revealing.

The foregoing statements provide an instance of statistical strategy; for constraining tests of the trustworthiness of results when many such tests could be done. In this chapter, the 3 X 3 X 3 X 3 X 3 design provides a non-arbitrarily structured scheme of conditions for evaluating the a *priori* expectations inherent to the design of the algorithms. This provides a framework for showing descriptive statistics for the dependent variables (e.g. means and the standard deviations). Secondly, to dissociate from statistical significance (which depends mainly on the size of the sample or the number of replicate measurements) – making the number of degrees of freedom (df) in the residual standard error (SE) (measures the quality of a linear regression fit) term in the F-ratio, the author also gives a partial eta squared (petasq for short) which is the best estimate from the available information on that is the size of the effect in question expressed as the variance within the total attributable to a specific effect in the design. These will be seen later in Tables (8-1) and (8-2).

Third, comparison tables are presented that illustrate the relationship between the variables (dependent and independent) for three measure types, namely throughput, delay and jitter, which have been extensively analysed and separately examined in detail for the three algorithms QoSVisor, TS and FIFO to acquire an overview of the totalled effects on performance.

8.2. Scheme of Conditions in Experimental Design for Evaluation

The evaluation plan for this study comprises a summary of the purposes of the evaluation and assessing the performance for the standard queueing model (FIFO) over various test conditions in comparison with other systems. Figure (8-2) shows the conceptual framework for the experimental design in conducting the evaluation for the three algorithms FIFO, Traffic shaping and QoSVisor within the sliced-SDN environment.

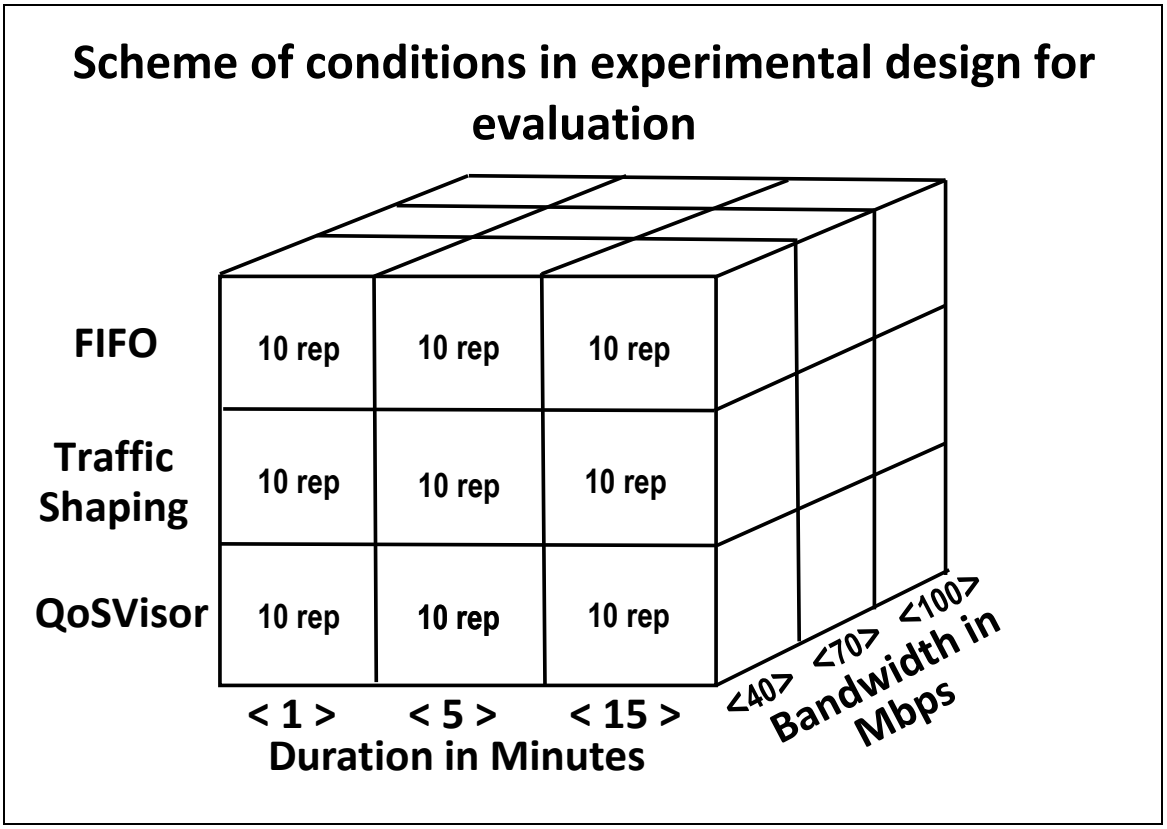


Figure (8-2) Schematic design cube for evaluation conditions

The special status of test run duration in the evaluation will be addressed later on in Section 8.3, where all the measurements for all the test period durations are reported. Interpretation of the effects of duration has to be made respecting two principles.

(a) The longer the period the more reliable the results and hence the more generalisable to application in practice.

(b) Because of the way the algorithms would work on packets of different length, the algorithms may predict differing patterns of results for different durations. Generally speaking, some similarity of pattern is to be expected, but the predictions can also make expected pattern differences quite specific. The underlying general point is that with a longer period the consequences of different queuing systems (the chief way in which the algorithms can differ) will be seen in different ways from how they are seen over a short period. Section 8.4 will explain the expectations from the algorithms as specific comparisons.

The long duration of 15 minutes is there to reliably determine genuine general and long-term effects and differences, (a) to see to what extent these are seen also at shorter duration – to be looking for a definite pattern – and (b) because 15 minutes may reveal some critical difference between the algorithms where the algorithms are not given room to fully use their queues and priority lists to good effect, especially at the short duration of 1 minute. Also (c) to avoid conclusions being confounded by samples of internet traffic of great length. In practice, it could be difficult with the results to distinguish between such a specific pattern and the generally greater reliability of the larger sample for spreading any (i.e. general) patterning.

8.3. Novel Statistical Strategy for Results Validation and Reliability Testing

The 243-cell (3 X 3 X 3 X 3 X 3) design, as summarised above, with 10 repeat measures in each cell, poses a number of analysis challenges. To meet these, the differing logical status of the design variables has to be understood. Clearly, the Ethernet switching algorithms is the chief (the researcher uses this term ‘chief’ to distinguish it from the ‘main’ effect in statistics which is a synonym for overall, i.e. not an interaction term) effect of interest. The algorithms have been developed as a useful solution to the problem of congestion, which might be implemented widely as constituting improvements to network performance. However, as the detailed rationale for the information processing proposals makes clear, a particular algorithm is unlikely to have more than minor direct and simple overall influence, but rather, is expected to show selective benefits. In other words, there are likely to be interdependences with the other design variables because those reflect particular adverse circumstances that the claimed improvement is intended to address. These interdependences are specifically the

predictable 3 X 3 patterns of the means reflecting the forms of interactions to be discussed in the predictions below. The first 3-level domains is the interactions occur with the other independent variables of (bandwidth and traffic type) (for example, some aspects of the algorithms are specifically for protecting the bandwidth-hungry video). Interactions pose particular problems for design and analysis, because (a) they demand a lot of data for certainty about them (power), and (b) there can be so many of them (combinatorial explosion) has to attend interpretation of any particular one apparently significant (further details are provided later).

The two remaining 3-level domains (parameters and durations) do not need essentially to be trichotomies, nor are they manipulated design variables like the first three, and not independent (in the sense of being separately controllable) interventions to affect an otherwise fixed process. The first of these final two pertain to the particular measures (throughput, delay and jitter) that are separate performance indices all contributing to QoS. They need to be integrated in some formal or informal way to provide an objectively rooted but overall and subjectively confirmable metric of QoS. In essence, these three levels of measures lead to three separate analyses, for which the predictions from the algorithm specifications are not necessarily similar. The same applies to predictions from particular patterns of interaction between the algorithm and bandwidth or traffic type.

Finally, the duration of test run does not have the same logical status as either of the former categories of design variable (wholly independent manipulation of conditions versus type of measure), but rather, stands somewhat ambiguously between them. In general, we can expect some degree of replication of patterns seen in the effects and interactions of the first three independent variables. So, when proceeding to take measurements for a further duration, which happens to be different, we should expect broadly to confirm patterns seen at the reference duration. For sampling reliability reasons, patterns would also be expected to be more distinct and hence, predictions from algorithm design more strongly confirmed at longer durations. However, there is also a paradoxical exception to this general statistical expectation. That is, at least for some classes of traffic type or some measured outcome variables, a pattern of internet traffic might critically test an algorithm at short duration, but do so less for a long duration, because the nature of the algorithm exploits the medium-term structure of the traffic. This duration variable seems to demand an instance of the repeated-measures class of analysis of variance, in which the consistency of other trends across all

durations could be attested, but in which also differences in them according to duration can be documented by an “interaction” term. in widespread use, this repeated-measures analysis of variance has a number of rarely acknowledged problems affecting the validity of estimates and p-values obtained, being vulnerable to the correctness of the assumption about the form of the variance/co-variance matrix. Hence, conclusions should not rely on that class of analysis alone. Also, the researcher has not tested the effects against their variability over the ten testing occasions as that error term reflects random variability in internet traffic, which is not of chief interest, reflecting chiefly variation in traffic type and packet length. Rather, the performance parameter is computed across the within-sample duration of each type of traffic, with that duration as denominator precisely to remove chance duration within the sample as a contribution to the error. Accordingly, in the factorial design applied to the means of the ten test periods in each condition, at any duration, the error term for the tests is the 2^{nd} order (3-way) interaction between algorithm, bandwidth and traffic type. Thus, for statistical significance of an interaction between, for example, algorithm and traffic type, the variance due to the table of means of the algorithm by traffic type possibilities has to be larger by a critical factor (the F-ratio) than that due to the dependence (i.e. 2^{nd} order interaction) of this upon the particular value of the third variable, here, generally bandwidth. So, significance in this example would correspond to its generalisability across bandwidth values.

The need for the present strategy section arises from the extremely large number of possible pairwise 2-cell contrasts in the 3X3X3X3X3 design (in the limit 243C2=29,403) or more plausibly 3C2= 3 per dimension of the four that are overall-effect independent variables (i.e. not jitter/delay/throughput as dependent variables, giving 243 such pairings in all). Hence, there is a very great danger of false-positive conclusions. For example, of the 243 pairwise single independent variables condition pairings, about 12 will be “significant” under chance alone, raising grave issues about which are made and which reported. There are various ways of tackling this danger, all of which involve some form of constraint limiting the number of statistical tests done, such as to allow the p-value (probability of getting such an extreme value under chance alone) to be taken literally. Notwithstanding the fallacies underpinning practice in past decades, taking the p-value literally is not a scientific aim in itself, but rather, has become *de facto* or the conventional scale for the degree of certainty to be attached to a finding in reporting empirical results. To handle these challenges, the following three-part strategy was adopted.

Part (1) The expectations from the comparisons of algorithms were reduced, to a limited number of specific predictions, to be found in Section (8.4) later. Mostly, these are for 1st order interactions (e.g. algorithm by traffic type or algorithm by bandwidth), but some are for further forms of prediction and those are discussed below in Subsections 8.4.1, 8.4.2, and 8.4.3.

Part (2) For the first inspection of the data an analysis model was adopted, in which all the 10 replicate measurements taken were first totaled and divided by 10, deliberately losing the degrees of freedom for replicate measurement but smoothing via aggregation. Analysis was then separated for measure type and duration, making 9 analysis, each with 27 cells.

The design adopted for these cells was factorial with an overall trichotomous ordinal effect for bandwidth against qualitative categorical traffic type and algorithm, thus permitting at least crude analysis as a function of quasi-continuous bandwidth; of linear versus non-linear influences from these other two variables. This permitted the data and the predictions to be first interfaced in a framework of the three 1^{st}-order interactions: algorithm by bandwidth, algorithm by traffic type, and traffic type by bandwidth. These six terms (six, because the entailed main effects of any significant interactions have also to be fitted) were to be retained, if significant, at the conventional criterion of $p \leq 0.05$, but otherwise, deleted one by one, in the order of interactions.

Then, main effects deleted first, within which the weakest would be deleted first, till only significant terms remained. This is the so-called backwards deletion approach (Sutter and Kalivas, 1993) (Mishra, C, 2019) and is the only practical way of filtering a large number of potential findings. It can be subject to false-positive errors, but mainly false-negative ones with variable data and small sample sizes. However, it is robust, provided that terms that have been only narrowly deleted are reconsidered in the context of the final post-deletions model. Deleted terms, by definition of the significance criterion for their deletion, do not improve the model when left in, so deleting them both simplifies presentation and reduces the degrees of freedom wasted. A standard format for extracting the summaries of the surviving models was prepared.

Clearly, this 3X3X3 factorial design, with replicates pooled, leaves 2^{nd} order interactions (i.e. 3-way interaction, henceforth called ‘triple’ for short) as the residual for use

as the denominator in significance testing, which makes the triple interaction unaccessible for significance testing**.** It should be noted that there are not many degrees of freedom in the triple interaction terms, which mostly has to be used as residual here,; therefore, although some very high percentages of variance explained (100 X Rsq) are found amid the results, compared with typical values in empirical social, biological, and medical research, the Rsq is also heavily penalised by (appropriate) adjustment for the few degrees-of-freedom (df) remaining in the error term.

An interaction (i.e. a dependency of one effect upon another) may provide a specific sharp finding for testing an explanation and the absence of an interaction is not necessarily a failure. Differences within the interaction table that are greater than 2 SE (standard error) and are subject to an a *priori* prediction are still worthy of comment. A simpler model of overall (i.e. 'main') effects only can be a simple and yet, still powerful summary, when there are no interactions found or expected. However, the null results for predicted effects are reporting concentrates on interactions and their components, even if there are obvious expected main effects, because it is in the nature of the algorithms' differing handling of constraints that interactions will be strong.

Part (3) A second general form of analysis was undertaken, now keeping the 10 replicate measurements separate. This had three different purposes and three levels, with corresponding classes of analysis, denoted (a-c). The first purpose was to document in mixed models, in hypothesis-testing mode, **(a)** the forms of triple interaction for which a prediction was made, and to report in exploratory mode on the remainder (i.e. not especially predicted, although not necessarily contra-predicted). Nine such triple interactions between algorithm and test conditions (bandwidth and traffic type) are of interest (i.e. one each for the three outcome measures by the three durations). The purpose of exploration can include hypothesis generation for future tests. but this was not a major aim here. Rather, it was to see where the triple interaction may be particularly large and so require cautious interpretation of the first order interactions already mentioned as generally carrying the chief predicted and obtained patterns of results. **(b)** From the same mixed models can come a more powerful test of 1st order interactions, using as residuals, not the triple interaction as above, but rather, the variance over measurements within cells of the 1st order interaction term of interest (i.e. testing the emergence of an effect now by its consistency over random resampling of the same condition, rather than over three way interaction patterning). Whilst addressing a

slightly different issue from that in the pooled-replicates analyses above, these tests can be complementary and useful in instances of statistically marginal 1st order interactions by the above method, where there is an expectation formulated beforehand. **(c)** Thirdly, within the context of a marginal or significant 1st order interaction, a contrast may be sought between two particular cells. These could differ between two levels of a design variable, where the p-value under the first method above might apply non-specifically to the variable as a whole or the p-value might apply between cells for which there was no overall effect documented, e.g. between corresponding cells in terms of some pairing within the 3 X 3 X 3 factorial design (i.e. between any of the nine analyses distinguished by measure and duration). This can be done with a t-test of the the unrelated form, as the first of 10 quasi-random replicates in one cell has no specific relationship with the first in another cell. This is broadly equivalent to an F-test with 1 df in the numerator.

Some scenario for adjusting multiple comparisons of this type e.g. Bonferroni appears to be highly needed (SPSS offers Bonferroni adjusted significance tests for pairwise comparisons involving the computation of a p-value for each pair of the compared groups) (Mishra et al., 2019) given the number of combinations possible. However, where there is not a small list of possibilities any of which would be taken as evidence underpinning the candidate conclusion, their precise application can become extremely complicated. Use carries its own uncertainty about the degree of caution that it really expresses, and the method can still be ‘gamed’ like the ordinary hypothesis tests for a single contrast. Check meaning Hence, it is simpler if the discipline that only a few such tests must be generated and run is accepted. Moreover, all that are run must be at least briefly reported, so as to avoid the possible reporting and publication biases that spread false-positive conclusions by piecemeal reporting of the more impressive ‘findings’. Where a multiple-level variable is used in exploratory fashion for novel findings, it is customary to incorporate a conservative correction for the number of tests done. Here, as all variables are expected to have some effect and there is a potentially very large number of tests, the constraint of a significant overall effect and the standard error (SE) from that term as a yardstick for what differences are worth comment are used.

8.4. Algorithm Predictions

8.4.1. QoSVisor algorithm predictions according to specific comparisons

A: Type of traffic combined with bandwidth

Process statement for QoSVisor

QoSVisor has a far more powerful method of processing data than FIFO and TS, owing to the LLQ priority scheme and the proposed extension to the Packet Tagging Prioritisation Agent (PTP Agent). This agent is utilised at each switch and the video flow policy allocates the highest (i.e maximum possible from what is available) bandwidth to the highest priority (priority=1). These allocations have to be envisaged within a total network of relays, where the algorithms operate at many stages along a packet path. The audio flow is given second highest priority (priority=2) and hence, it can use the maximum available bandwidth, if there are no packets from the priority=1 flow in the queue. Data traffic has the lowest priority policy (priority=0). The basic priority policy for any algorithm has been shown to have a real impact on the adequacy of flow statistics, using simulation techniques, so generally, informs priority policies for the three traffic type classes. Some traffic periods (represented here by test samples) with high video flow can consume all the available bandwidth (only in QoSVisor and not as a generality for any algorithm), and this arises where there are more video packets than audio and data ones. Based on the theory underlying the new proposed policy in QoSVisor, the expected outcomes for video combined with the bandwidth must show lowest delay compared to the other two algorithms, lower jitter, and the highest throughput in all bandwidth groups (40, 70, 100 Mbps). However, in general, there is no control over how many video packets might have been applied to the internet in a given sampling period and as the sampling of flow in the tests run reflects this lack of control, the inclusion of the long sampling period of 15 min is desirable.

B: Traffic type combined with test duration

Process statement for QoSVisor

The test duration is mostly a matter of general reliability and clarity of the results. Some specific patterns may be more evident during short durations than long ones, like the interaction between the traffic type and the algorithms, reflecting how each algorithm processes the traffic type. Some differences in what the algorithms can achieve might be expected when the duration is very short. The predicted pattern is for slightly decreasing delay and greatly decreasing jitter, with increasing throughput, as the test duration increases. The delay improvement may not be so great due to the memory used to implement the more advanced queueing system. These predictions are weakened when considering mainly data traffic, as it has the lowest priority in the QoSVisor algorithm. This statement does not hold for the other two algorithms.

C. Bandwidth by test duration

Process statement for QoSVisor

The duration of the test run allows for longer transmission; however, presence of maximum bandwidth should enable support of the QoSVisor algorithm. There is no essential correlation between bandwidth and test duration as they are independently manipulated in the evaluation design. For longer test durations, the QoSVisor algorithm should show increasing throughput, along with decreasing delay and jitter, given more bandwidth and longer test duration. Traffic type is a separate variable that can modify the overall predicted effects of bandwidth and test duration, negatively or positively, depending on the class of traffic type that predominates in the mix of traffic type that is put in to the network.

8.4.2. Traffic shaping (TS) algorithm predictions according to specific comparisons

A: Traffic type combined with bandwidth

Process statement for TS

According to the weighted fair queueing process within TS, a defined set of parameters describes the consequences of the queues for performance and these should be evident when evaluating the algorithm. Three queues are created for each bandwidth (i.e 40, 70, 100) as Q1: high-expedited forwarding (EF), allocated to video; Q2: medium-assured forwarding (AF), allocated to audio; and Q3: low-default forwarding (BE), being allocated to data, which are performed by the SDN controller used in the system). However, there is no general inbuilt priority for allocating bandwidth to a specific traffic type, but the WFQ scheme can give video the highest bandwidth allocation. For all traffic types, the expectation is for lower delay, lower jitter and higher throughput, for TS compared to FIFO. The audio flow can dynamically use the remaining bandwidth after accommodating the video flow as well as the allocated bandwidth to audio from the WFQ scheme, whilst the data flow is expected to show higher delay than the video and audio and also depends on the bandwidth limit set in the evaluation conditions.

B: Traffic type combined with test duration

Process statement for TS

In the TS algorithm, there will be similar effects for the traffic types at all test durations. This algorithm is anticipated to provide low delay and jitter, with higher average throughput for video, then audio and subsequently, to data, for all test durations 1, 5, 15 minutes, providing the WFQ sees enough bandwidth remaining to cater for all types of information flow. With longer durations, there should be a good pattern of performance, better than for short duration. However, some variability the in patterns of results may be seen due to the statistics of the flow type over the 10 replicates for audio and data.

C. Bandwidth by test duration

Process statement for TS

Sharing the remaining bandwidth enables the TS algorithm to provide good throughput even for the shortest duration, with lowest delay, lower jitter and good throughput for all flow types. For long durations, it might be a challenge to accommodate high availability of video over the other traffic types, this being because performance will depend on the chance of instantaneous demand.

8.4.3. FIFO algorithm predictions according to specific comparisons

A: Traffic type combined with bandwidth

Process statement for FIFO

The ordinal predictions for delay, jitter and throughput from the FIFO algorithm are limited, as the patterns of delay, jitter and throughput will be driven primarily by the partly random properties of the input. There is no priority or ordering scheme for traffic type to enter the flow table, and this will result in somewhat equal delay across all traffic types. All types of traffic will compete for the bandwidth. The average waiting times will be longer than for the other algorithms i.e TS and QoSVisor and throughput may not be much lower than for them, but the level of jitter is generally expected to be high.

B: Traffic type combined with test duration

Process statement for FIFO

Results will be very variable at short test duration with FIFO, because of the lack of prioritisation and, it is expected that the effects of congestion will be evident. The delay, jitter and throughput should increase for growing test duration, and should take on a similar pattern.

C. bandwidth by test duration

Process statement for FIFO

The effect of bandwidth with the test duration, is that in FIFO, whatever class of traffic type enters the pipeline, the first will be processed and what this entails will be variable from one test run to another so the results will be highly variable: the test-to-test variation will be great. Furthermore, at short duration, the variability of the performance parameters with traffic type will be big, because it depends on the single accident of the first traffic type in. With increased bandwidth and duration, this variability due to the chance of the initial traffic type will narrow, due to the reduced importance of the starting traffic type, and the long-term averages will come to reassert themselves with lesser variability. Mean throughput, delay, and jitter effects will differ between traffic types, particularly at short durations, but they will become less traffic type-dependent at higher durations and bandwidths.

8.5. Statistical Significances of Differences by ANOVA on Cell Averages

Subsection 8.5.1 below, shows tests of between-subjects effects results of the two-way ANOVA (Rutherford, 2001), showing that the statistical significance of the overall effects of independent variables and the interaction between two of these. The researcher screen for effects deserving any comment to those with $p < 0.05$ overall but limit detailed interpretation to those of appreciable magnitude with partial eta squared (petasq for short, as the Greek ήp2 is burdensome) above 0.020. There is no universal set of anchor standard values for 'small' or 'large' effect with partial eta squared, and the observed values of this index depend on the range or strength of contrast put into the independent variables in the study design and on the measurement error or reliability. However, if the set of values and their ranges have been chosen to be representative of the conditions of application of the result, then there is some traction in using the error (via total Rsq or % of variance accounted for, which is 100X that) for the total prediction from a set of variables in an analysis. Partial eta squared embraces both of the above sources of influence and unlike correlations or differences can be used on and compared between both continuous (e.g linear) and categorical effects (eg differences). Smaller values than 0.020 may be discussed scientifically and usefully explained, if they are statistically significant. But effects with partial eta squared 0.020 represent in a generalised

and abstract way effects that account for 2% of the variability; they may not be large but in engineering or business accounting terms for example, 2% of performance or commercial turnover can make the difference between success and failure. For a given sample size, the relation between this magnitude and statistical significance is fixed.

The figures in Sections 8.6, 8.7 and 8.8 below are averaged. This entails that, although the cumulation of all jitters, delays and throughputs will be greater for longer samples, the means in the given figures will be broadly comparable, because they have been divided by the denominator duration (i.e. the performance figures quoted amount to averages per unit time). Where the means are not similar in the way thus expected, EITHER this is random, due to the particular internet traffic that just happened to be around when the intervals were sampled, OR, and particular for 1 min, it must be due to systematic differences between the algorithms (in relation to typical packet length) in terms of how the length of their memory buffers and queuing priorities operate upon the traffic types received. In addition, the results are reported in batches of three models (one for each duration), because similar results will be expected (if perhaps more variable for the shorter duration) and this makes it easier to see, for example, how general a term like Traffic type * algorithm is interact.

A. Tests of Between-Subjects

'Tests of between-subjects' is the name conventionally given by SPSS for most ANOVA (Rutherford, 2001) applications, as most effects are, indeed, between the individual sampling units, usually individual people. It is a gross apportionment of variance to sources in the independent variables. This is the allocation of variance between the effects in the design in ANOVA (Rutherford, 2001), showing whether there are significant differences between the conditions as a whole, before drilling down to contrasts involving the individual levels of a design variable (confusingly called a 'factor').

In the analyses utilised, the researcher has accepted a limitation for the purpose of simplicity, because of the very large number of analyses: the 10 replicate measurements are collapsed in data-management prior to analysis for basic reliability, as independent documentation of the long- or short-term statistics of the internet traffic as an error term against which to test effects of interest is not available. Hence, the concern is not so much

about the replicability of a small difference over against traffic samples, but rather, the emergence of the overall ('main') effects and 1st order interactions between pairings of design variables emerging from an error term which is not within-condition replicates, but the 2nd order (3-way) interaction of the three variables in each design.

To conserve degrees of freedom for the error in each analysis, the researcher has fitted only the two of the three 1st-order interactions that are of most interest; as 'algorithm' is the aim of the software development its main-effect and particularly its interactions are usually the effects of chief interest for testing the reliability of differences created by the development.

Overall ('main') effects cannot be interpreted (as to p-value and partial eta-squared) in models containing interactions involving them, although they can be 'seen' in the adjusted means table, which is simpler to read, in the way that raw descriptive are, and here the standard error estimated from the model as a whole can be used as a yardstick of reliability. In the first model shown below, the format of one analysis is illustrated in some detail. The dependent variable is throughput. The high value of Rsq shows that it is generally a good model (over 80% of the variance is accounted for) and indeed, all terms within it are significant.

The two significant interactions are not subject to the need to show a main-effects-only (MEO) analysis and they reveal that a simple overall interpretation of traffic type, bandwidth would probably not be valid, certainly cannot be taken literally, because of these dependencies; rather, and further interpretation must rely on the adjusted means table (sometimes called the interaction table). The parameter estimates table is complex and hard to read, so after the example presented below, only the adjusted means table and standard error approach are used, with the full parameter estimate table archived as a backup as to what should be interpreted.

B. 3-way ANOVA of throughput as function of algorithm, bandwidth and traffic-type

The interaction terms all have * in their row titles. To illustrate the location of significant differences in the tables below, the differences that are significant from the reference values (0.0) have been italicised (green) or emboldened (yellow) So, it can be seen in Table (8-2) that Traffic type=1.00 is 0.000, Bandwidth=1.00*Traffic type=1.00 is 0.001, Bandwidth=2*Traffic type=1 is 0.005 and the Algorithm=1.00*Traffic type=1.00=0.000. In addition, the bandwidth parameter IDs are defined as follows in Mbps: Bandwidth=1.00 is the ID for Bandwidth 40, Bandwidth=2.00 is the ID for Bandwidth 70, and Bandwidth=3.00 is the ID for Bandwidth 100. Algorithm=1.00 is referred to FIFO, Algorithm=2.00 is referred to TS, and Algorithm =3.00 is referred to QoSVisor. While traffic type =1.00 referred to video, traffic type =2.00 referred to audio, and traffic type=3.00 referred to data.

Table (8-1) Dependent variable: Throughput traffic averaged across 10 tests

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	11510449739.364[a]	14	822174981.383	12.642	0.000	0.937
Intercept	12697674532.85	1	12697674532.8	195.246	0.000	0.942
Bandwidth	1432600329.269	2	716300164.635	11.014	0.002	0.647
Algorithm	6641831.682	2	3320915.841	0.051	0.008	0.950
Traffic type	5536612646.303	2	2768306323.15	42.567	0.000	0.876
Bandwidth*Traffic type	1694206148.622	4	423551537.156	6.513	0.005	0.685
Algorithm*Traffic type	2840388783.488	4	710097195.872	10.919	0.001	0.784
Error	780409304.713	12	65034108.726			
Total	24988533576.93	27				
Corrected Total	12290859044.07	26				

a. R Squared = 0.937 (Adjusted R Squared = 0.862)

C. Parameter estimates

Table (8-2) Dependent variable: Throughput traffic averaged across 10 tests

Parameter	B	Std. Error	t	Sig.	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound	Partial Eta Squared
Intercept	2912.823	6010.829	0.485	0.637	-10183.647	16009.293	0.019
[Bandwidth=1.00]	-4794.977	6584.533	-0.728	0.480	-19141.441	9551.488	0.042
[Bandwidth=2.00]	335.319	6584.533	0.051	0.960	-14011.145	14681.784	0.000
[Bandwidth=3.00]	0[a]	.	.	.	.	.	.
[Algorithm=1.00]	19106.238	6584.533	2.902	0.013	4759.773	33452.702	0.412
[Algorithm=2.00]	1818.843	6584.533	0.276	0.787	-12527.622	16165.308	0.006
[Algorithm=3.00]	0[a]	.	.	.	.	.	.
[Traffic type =1.00]	78587.338	8500.595	9.245	0.000	60066.132	97108.544	0.877
[Traffic type =2.00]	6366.194	8500.595	0.749	0.468	-12155.012	24887.401	0.045
[Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Bandwidth=1.00] * [Traffic type =1.00]	-38708.780	9311.936	-4.157	0.001	-58997.744	-18419.815	0.590
[Bandwidth=1.00] * [Traffic type =2.00]	-128.408	9311.936	-0.014	0.989	-20417.373	20160.556	0.000
[Bandwidth=1.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Bandwidth=2.00] * [Traffic type =1.00]	-31909.632	9311.936	-3.427	0.005	-52198.597	-11620.668	0.495
[Bandwidth=2.00] * [Traffic type =2.00]	-652.596	9311.936	-0.070	0.945	-20941.560	19636.369	0.000
[Bandwidth=2.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Bandwidth=3.00] * [Traffic type =1.00]	0[a]	.	.	.	.	.	.
[Bandwidth=3.00] * [Traffic type =2.00]	0[a]	.	.	.	.	.	.
[Bandwidth=3.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Algorithm=1.00] * [Traffic type =1.00]	-53733.234	9311.936	-5.770	0.000	-74022.199	-33444.269	0.735
[Algorithm=1.00] * [Traffic type =2.00]	-5940.373	9311.936	-0.638	0.536	-26229.337	14348.592	0.033

[Algorithm=1.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Algorithm=2.00] * [Traffic type =1.00]	-11917.109	9311.936	-1.280	0.225	-32206.074	8371.856	0.120
[Algorithm=2.00] * [Traffic type =2.00]	7692.200	9311.936	0.826	0.425	-12596.765	27981.165	0.054
[Algorithm=2.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.
[Algorithm=3.00] * [Traffic type =1.00]	0[a]	.	.	.	.	.	.
[Algorithm=3.00] * [Traffic type =2.00]	0[a]	.	.	.	.	.	.
[Algorithm=3.00] * [Traffic type =3.00]	0[a]	.	.	.	.	.	.

This parameter is set to zero because it is redundant.

The 0[a] coding here in parameter estimates is closer to how the estimation algorithm works, but this table contains the p-value and partial eta-squared and confidence intervals. This table is included to show the necessary stages, whilst the later results are given by the adjusted means table with those statistics summarily stated in the title, as set out in the text at the beginning, but the Standard Error (SE) is there available as general guidance on what deserves comment or explanation i.e. +/- 2 SE.

8.6. Throughput results analysis

In this section, throughput results are broken down for durations of 15, 5, and 1 minute(s). More satisfactory in this instance than the parameter estimates table are the graspable adjusted means tables for the respective two interactions retained. The first pools across the algorithm and shows the clearly increasing capacity with high bandwidth in transmitting video as well as the overall greater throughput of video. For 15 minutes duration , with the results largely independent of the algorithm, Bandwidth * Traffic type, and Traffic type * Algorithm are significant. With a duration of 5 minutes, the results are also largely independent of bandwidth, Traffic type * Algorithm was significant. For 1 minute duration, largely independent of bandwidth, Traffic type * Algorithm is, again, significant.

8.6.1. Analysis of test duration 15 minutes

(Model 2 in deletion sequence Adj Rsq 0.862, results being largely independent of the algorithm.)

Bandwidth * Traffic Type Interaction for throughput [p-value =0.005 petasq=0.685]

Table (8-3) Results for throughput in megabits per second (Mbps) measures expressing Bandwidth * Traffic Type interaction for 15 minutes duration

Dependent Variable: Throughput traffic averaged across 10 tests

		Mean		95% Confidence Interval	
Bandwidth	Traffic Type	Throughput	Std. Error	Lower Bound	Upper Bound
1.00 40	1.00 Video	23087.984	4655.968	12943.501	33232.466
	2.00 Audio	11914.602	4655.968	1770.119	22059.084
	3.00 Data	5092.873	4655.968	-5051.609	15237.356
2.00 70	1.00 Video	35017.427	4655.968	24872.944	45161.909
	2.00 Audio	16520.710	4655.968	6376.228	26665.193
	3.00 Data	10223.169	4655.968	78.687	20367.651
3.00 100	1.00 Video	66591.740	4655.968	56447.258	76736.223
	2.00 Audio	16837.987	4655.968	6693.504	26982.469
	3.00 Data	9887.850	4655.968	-256.633	20032.332

In Table 8-3, it can be seen that the relative throughput of audio and video is pretty similar, with relatively slight influence of bandwidth on this traffic class (traffic type), the effect being about 2- fold at both 40 and 70 Mbps, but about 4-fold at 100 Mbps. This pattern is discussed because the Bandwidth * Traffic type interaction is significant (P= 0.005 in the Table) for the between-subjects effects (Al-Haddad et al., 2021). The throughput for each algorithm is slightly higher for the highest two bandwidths, as generally expected. However, for all the algorithms across all bandwidths the relation of audio to data is about the same, which presumably because the presented bit rate is higher for audio than it is for data.

Most of the distinctiveness in this same interaction table for Traffic type * Bandwidth is accounted for by the fact that video, with its higher presented bit rate, has a throughput measure that is critically dependent on the available bandwidth. It climbs more steeply than for audio and data overall, but even more steeply, by over 6X the Standard Error (SE) for the BW increase from 70 to 100 than it does for the increase from 40 to 70 (here by only just over 2X the SE). This is presumably because in the algorithms where the scheduling is done

by assigning a bit weight to each flow and to the (PTP Agent), the video flow is given more weight than the other two flows.

These findings are unsurprising and represent the basic realities confronting all the algorithms. The chief combination of variables for network design evaluation (algorithm in relation to traffic type) is not involved in this first analysis, which has been done with the reliability offered by a long duration sample. It serves, to some extent, to calibrate the measurement paradigm with the subroutines for measurement implemented in the platform, by giving p-value=0.005 (highly certain) and petasq=0.685 (very large effect).

These results provide further support for this research hypothesis in Section 1.3 and the predictions in Section 8.4 and Subsection 8.4.1 (A: traffic type combined with bandwidth), where the expectations from the algorithm QoSVisor specifically matched the results. QoSVisor has a far more powerful method of processing data than FIFO and TS, owing to the LLQ priority scheme and the proposed extension to the Packet Tagging Prioritisation Agent (PTP Agent). This agent is utilised at each switch and the video flow policy allocates the highest (i.e. maximum possible from what is available) bandwidth to the highest priority (priority=1). These allocations have to be envisaged within a relay total network, where the algorithms operate at many stages along a packet path. The audio flow is given second highest priority (priority=2) and hence, it can use the maximum available bandwidth, if there are no packets from the priority=1 flow in the queue, whilst data traffic types have the lowest priority policy (priority=0). This basic priority policy for any algorithm has been shown to have a real impact on the adequacy of flow statistics, using simulation techniques, so generally informs priority policies for the three traffic type classes. The predictions form was based on the theory of the new proposed policy in QoSVisor, the expected outcomes for video combined with the bandwidth must show lowest delay compared to the other two algorithms, lower jitter, and highest throughput in all bandwidth groups (40, 70,100 Mbps). TS algorithm and QoSVisor were slightly similar, but this analysis for QoSVisor has only shown lower throughput for data traffic type based on the PTP Agent. In contrast, in the second of the two important 1st-order interaction, as shown in Table (8-4), it can be seen that it is somewhat stronger and distinctive in form (partial eta-squared 0.784) and (p-value=0.001), involving both algorithm and traffic type.

Traffic type * Algorithm for throughput [p-value =0.001 petasq=0.784]

Table (8-4) Results for throughput in megabits per second (Mbps) measures expressing Traffic type * Algorithm interaction for 15 minutes duration

Dependent Variable: Throughput traffic averaged across 10 tests

Traffic type	Algorithm	Mean throughput	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 Video	1.00 FIFO	21847.142	4655.968	11702.659	31991.624
	2.00 Traffic Shaping	46375.872	4655.968	36231.389	56520.354
	3.00 QoSVisor	56474.138	4655.968	46329.655	66618.620
2.00 Audio	1.00 FIFO	20697.995	4655.968	10553.513	30842.477
	2.00 Traffic Shaping	17043.173	4655.968	6898.691	27187.656
	3.00 QoSVisor	7532.130	4655.968	-2612.352	17676.613
3.00 Data	1.00 FIFO	20532.508	4655.968	10388.026	30676.990
	2.00 Traffic Shaping	3245.114	4655.968	-6899.369	13389.596
	3.00 QoSVisor	1426.270	4655.968	-8718.212	11570.753

Looking more closely at Table (8-4), it can be observed that the throughput of audio, data and video is pretty similar in FIFO. Throughput for TS and QoSVisor is similarly high for the two most demanding and thus, prioritised traffic types, as generally expected. Moving to lower-priority audio and then data, it can be seen that TS lets through just over twice the amount of information compared to QoVisor. Comparing audio with data, there is, again an approximate five-fold advantage for audio related to data. This is a direct consequence of the algorithms as programmed, with the bit rate weight policy being higher for audio than for data in both TS and the proposed prioritisation agent of QoSVisor. As to reliability (significance), this is expressed in difference and not ratio terms, so some of these apparent differences would not be considered significant, at least in this type of model, but the issue can be taken up with additional degrees-of-freedom when distinguishing the 10 repeat measures. Only differences of around 10,000 exceed 2X the SE). That entails all of the algorithm contrasts for video, QoSVisor versus the other two for audio, and only FIFO versus the other two for data. These findings are unsurprising and represent the basic realities confronting all the algorithms; chiefly, penalisation of audio and data in favour of video. They conform to the predictions, as stated in Subsection 8.4.1 (B: Traffic type combined with test duration) for the QoSVisor algorithm. As regard total throughout across all traffic types, these are highly similar for the three algorithms in the upper 60,000s. In contrast to the foregoing pattern, differences can be seen, particularly in the important 1st -order interaction, as shown in Table (8-5) and Table (8-6), where shorter durations are considered.

8.6.2. Analysis of test duration 5 minutes

It can be seen that the Traffic type * Algorithm interaction is similar in form at 5 minutes to what is seen for other durations, it is somewhat stronger and distinctive in form (partial eta-squared 0.784) and (p-value=0.001); it involves traffic type and algorithm.

(Model 3 Adj Rsq 0.566, results being largely independent of bandwidth)

Traffic Type * Algorithm for throughput [p-value =0.001 petasq=0.784]

Table (8-5) Results for throughput in megabits per second (Mbps) measures expressing Traffic Type * Algorithm interaction for 5 minutes duration

Dependent Variable: Throughput

Traffic type	Algorithm	Mean Throughput	Std. Error	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
1.00 Video	1.00 FIFO	16536.844	7375.332	901.838	32171.850
	2.00 Traffic Shaping	34465.388	7375.332	18830.383	50100.394
	3.00 QoSVisor	53104.475	7375.332	37469.470	68739.481
2.00 Audio	1.00 FIFO	17434.975	7375.332	1799.969	33069.981
	2.00 Traffic Shaping	20498.615	7375.332	4863.609	36133.621
	3.00 QoSVisor	5914.243	7375.332	-9720.762	21549.249
3.00 Data	1.00 FIFO	17998.648	7375.332	2363.643	33633.654
	2.00 Traffic Shaping	25733.105	7375.332	10098.099	41368.110
	3.00 QoSVisor	1538.858	7375.332	-14096.148	17173.863

The entry for data traffic type in the TS algorithm appears oddly out of line with the corresponding adjusted mean entries in the separate tables for the longer and shorter durations, having a value of (25,733.105) This is because of the properties of TS using WFQ, as explained in Subsection 8.4.2. (B).

8.6.3. Analysis of test duration 1 minute

It can be seen that the Traffic Type * Algorithm interaction is similar in form at 1 minute to what is seen at other durations. It is somewhat stronger and distinctive in form (partial eta-squared 0.784) and (p-value=0.001), involving traffic type and algorithm.

(Model 3 Rsq adj 0.866; results being largely independent of bandwidth)

Traffic Type * Algorithm for throughput [p-value =0.001 petasq=0.784]

Table (8-6) Results for throughput in megabits per second (Mbps) measures expressing Traffic type * Algorithm interaction for 1 minute duration

Dependent Variable: Throughput

Traffic type	Algorithm	Mean Throughput	Std. Error	95% Confidence Interval Lower Bound	Upper Bound
1.00 Video	1.00 FIFO	21815.346	4015.542	13302.778	30327.915
	2.00 Traffic Shaping	38837.933	4015.542	30325.364	47350.501
	3.00 QoSVisor	56969.824	4015.542	48457.255	65482.392
2.00 Audio	1.00 FIFO	21606.774	4015.542	13094.206	30119.343
	2.00 Traffic Shaping	20899.601	4015.542	12387.032	29412.170
	3.00 QoSVisor	7001.195	4015.542	-1511.373	15513.764
3.00 Data	1.00 FIFO	21728.523	4015.542	13215.954	30241.091
	2.00 Traffic Shaping	6608.666	4015.542	-1903.903	15121.235
	3.00 QoSVisor	3645.957	4015.542	-4866.612	12158.525

8.7. Delay results analysis

In this section, the delay results in milliseconds (ms) are reported for durations 15, 5, and 1 minutes. At 15 minutes duration, the bandwidth factor is significant, whilst at 5 minutes, the effects of algorithm and bandwidth overall are significant. At 1 minute duration the interaction of Traffic type * Algorithm is significant and no others are. The delay in milliseconds (ms) represents the corresponding input to output delays of each part of the electrical signal was measured, that enabled the means would be comparable across delays.

8.7.1. Analysis of test duration 15 minutes

(Model 8 Rsq adj 0.184, where only is bandwidth significant and apparently non-monotonic)

Bandwidth for delay [p-value =0.002 petasq=0.647]

Table (8-7) Results for delay in milliseconds (ms) measures of bandwidth for 15 minutes duration

Dependent Variable: Delay traffic averaged across 10 tests

Bandwidth	Mean delay	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
1.00 40	6.194	11.938	-18.444	30.833
2.00 70	0.164	11.938	-24.475	24.802
3.00 100	43.809	11.938	19.171	68.448

Whilst the overall difference between bandwidth levels in the transmission delay experienced is significant (P-value=0.002) and with a fairly large effect size (petasq=0.647), it can be seen in Table 8-7 that this is apparently non-monotonic. Considering the SE, only the difference between BW 100 and the other two is shown to be significant. These findings are unsurprising and represent the nature of all the algorithms that have been implemented. To confirm this, for delay in (Table 8-8) the algorithm overall effect is significant (p-value=0.008) and (petasq=0.950) (Al-Haddad et al., 2021).

8.7.2. Analysis of test duration 5 minutes

(Model 7 Rsq adj 0.404, where only algorithm and bandwidth overall effects are significant)

Algorithm for delay [p-value =0.008 petasq=0.950]

Table (8-8) Results in milliseconds (ms) for delay measures of the algorithms for 5 minutes duration

Dependent Variable: Delay

Algorithm	Mean delay	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
1.00 FIFO	5.731	2.596	0.348	11.114
2.00 Traffic Shaping	0.845	2.596	-4.538	6.228
3.00 QoSVisor	11.558	2.596	6.175	16.941

At test duration 5 minutes, the overall effect of algorithm on transmission delay is significant (p-value=0.008) and (petasq=0.950). These findings are unsurprising and represent the nature

of all the algorithms that have been implemented. Moreover, the delay climbs even more steeply in data traffic type, (refer to Table 8-16) the mean delay for data traffic type in QoSVisor algorithm shown (57.301 ms) (Al-Haddad et al., 2021).

Bandwidth for delay [p-value=0.002 petasq=0.647]

Table (8-9) Results of delay in milliseconds (ms), for the different bandwidths at 5 minutes duration. The identity of the standard error (SE) to that in the previous table is not a mistake, but rather, a coincidence favoured by the presence of low values and the limited distribution of errors (Al-Haddad et al., 2021).

Dependent Variable: Delay

			95% Confidence Interval	
Bandwidth	Mean delay	Std. Error	Lower Bound	Upper Bound
1.00 40	2.969	2.596	-2.414	8.352
2.00 70	1.508	2.596	-3.875	6.891
3.00 100	13.656	2.596	8.273	19.039

Delay of bandwidth overall effects is significant (p-value=0.002) and (petasq=0.647) at 5 minutes duration, as shown in Table (8-9) above. The pattern of bandwidth effects.

8.7.3. Analysis of test duration 1 minute

(Model 4 Rsq adj 0.945, where the results of delay in milliseconds(ms) are largely independent of bandwidth)

Traffic type * Algorithm for delay [p-value =0.001 petasq=0.784]

Table (8-10) Results for delay in milliseconds (ms) measures expressing Traffic Type * Algorithm interaction for 1-minute duration (Al-Haddad et al., 2021)

Dependent Variable: Delay

				95% Confidence Interval	
Traffic type	Algorithm	Mean delay	Std. Error	Lower Bound	Upper Bound
1.00 Video	1.00 FIFO	0.002	0.114	-0.237	0.241
	2.00 Traffic Shaping	0.001	0.114	-0.239	0.240
	3.00 QoSVisor	0.000	0.114	-0.239	0.239
2.00 Audio	1.00 FIFO	0.002	0.114	-0.237	0.241
	2.00 Traffic Shaping	0.004	0.114	-0.235	0.243

	3.00 QoSVisor	0.052	0.114	-0.187	0.291
3.00 Data	1.00 FIFO	0.002	0.114	-0.237	0.241
	2.00 Traffic Shaping	0.322	0.114	0.083	0.561
	3.00 QoSVisor	2.593	0.114	2.354	2.832

The results obtained in Table (8-10) show Traffic type * Algorithm as being largely independent of bandwidth (p-value=0.001) and (petasq=0.784) as overall effect. The delays at (1-minute) sample length are comparable under QoSVisor for data with the delays met at longer duration and for other traffic types in other conditions. The delay is generally short, but increases by almost 2 SE for data with traffic shaping and to 2.593 with QoSVisor interaction, which is unsurprising in that the more sophisticated queuing systems will constructively allow delay to data as being of low priority.

8.8. Jitter Results Analysis

In this section, Jitter results are reported for durations 15, 5, and 1 minutes in milliseconds (ms). In the 15 minutes duration tests, the results are largely independent of traffic type and the two other 1st order interactions are found to be significant. For 5 minutes duration, both the interactions Bandwidth * Algorithm and Bandwidth * Traffic type are significant, while with duration 1 minute, algorithm overall is not significant, whilst both bandwidth and traffic type are significant in overall-effects-only. Also, the tables of null results for predicted effects are also reported. D-ITG is a network measurement tool used to measure the performance parameters, including jitter, then it has been analysed based on Section 8.2 (page) 137 and Section 8.3 (page 138).

8.8.1. Analysis of test duration 15 minutes

(Model 4 adj Rsq 0.483, where the results for jitter in milliseconds (ms) are *largely independent of traffic type)*

Bandwidth * Algorithm for jitter **[p-value = 0.969]**

Table (8-11) Results for jitter in milliseconds (ms) measures expressing Bandwidth * Algorithm interaction for 15 minutes duration (Al-Haddad et al., 2021)

Dependent Variable: Jitter traffic averaged across 10 tests

Bandwidth	Algorithm	Mean jitter	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 40	1.00 FIFO	0.000	1.925	-4.045	4.046
	2.00 Traffic Shaping	0.001	1.925	-4.044	4.047
	3.00 QoSVisor	0.006	1.925	-4.039	4.051
2.00 70	1.00 FIFO	0.000	1.925	-4.045	4.045
	2.00 Traffic Shaping	0.001	1.925	-4.045	4.046
	3.00 QoSVisor	0.003	1.925	-4.042	4.048
3.00 100	1.00 FIFO	11.593	1.925	7.548	15.639
	2.00 Traffic Shaping	0.015	1.925	-4.030	4.060
	3.00 QoSVisor	3.049	1.925	-0.996	7.094

In Table (8-11), for (duration 15) the overall model (Rsq was 0.483) but the Bandwidth * Algorithm interaction deleted at p-value = 0.969. With an interaction so far from significant it is not meaningful to state its effect size and this is the result summated across all traffic types, with there also being very little jitter. The effect of the algorithm is seen only at widest bandwidth, whilst the other bandwidths have values so low that they do not emerge from the error. The reason for this is that the ordinal predictions for jitter from the FIFO algorithm is limited, as the patterns of jitter are being driven primarily by the partly random properties of the input and all types of traffic compete for the available bandwidth. The average waiting times will be longer, as the process is the same for all traffic types within the repeated traffic samples (Al-Haddad et al., 2021).

8.8.2. Analysis of test duration 5 minutes

*(Model 2 Rsq 0.973, where both interactions Bandwidth*Algorithm & Bandwidth*Traffic type are significant)*

Bandwidth * Algorithm for jitter **[p-value = 0.969]**

Table (8-12) Results for jitter in milliseconds(ms) measures expressing Bandwidth * Algorithm interaction for 5 minutes duration (Al-Haddad et al., 2021)

Dependent Variable: Jitter

Bandwidth	Algorithm	Mean jitter	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 40	1.00 FIFO	0.070	0.147	-0.251	0.391
	2.00 Traffic Shaping	0.001	0.147	-0.320	0.322

	3.00 QoSVisor	0.009	0.147	-0.313	0.330
2.00 70	1.00 FIFO	0.485	0.147	0.164	0.806
	2.00 Traffic Shaping	0.071	0.147	-0.250	0.392
	3.00 QoSVisor	0.002	0.147	-0.319	0.323
3.00 100	1.00 FIFO	5.451	0.147	5.130	5.772
	2.00 Traffic Shaping	0.657	0.147	0.336	0.978
	3.00 QoSVisor	0.361	0.147	0.040	0.682

In table (8-12), (duration 5) the overall model variance explained was high (Rsq 0.973), and the Bandwidth * algorithm interaction was significant. The jitter values are variable and the differences among them below 2SE except that at bandwidths 70 and 100 FIFO stands out from the other algorithms, and at 100 bandwidth the jitter is greater overall with (5.451 ms) mean jitter but interpretation will be left until after the next table (Al-Haddad et al., 2021).

Bandwidth * Traffic Type for jitter [p-value =0.005 petasq=0.685]

Table (8-13) Results for jitter in milliseconds(ms) measures expressing Bandwidth * Traffic Type interaction for 5 minutes duration (Al-Haddad et al., 2021)

Dependent Variable: Jitter

				95% Confidence Interval	
Bandwidth	Traffic type	Mean jitter	Std. Error	Lower Bound	Upper Bound
1.00 40	1.00 Video	0.002	0.147	-0.319	0.323
	2.00 Audio	0.002	0.147	-0.319	0.323
	3.00 Data	0.076	0.147	-0.245	0.397
2.00 70	1.00 Video	0.295	0.147	-0.026	0.616
	2.00 Audio	0.261	0.147	-0.060	0.582
	3.00 Data	0.002	0.147	-0.319	0.323
3.00 100	1.00 Video	1.593	0.147	1.272	1.914
	2.00 Audio	1.820	0.147	1.499	2.141
	3.00 Data	3.056	0.147	2.735	3.377

For jitter (Table 8-13) (duration 5 minutes) the model fit with bandwidth and traffic-type was good (Rsq 0.973), again show more jitter for bandwidth 100 than for other bandwidths, with a slight dependence on traffic type, greatest for data as elsewhere, due to its low prioritisation.

Below is a discussion of algorithms effects on jitter and traffic type at longer durations

High jitter values at high bandwidth are seen for all algorithms (perhaps especially for FIFO) this would be expected from the properties of the algorithms. In QoSVisor, owing to

the LLQ priority scheme and the extension to the Packet Tagging Prioritisation Agent (PTP Agent). This agent is utilised at each switch and the video flow policy allocates the highest (i.e maximum possible from what is available) bandwidth to the highest priority (=1, normally allocated to video).

These allocations have to be envisaged within a relay total network where the same algorithms operate at many stages along a packet path. The audio flow is given second highest priority (priority=2) and hence, it can use the maximum available bandwidth, if there are no packets from the priority=1 flow in the queue. The data traffic type has the lowest priority within the policy (priority=0). This basic priority policy for the three traffic types under any algorithm has been shown to have a real impact on whether the measures of throughput, jitter and delay are deemed to be satisfactory, using simulation techniques, so generally informs priority policies for the three traffic type classes. Some traffic periods (of which some will have been captured within the repeated traffic samples) with high video flow can consume all the available bandwidth (only in QoSVisor and not as a generality for any algorithm), and this arises where there are more video packets than audio and data ones.

In the TS algorithm, three queue-types are created, whatever the current bandwidth (i.e 40, 70, 100) as Q1: high-Expedited Forwarding (EF), allocated to video; Q2: medium-Assured Forwarding (AF), allocated to audio; and Q3: low-Default Forwarding (BE), allocated to data. These queues are formed by the SDN controller used in the system. There is no general inbuilt priority for allocating bandwidth to a specific traffic type, but the WFQ scheme can give the video highest bandwidth allocation. The audio flow can use dynamically the remaining bandwidth after accommodating the video flow as well as the basic bandwidth allocated to audio from the WFQ scheme. Thus, the data flow is expected to show higher delay than the video and audio, but in a pattern also depending on the bandwidth limit set in the evaluation conditions.

For the FIFO algorithm, the ordinal predictions for jitter are limited, as the patterns of jitter will be driven primarily by the partly random properties of the input. There is no priority or ordering scheme for traffic type to enter the flow table, and this will create somewhat equal delay across all the traffic types. All of them will compete for the bandwidth. The average waiting times will be longer than for the other two algorithms and throughout

may not be much lower, but consequently jitter is expected to be high (Al-Haddad et al., 2021).

8.8.3. Analysis of test duration 1 minute

(Model 4 Rsq 0.737, algorithm overall not significant but both bandwidth and traffic type significant in overall-effects-only Rsq 0.324)

Bandwidth * Traffic Type for jitter [p-value =0.005 petasq=0.685]

Table (8-14) Results for jitter in milliseconds(ms) measures expressing Bandwidth * Traffic Type interaction for 1 minute duration (Al-Haddad et al., 2021)

Dependent Variable: Jitter

Bandwidth	Traffic type	Mean jitter	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 40	1.00 Video	0.002	0.147	-0.319	0.323
	2.00 Audio	0.002	0.147	-0.319	0.323
	3.00 Data	0.076	0.147	-0.245	0.397
2.00 70	1.00 Video	0.295	0.147	-0.026	0.616
	2.00 Audio	0.261	0.147	-0.060	0.582
	3.00 Data	0.002	0.147	-0.319	0.323
3.00 100	1.00 Video	1.593	0.147	1.272	1.914
	2.00 Audio	1.820	0.147	1.499	2.141
	3.00 Data	3.056	0.147	2.735	3.377

For 1-minute duration and Jitter as dependent variable, the interaction Bandwidth * Traffic Type (p-value=0.005) and (petasq=0.685) shows that the jitter suffered from any algorithm depends on bandwidth available (Al-Haddad et al., 2021).

8.9. Deviated results for predicted effects

In the adjusted means table for the interaction, (At duration 5 traffic type * algorithm interaction deletes at p=0.1356, Model 3 i.e. not meeting p-value <0.05 retention criterion) and (at duration 15 it deletes at p=0.158 also not meeting p-value <0.05 retention criterion).

*1st instance Traffic Type * Algorithm at Durations 5 and 15, for DELAY measure.*

Traffic Type * Algorithm for delay (Duration 5) [p-value=0.1356 petasq=.784]

Table (8-15) Results for delay in milliseconds (ms) measures expressing Traffic Type * Algorithm interaction for 5 minutes duration (Al-Haddad et al., 2021)

Dependent Variable: Delay

Traffic type	Algorithm	Mean delay	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 Video	1.00 FIFO	7.033	3.905	-1.244	15.311
	2.00 Traffic Shaping	0.035	3.905	-8.243	8.313
	3.00 QoSVisor	1.644	3.905	-6.634	9.921
2.00 Audio	1.00 FIFO	4.052	3.905	-4.226	12.329
	2.00 Traffic Shaping	0.181	3.905	-8.097	8.459
	3.00 QoSVisor	11.754	3.905	3.477	20.032
3.00 Data	1.00 FIFO	6.107	3.905	-2.171	14.384
	2.00 Traffic Shaping	2.318	3.905	-5.960	10.596
	3.00 QoSVisor	21.276	3.905	12.999	29.554

In Table 8-15, the standard error is large and only two of the nine means exceed 2 SE of the two extremely long difference that stand out from those in other conditions, both are with QoSVisor and involve the non-prioritised traffic types, audio and data compared with video. This is entirely deducible from the programmed queueing priorities. FIFO has moderately long delay for all traffic types; this is as predicted because there is no priority or ordering scheme for traffic type to enter the flow table, and this will create somewhat equal delay across all the traffic types, with all of them competing for the bandwidth leading to moderately high delay. Thus, the hypothesis was supported, and the results matched the predicted patterns for QoSVisor and FIFO and TS. Although the error is even larger in the pattern for 15 minutes duration data (below, next page), there is some inconsistency of pattern between FIFO and TS algorithms, the general pattern for QoSVisor with traffic type is consistent with that for shorter test durations.

Traffic Type * Algorithm for delay (Duration 15) [p-value=0.158 petasq=0.784]

Table (8-16) Results for delay in milliseconds (ms), measures expressing Traffic type * Algorithm interaction for 15 minutes duration (Al-Haddad et al., 2021).

Dependent Variable: Delay traffic averaged across 10 tests

Traffic type	Algorithm	Mean delay	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
1.00 Video	1.00 FIFO	9.817	19.249	-30.989	50.622
	2.00 Traffic Shaping	0.089	19.249	-40.716	40.895
	3.00 QoSVisor	0.004	19.249	-40.802	40.809
2.00 Audio	1.00 FIFO	56.480	19.249	15.675	97.285

	2.00 Traffic Shaping	1.315	19.249	-39.490	42.121
	3.00 QoSVisor	11.754	19.249	-29.051	52.559
3.00 Data	1.00 FIFO	2.096	19.249	-38.709	42.901
	2.00 Traffic Shaping	11.645	19.249	-29.160	52.450
	3.00 QoSVisor	57.301	19.249	16.496	98.106

In Table 8-16, video mean delay is 0.004 ms for QoSVisor, and 0.089 ms for the TS algorithm, reflecting the provided ability of these more advanced algorithms to prioritise and minimise video delays. In contrast, the mean delay of data traffic type in the QoSVisor algorithm is very long at 57.301 ms compared to both video and even audio (more than 2 SEs) in QoSVisor. The reversal of the pattern between FIFO and QoS Visor for audio versus data is striking. The long delay could perhaps be seen as a limitation of the QoSVisor policy, it being a direct consequence of the protection of video and some further prioritising system within data (eg urgent and non-urgent) would be needed, if this is seen as genuinely a consequent limitation to be got around somehow. In general, the results have confirmed the predictions for traffic type compare with test durations across the three algorithms. At the shortest test duration of 1 minute, the results have not confirms the predictions.

But this may be due to high relative variability, with this due in turn to inability of the algorithms to get into their stride given their buffer lengths and the small number of packets of each type of traffic, and of typical length for their type, within the one minute.

8.10. Discussion and summary

This chapter has reported a novel approach for evaluating the queueing systems in packet switching in SDN. To achieve this, the mean results for 10 trials in each combination of design variables were pre-extracted for each of the three performance measures and tabulated with regard for reliability according to a standard analysis of variance (in the SPSS statistical package). It is relatively unusual in communications engineering or computer science to follow up the implementation of algorithms with such a systematic and formal evaluative account of how detailed patterns of performance do, indeed, result from the strategic policy decisions, which led to development of the classes of algorithm.

Researchers are often content with one main ordinal comparison, assuming high general evaluation for the single performance index used. It is also unusual to do so within a comprehensive balanced design ('factorial'). varying several aspects of the traffic as well as the algorithm. The fact that the predictions are largely confirmed is not to be seen as a surprising new finding, as would be the typical interpretation in empirical science, because in each instance the code has been shown to function in a software sense (See Chapter 7, Section 7.3, page 120): The operational stages of the proposed architecture). Rather, the differences in mean delay, jitter and throughput have demonstrated on typical internet traffic the magnitude of gains achieved by the algorithms and the trade-offs with various, perhaps more tolerable, losses. In order to embrace several more detailed research hypotheses, Chapter 1 (Section 1.3) stated the overall aim for this research in the form of a general claim: *An enhanced Quality of Service (QoS) framework in Software-Defined Networks (SDN) provides a suitable solution for solving congestion and latency problems, thus leading to measurably augmented QoS for streamed data in networks.* Technically and theoretically, FlowVisor is a software slicing tool plus special purpose controller used in the testbed to create slices on the basis of different types of traffic, which allows the algorithms to achieve the various patterns of performance that they do.

FlowVisor's major contributions include the possibility of slicing any control plane message format used in OpenFlow (OF) and being the first slicing mechanism that enables a user-defined control plane to exercise control over the production hardware's forwarding behaviour. The exact decisions on slicing in optimising use of the network's resources have to occur in the context of the available bandwidth, the stored forwarding table entries, the pre-defined custom topology, and within the limits of the device's CPU of the traditional network. Depending on FlowVisor, a 'policy language' is used to trace the network flows to slices, and this provides flexibility to the user, for example, in engaging with new algorithms for improved services, which can be stored in OpenFlow switches. Comparisons of parameters of performance on a uniform testbed showed differences among the three algorithms implemented to support this claim.

This consistency of results has shown that the more advanced algorithms of TS and QoSVisor (compared to FIFO) do, indeed, permit more advanced allocation of bandwidth, and that they reduce critical delays. Moreover, this research has provided a new methodological contribution via the testbed enabling actual measures of the performance

parameters, thereby allowing each algorithm to be evaluated. Using delay, jitter and throughput for the three traffic types is new compared to previous studies that used a small set of parameters to measure the performance, these studies are illustrated in Chapter 3.

Haiyan, et al. (2016) derive a model of queue delay from network parameters, they stated that the delay in a queue is the primary explanation for the latency of the network, since packet processing is negligible, and propagation is constant. Similarly, Rowshanrad, Namvarasl, and Keshtgari (2017) only selected the difference with the throughput to measure queue delay. While Wallner and Cannistra (2013) used queue-based classification strategies to provide QoS support for floodlight-controlled SDN networks, but to measure QoS metrics, their work requires a crucial and detailed assessment.

This research contributes to the methodology for the evaluation of digital telecommunications by comprehensively gathering performance data within an experimental design. Furthermore, the use of the analysis of variance on subsets of the conditions in the experimental design enabled the examination (in terms both of statistical significance and effect size) of apparent differences in the performance parameters and of their dependences (i.e. statistical interactions among the independent variables) regarding other conditions. The statistical analyses permit degrees of certainty to be attached to the comparisons between algorithms.

The enquiry began with the posing of the research question: *"How to establish a QoS framework that can help solve the congestion, resource allocation and delay problems found in networks, based on slicing technology within SDN?* The subsequent implementation and evaluation phases have led to applicable findings, amounting to the achievement of a QoS goal. Under FIFO, throughput is largely homogeneous across data classes, whilst under traffic-shaping, video is clearly prioritised above audio and data. Under the QoSVisor algorithm the expected gains of the same general type were shown, only more so, particularly for video traffic.

A major sub-question of this research was posed in chapter 1, Section 1.3: "*Can SDN help to reduce the network congestion, resource allocation and delay problems found in networks for delay-sensitive traffic, such as video and/or audio applications?".* In SDN, the "new separation concept" means that the control plane can reside outside the networking

device and can be developed from one or multiple controllers, with the precise number of controllers being determined by the network size. Furthermore, this separation of planes allows network engineers to handle network protocols and the resulting services as software. The data plane, separated from the control plane, receives information and requests from the latter and implements these in the hardware to the extent needed (Al-Haddad et al., 2021; Rowshanrad et al., 2014). Moreover, the slicing that is used in the testbed, works as a transparent proxy in OpenFlow; it adds a new mechanism in the form of a software-slicing layer, sitting between the network devices' forwarding and control planes (Sherwood et al., 2010).

This study has found that for delay measures as the dependent variable, for all the three durations of test period and despite some slight differences in what other effects were significant, the interaction Traffic type * Algorithm was always significant, and the table of means displayed a consistent form. Next, the results for jitter did not show the pervasive Algorithm * Traffic type interaction that was seen for delay. That is, it was largely independent of the algorithm, that you might not expect it to be so even at the very short duration of 1 minute but both bandwidth and traffic type overall-effects were significant. This is consistent with how the algorithms use bandwidth and have limits overall effects on jitter. However, high jitter values at high bandwidth are seen for all the algorithms (perhaps especially for FIFO), which is to be expected given their properties.

Finally, the results for throughput measures as the dependent variable have clearly shown the increasing capacity with high bandwidth in transmitting video, as well as the overall greater throughput of this data type. For duration 15 minutes, with the results largely independent of the algorithm, the interactions Bandwidth * Traffic type, and Traffic type * Algorithm were significant. With durations 1 and 5 minutes, the results were also largely independent of bandwidth, but the Traffic type * Algorithm term was always significant. Throughput for TS and QoSVisor was similarly high for the two most demanding and thus, prioritised traffic types, as generally expected. Moving to lower-priority audio then data we saw that TS lets through just over twice the amount of information compared to QoSVisor, and then comparing audio with data there was again an approximately five-fold advantage for audio related to data. This is a direct consequence of the algorithms as programmed, with the bit rate weight policy higher for audio than for data in both TS and the proposed prioritisation agent, namely QoSVisor.

9. Chapter Nine: Conclusion and Future work

9.1. Conclusion

In this thesis, the challenges regarding traffic performance in the SDN network caused by the increasing demand for network services and quality across extensive selections of digital applications on the internet have been investigated. New methodologies of QoSVisor and TS Algorithms and Packet Tagging Prioritisation Agent (PTP Agent) extension have been proposed to measure to exactly to what extent the intended benefits of using a slicing and queueing discipline established in WFQ in SDN are realised in practice, also to implement and testify to the strict priority policy that we added to this algorithm.

For the first proposed approach, the FIFO algorithm was developed and implemented in a sliced-SDN framework as a baseline condition for quantitative performance comparisons, with detailed implementation of the template design for this algorithm module in a sliced-SDN Testbed. The particular traffic measures (throughput, delay and jitter), which are separate performance indices all contributing to QoS are integrated to provide an objectively rooted but overall and subjectively confirmable metric of QoS for each switch. Floodlight and FlowVisor controllers as well as OpenFlow (OF) switches, which comprise characteristic behaviour of SDN, have been modelled and simulated via a Mininet testbed emulator platform. A custom topology has been used with five switches, all the topology being connected to Floodlight and FlowVisor controllers, with the ingress bound interface switch (S1) being connected to three hosts. At the same time, the outbound interface switch (S4) was also connected to three hosts. The server(s) processed the queued data at a rate of 40, 70 and 100 b/s as the parameters chosen for the designed study, representing the limiting speed of the outbound interface and ten replicate measurements were taken.

The simulation was run for different timescale parameters, both those inherent to the timescales of the operation within the algorithm for five seconds intervals for each test and those in the sample lengths as well as the scheme of rotation across the defined stress test

conditions of 1, 5 and 15 minutes. These timescale parameters were applied to all three switching algorithms of the simulation techniques: FIFO, TS and QoSVisor.

For the second proposed approach, (TS) algorithm has been proposed as a new contribution to work as a bandwidth management technique to optimise performance in an sliced-SDN network and to overcome the limitation observed in FIFO queuing of buffer overflow. Two algorithms, namely "Algorithm 6-2: Packet tagging, Queueing, Forwarding to Queues" and "Algorithm 6-2-A: Allocate Bandwidth" have been proposed to implement a weighted fair queuing (WFQ) technique as a new methodology in a sliced-SDN testbed. Traffic shaping works mainly via WFQ part-function of TS the queueing mechanism to reduce congestion and smooth traffic flow. This methodology is used for (QoS) and does two things simultaneously: making traffic conform to an individual rate by using WFQ to decide the appropriate queue for each packet; and combining the methodology with buffer management that decides whether to put the packet into the queue, according to Algorithm (6-2) defined for this purpose. During this tagging and queueing, the bandwidth is managed based on Algorithm 6-2-A: Allocate Bandwidth. By so doing, the latency and congestion remain in check, thus, meeting the requirements of real-time services. According to the particular traffic measures (throughput, delay and jitter), which are separate performance indices all contributing to QoS, which is integrated to provide an objectively rooted but overall and subjectively confirmable metric of QoS for each switch.

The Differentiated Service (DiffServ) protocol is used to define classes in order to make network traffic patterns more sensitive to traffic class, by specifying precedence for each traffic type. A floodlight and FlowVisor controllers, and OpenFlow (OF) switches, which is characterise the behaviour of SDN, have been modelled and simulated via a Mininet testbed emulator platform. A custom topology was used in this research with five switches, all the topology being connected to the floodlights and FlowVisor controllers and the ingress bound interface switch (S1) connected to three hosts. At the same time, switch (S4) performed the traffic shaping by having three queues in the output interface also connected to the three hosts, while the server (s) processed the queued data at rates of 40, 70, 100 (b/s) as the parameters chosen for the designed study, representing the limiting speed of the outbound interface and ten replicate measurements were taken. The network was run in timescale parameters for those in the sample lengths and the scheme of rotation between those stress

test conditions as 1, 5, and 15 minutes to study the network performance, according to the implemented algorithm.

For the third proposed approach, a new methodology of QoSVisor and a Packet Tagging Prioritisation Agent (PTP Agent) extension algorithm for video, audio and data over TCP sliced-SDN networks has been developed and tested. This was to overcome the limitations that exist with the traditional FIFO queueing method. Specifically, the algorithm is aimed at reducing the delay and jitter of the packet transmission process, as witnessed with the standard algorithm FIFO, whilst at the same time maintaining or even improving the results obtained from the traffic shaping algorithm.

The implementation of this distinction was via a local weighting function, which sorts packets by the port and flow policy setup. This enables the use of more advanced queueing systems, for which more than one stage is necessary, along with extra memory capacity for handling long queues. Depending on each packet (flow, packet ID and weighted tag), the weighted tag is the weighting assigned to the queue based on the traffic type, which is derived from the element in WFQ called WFQ components.

The more advanced queueing algorithms have developed based on the following proposed algorithms: Packet Tagging and Forwarding, Tagging and queueing, and Packet Scheduling algorithms. These have been presented in Chapter 6, to meet the QoS in order to filter pre-classified packet data to prioritise those packets for delivery, and to achieve the best performance for applications using different queueing techniques in an SDN.

The objective of the proposed PTP Agent is to extend the current internal operation of FlowVisor with an enhanced QoS model. This will mean guaranteed and more precise QoS for different applications on a continuous basis, even during peak congestion times, and with priorities able to be agreed for specific requirements. In general, the proposed design is based on gathering four main technologies together: traffic engineering (TE), SDN, network hypervisors and network slicing. A new modification model is proposed for enhancing the current FlowVisor to meet the requirements of improving the QoS classification within a sliced-SDN. The proposed algorithm employs similar techniques to those used in the TS algorithm, with some enhancements that take into consideration the limitations of network devices.

To achieve the highest level of accuracy, a novel comparative approach for evaluating FIFO queueing, TS, and QoSVisor systems in packet switching in SDN has been proposed. The SPSS statistical package Analysis of Variance (ANOVA) was used to perform the first level of analysis of the data, with various tests of hypotheses for benefit to quality. Generally speaking, this involves testing for continuous or categorical relationships between conditions set up in the study design as independent variables and performance outcomes (dependent variables). The analyses involved various ways of collapsing means of 10 repeated measures in each of the cells in the 243-cell structure of data (3 traffic types) X (3 algorithms) X (3 durations) X (3 bandwidths) X (3 measures parameters). The advantage of three levels is that it provides at least a preliminary indication as to whether relationships may be linear or not. The analysis provided an overall performance comparison between the three algorithms, and some in-depth identification of the differences between them. The most general first analysis of such data structures for the last 80 years has been the Analysis of Variance (ANOVA) or F-test (Rutherford, 2001), which can handle a 3-level categorical independent variable. Often such analysis involves comparisons of two conditions, using the Independent Samples t-Test. Rather than then applying formal rules for conservative thresholds, when there are three possible comparisons between the two levels of a categorical design variable, the researcher adopted the convention of avoiding describing single results as categorically 'significant' or not, but rather, handled the results near the conventional margin of significance as what they are. She also made an overview of parallel contrasts in other parts of the data-structure and avoided interpreting isolated ones, which can be confusing regarding what the dataset as a whole is letting on.

The experimental results presented with the corresponding statistical analysis have shown that depending on FlowVisor, a 'policy language' is used to trace the network flows to slices, and this provides flexibility to the user, for example in attempting new algorithms for improved services, which can be stored in OpenFlow switches. Comparisons of parameters of performance on a uniform testbed showed differences among the three algorithms implemented to support this claim. The findings have shown that the more advanced algorithms, namely TS and QoSVisor do, indeed, permit more advanced allocation of bandwidth, and that they reduce critical delays. This is unlike FIFO, the application of which resulted in network congestion in the form of buffer overflow leading to network delay. Moreover, this research provided a new methodological contribution via the testbed

enabling actual measures of the performance parameters and allowing each algorithm to be evaluated.

Using only delay, jitter and throughput for the three traffic types is a novel approach not found in previous studies that used a small set of parameters to measure performance. The consequent implementation and evaluation phases have led to findings that have demonstrated the achievement of the QoS goal.

With the FIFO algorithm, throughput was largely homogeneous across data classes, whilst under traffic-shaping, video was clearly prioritised above audio and audio above data. Under the QoSVisor algorithm expected gains of the same general type were elicited, with further gains particularly for video traffic. It emerged that for delay measures as the dependent variable, over all the three durations of test period and despite some slight differences in what other effects were significant, the interaction Traffic type * Algorithm was always significant, and the table of means displayed a consistent form.

Next, the results for jitter did not show the pervasive Traffic type*Algorithm interaction that was seen for delay and that jitter was largely independent of the algorithm, that you might not expect it to be so even at the very short duration of 1 minute but both bandwidth and traffic type overall-effects were significant. This is consistent with how the algorithms use bandwidth and have limited overall effects on jitter. However, high jitter values at high bandwidth were seen for all the algorithms (in particular for FIFO), which was to be expected given their properties.

Finally, the results for throughput measures as the dependent variable have clearly shown increasing capacity with high bandwidth in transmitting video, as well as the overall greater throughput of this data form. For a duration of 15 minutes, with the results largely independent of the algorithm, the interactions Bandwidth * Traffic type, and Traffic type * Algorithm were significant. With durations of 1 and 5 minutes, the results were also largely independent of bandwidth, but the Traffic type * Algorithm term was always significant.

Throughput for TS and QoSVisor was similarly high for the two most demanding bandwidth and so prioritised traffic types as generally expected. Moving to lower-priority audio then data it was elicited that TS lets through just over twice the amount of information

compared to QoSVisor. Then comparing audio with data there was again an approximately five-fold advantage for audio related to data. This is a direct consequence of the algorithms as programmed with the bit rate weight policy higher for audio than for data in both TS and the proposed prioritisation agent of QoSVisor. The interaction Traffic type * Algorithm in QoSVisor is utilised at each switch and the video flow policy allocates the highest bandwidth with highest priority (priority=1) in a relay total network with these algorithms operating at many stages along a packet path. While the prediction for TS, according to the WFQ scheme, as explained in Subsection 8.2.2.2/A, was supportive and matched the results for all test durations.

9.2. Future work

The research contributions achieved through this research open up new research avenues for QoS-based SDN networks aimed at improved queuing and bandwidth management techniques that involve machine learning (ML). Specifically, a project is proposed to compare the performance between QoSVisor sliced-SDN networks and the deep learning (DL) C5.0 Decision Tree algorithm. The SDN template constructed within this research could be used as a basis to implement the DL algorithm to meet the QoS requirements in relation to network implementation, both real and virtual, using a variety of SDN topology selections, such as mesh, fat-tree topology, or custom topology, with up to 60 switches.

Due to SDN's simplicity for developing and implementing new algorithms, as has been shown in this research, this opens up a new future objective: maximising network intelligence. Intelligent SDN network implementation using DL routing optimisation for traffic prediction will play a significant role in intelligent routing management and planning, resource management, traffic scheduling and flow control, among other future network building and management tools. Accurate traffic predicting will allow for the congestion problems on links to be addressed before they affect the QoS and QoE by proposing new routes that result in the traffic being managed better.

Another research path that requires serious investigation is ML for an SDN-based traffic classification system. Classification of traffic requires encrypted flow packets that mask flow features. For this classification, advanced deep-learning methods need to be deployed to

generate patterns using large quantities of training data and to predict the host's bandwidth behaviour (Al-Haddad et al., 2021).

SDN has potential application to the field of network virtualisation technology investigation, such as virtualisation networking functions (NFV) designed to increase wireless networking, (5G), (6G) network technologies and the Internet of Things (IoT). Given the lack of understanding regarding these technologies, they require further probing on different aspects, such as how to engender energy efficiencies and minimise congestion. There has also been a lack of research on QoS, scalability, reliability, extensibility and SDN service management. Programming and performance problems must be resolved in SDN, if 100 Gb/s and higher speeds are to be dealt with effectively. The need for network security sensors built into the controller is pressing, as SDN controller overload with malicious flows causes a bottleneck problem, which impacts negatively on network QoS, especially with the deployment of large scale SDN networks. Deep learning algorithms could also be used in network security sensors to detect suspicious flows and abnormal attacks, thereby increasing the scalability of QoS, quality of experience (QoE) and network control.

Another future objective is controller scalability and improving networking protocol security. Controller scalability is also a problem that has to be addressed regarding large SDNs, failing which this can result in a complicated optimisation problem that can adversely affect fault tolerance, latency, and capacity. Whilst some studies have taken into account control plane design, others have examined the controller’s design for enhancing the native switch. SDN researchers must also address security concerns that have led to more debates about such challenges. To deliver secure communication, it is necessary to ensure that key security characteristics are taken into consideration, including confidentiality, availability, non-repudiation, authentication, and integrity (Essafi, Labed, and Ghezala, 2006). As noted by Ranjbar et al. (2016), SDN applications can be developed for enhancing other applications and/or networking protocol security.

9.3. List of Publications and Presentations

9.3.1. List of Publications

- Al-Haddad, R., Velazquez, E., Fatima, A., Winckles, A., 2021. A Novel Traffic Shaping Algorithm for SDN-Sliced Networks using New WFQ Technique. *International journal for advanced computer science and applications (IJACSA).*12(1). http://dx.doi.org/10.14569/IJACSA.2021.0120101.
- Al-Haddad R., Velazquez E.S. (2019) A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN). In: Arai K., Kapoor S., Bhatia R. (eds) Intelligent Computing. SAI 2018. Advances in Intelligent Systems and Computing, vol 857. Springer, Cham.
- Al-Haddad, R. and Velazquez, E.S., 2018, July. A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN). In *Science and Information Conference* (pp. 527-545). Springer, Cham.
- Al-Haddad, R., Velazquez, E.S. and Winckles, A., 2017, July. QoSVisor: QoS framework for SDN. In *Computing Conference*, (pp. 1290-1296). IEEE. Doi: 10.1109/SAI.2017.8252257.

9.3.2. List of Seminars, presentaions and posters

- Alhaddad, R., Sanchez Velazquez, E., 2018, A new framework based on Software-Defined Networks to support QoS in a sliced architecture. In: Anglia Ruskin University/ ICE Research Seminar series for PhD students (Presentation), School of Computing and Information Science /Cambridge, UK.
- Al-Haddad, R. and Velazquez, E.S., 2018, July. A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN). (Presentation), In Science and Information Conference/London, UK.
- Alhaddad, R., Sanchez Velazquez, E., 2017, New framework based on Software-Defined Networks to support QoS in a sliced architecture. In: Anglia Ruskin

University/ ICE Research Seminar series for PhD students (Presentation), School of Computing and Information Science /Cambridge, UK, 13th of December 2017.

- Al-Haddad, R., Velazquez, E.S. and Winckles, A., 2017, July. QoSVisor: QoS framework for SDN. (Presentation), In Computing Conference /London, UK.
- Alhaddad, R., Sanchez Velazquez, E., 2016, QoSVisor: Action QoS Slicer based SDN. In: Anglia Ruskin University/ VSIRG (Poster), Department of Computing and Technology /Chelmsford, UK, 14th of April 2016.
- Alhaddad, R., 2016, QoSVisor: QoS Framework for SDN. In: Anglia Ruskin University/ VSIRG Seminars Series (Presentation), Department of Computing and Technology /Chelmsford, UK, 13th of December 2016.

9.3.3. List of ARU annual research conferences

Alhaddad, R., Sanchez Velazquez, E., 2016, QoSVisor: Action QoS Slicer based SDN. In: Anglia Ruskin University/Poster, FST, 10th Annual Research Student Conference, Chelmsford, UK, 17th June 2016.

Alhaddad, R., Sanchez Velazquez, E., Winckles, A., 2016, New design based on Software Defined Networks to support QoS in a sliced architecture. In: Anglia Ruskin University/Poster, FST, 6th Annual Research and Scholarship Conference, Chelmsford, UK, 6th July 2016.

References

Abuarqoub, A., 2020. A Review of the Control Plane Scalability Approaches in Software Defined Networking. *Future Internet,* 12(3), p.49.

Acharya, H.S., Dutta, S.R. and Bhoi, R., 2013. The Impact of self-similarity Network traffic on quality of services (QoS) of Telecommunication Network. *International Journal of IT Engineering and Applied Sciences Research (IJIEASR),* 2, pp.54-60.

Agarwal, S., Kodialam, M. and Lakshman, T.V., 2013, April. Traffic engineering in software defined networks. In *2013 Proceedings IEEE INFOCOM* (pp. 2211-2219). IEEE.

Ahmed, M., Huici, F. and Jahanpanah, A., 2012. Enabling dynamic network processing with clickOS. *ACM SIGCOMM Computer Communication Review*, *42*(4), pp.293-294.

Akin, E. and Korkmaz, T., 2019, May. Link-Prioritized Network State Information Collection in SDN. *In ICC 2019-2019 IEEE International Conference on Communications (ICC)* (pp. 1-7). IEEE.

Akiyama, T., Kawai, Y., Teranishi, Y., Banno, R. and Iida, K., 2015, July. SAPS: Software Defined Network Aware Pub/Sub--A Design of the Hybrid Architecture Utilizing Distributed and Centralized Multicast. *In Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual (Vol. 2*, pp. 361-366). IEEE.

Akyildiz, I.F., Lee, A., Wang, P., Luo, M. and Chou, W., 2014. A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, *71*, pp.1-30.

Alabarce, M.G. and Bravalheri, A., 2018, July. Overview of South-Bound Interfaces for Software-Defined Optical Networks. In *2018 20th International Conference on Transparent Optical Networks (ICTON)* (pp. 1-5). IEEE.

Al-Haddad, R. and Sanchez, E., 2018. A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN). *Proceedings of the 2018 Computing Conference*, [e-journal] 857, pp. 527 - 545. https://doi.org/10.1007/978-3-030-01177-2_38.

Al-Haddad, R. and Velazquez, E.S., 2018, July. A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN). In *Science and Information Conference* (pp. 527-545). Springer, Cham.

Al-Haddad, R., Velazquez, E., Fatima, A., Winckles, A., 2021. A Novel Traffic Shaping Algorithm for SDN-Sliced Networks using New WFQ Technique. In *International journal for advanced computer science and applications (IJACSA)*, 12(1). https://dx.doi.org/10.14569/IJACSA.2021.0120101.

Al-Haddad, R., Velazquez, E.S. and Winckles, A., 2017, July. QoSVisor: QoS framework for SDN. *In 2017 Computing Conference* (pp. 1290-1296). IEEE.

Allakany, A.M. and Okamura, K., 2017, June. Latency monitoring in software-defined networks. *In Proceedings of the 12th International Conference on Future Internet Technologies* (pp. 1-4).

Alsaeedi, M., Mohamad, M.M. and Al-Roubaiey, A.A., 2019. Toward adaptive and scalable OpenFlow-SDN flow control: A survey. *IEEE Access, 7,* pp.107346-107379.

Al-Shabibi, A., De Leenheer, M., Gerola, M., Koshibe, A., Snow, W. and Parulkar, G., 2014. Openvirtex: A network hypervisor. In *Open Networking Summit 2014 ({ONS} 2014)*.

Altukhov, V. and Chemeritskiy, E., 2014, October. On real-time delay monitoring in software-defined networks. In *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)* (pp. 1-6). IEEE.

Azodolmolky, S., 2013. *Software defined networking with OpenFlow*. Packt Publishing Ltd.

Badotra, S. and Panda, S.N., 2020. Software-Defined Networking: A Novel Approach to Networks. In *Handbook of Computer Networks and Cyber Security* (pp. 313-339). Springer, Cham.

Baker, F., Black, D., Blake, S. and Nichols, K., 1998. Definition of the Differentiated Services Field (DS field) in the IPv4 and IPv6 headers. RFC 2474, Dec.

Bakhshi, T., 2017. State of the art and recent research advances in software defined networking. *Wireless Communications and Mobile Computing*, *2017*.

Bakshi, K., 2013, March. Considerations for software defined networking (SDN): Approaches and use cases. In *2013 IEEE Aerospace Conference* (pp. 1-9). IEEE.

Balakrishnan, R., 2012. *Advanced QoS for multi-service IP/MPLS networks*. John Wiley & Sons.

Balogh, T. and Medvecký, M., 2010. *Comparison of Priority Queuing Based Scheduling Algorithms*. Elektrorevue, 15, p.11.

Bari, M.F., Chowdhury, S.R., Ahmed, R. and Boutaba, R., 2013, November. PolicyCop: An autonomic QoS policy enforcement framework for software defined networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)* (pp. 1-7). IEEE.

Belter, B., Binczewski, A., Dombek, K., Juszczyk, A., Ogrodowczyk, L., Parniewicz, D., Stroiński, M. and Olszewski, I., 2014, September. Programmable abstraction of datapath: Advanced programmability of heterogeneous datapath elements through hardware abstraction. In *2014 Third European Workshop on Software Defined Networks* (pp. 7-12). IEEE.

Belter, B., Binczewski, A., Dombek, K., Juszczyk, A., Ogrodowczyk, L., Parniewicz, D., Stroiński, M. and Olszewski, I., 2014, September. Programmable abstraction of datapath. In *2014 Third European Workshop on Software Defined Networks* (pp. 7-12). IEEE.

Benzekki, K., El Fergougui, A. and Elbelrhiti Elalaoui, A., 2016. Software-defined networking (SDN): a survey. *Security and communication networks, 9(18),* pp.5803-5833.

Binsahaq, A., Sheltami, T.R. and Salah, K., 2019. A survey on autonomic provisioning and management of QoS in SDN networks. *IEEE Access, 7*, pp.73384-73435.

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

Blenk, A., Basta, A., Reisslein, M. and Kellerer, W., 2015. *Survey on Network Virtualization Hypervisors for Software Defined Networking.*

Bosshart, P., Gibb, G., Kim, H.S., Varghese, G., McKeown, N., Izzard, M., Mujica, F. and Horowitz, M., 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review, 43(4),* pp.99-110.

Botta, A., Dainotti, A. and Pescapè, A., 2007. *Multi-protocol and multi-platform traffic*.

Botta, A., de Donato, W., Dainotti, A., Avallone, S. and Pescape, A., 2013. D-ITG 2.8. 1 Manual. *Computer for Interaction and Communications (COMICS) Group*, Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy (www. grid. unina. it/software/ITG/manual).

Boughzala, B., Ali, R.B., Lemay, M., Lemieux, Y. and Cherkaoui, O., 2011, May. OpenFlow supporting inter-domain virtual machine migration. In *2011 Eighth International Conference on Wireless and Optical Communications Networks* (pp. 1-7). IEEE.

Bozakov, Z. and Papadimitriou, P., 2012, December. Autoslice: automated and scalable slicing for software-defined networks. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop* (pp. 3-4).

Braden, R., Clark, D. and Shenker, S., 1994. Integrated Services in the Internet architecture: An overview. *Internet Engineering TaskForce.* Request for Comments. RFC 1633. [Online] Available FTP: ds. internic. net/rfc/rfc1633. txt.

Braden, R., Zhang, L., Berson, S., Herzog, S. and Jamin, S., 1997. Resource ReSerVation Protocol:(RSVP); Version 1 Functional Specification.

Braga, R., Mota, E. and Passito, A., 2010, October. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *IEEE Local Computer Network Conference* (pp. 408-415). IEEE.

Braun, W. and Menth, M., 2014. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet*, *6*(2), pp.302-336.

Cao, Z., Kodialam, M. and Lakshman, T.V., 2014, August. Traffic steering in software defined networks: Planning and online routing. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing* (pp. 65-70).

Carlos, A.M., Rothenberg, C.E. and Maurício, F.M., 2010, December. In-packet Bloom filter based data center networking with distributed OpenFlow controllers. In *2010 IEEE Globecom Workshops* (pp. 584-588). IEEE.

Ching-Hao, C. and Lin, Y.D., 2015. *OpenFlow Version Roadmap*. tech. rep, 2015. http://speed. cis. nctu. edu. tw/~ ydlin/miscpub/indep_frank. pdf.

Cisco, 2013. Connections Counter Connections Counter: The Internet of Everything in Motion. [Online] Available at: < https://newsroom.cisco.com/feature-content?articleId=1208342> [Accessed 5 April 2015].

Cisco, 2019. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. [pdf] Available at: < https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf> [Accessed 20 July 2019].

Clark M. P., 2003. *Data Networks, IP and the Internet Protocols, Design and Operation.* Chichester, West Sussex, England: John Wiley & Sons Ltd.

Clough, P. and Nutbrown, C., 2012. *A student's guide to methodology*. Sage.

Comer, D.E., 2018. *The Internet book: everything you need to know about computer networking and how the Internet works*. CRC Press.

Costa, V.T., and Costa, L.H.M., 2015. Vulnerabilities and solutions for isolation in FlowVisor-based virtual network environments. *Journal of Internet Services and Applications, 6(1),* pp.1-9.

Cox, J.H., Chung, J., Donovan, S., Ivey, J., Clark, R.J., Riley, G. and Owen, H.L., 2017. Advancing software-defined networks: A survey. *IEEE Access*, *5*, pp.25487-25526.

Das, S., Yiakoumis, Y., Parulkar, G., McKeown, N., Singh, P., Getachew, D. and Desai, P.D., 2011, March. Application-aware aggregation and traffic engineering in a converged packet-circuit network. In *National Fiber Optic Engineers Conference* (p. NThD3). Optical Society of America.

Davie, B., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Gude, N., Padmanabhan, A., Petty, T., Duda, K. and Chanda, A., 2017. A database approach to sdn control plane design. *ACM SIGCOMM Computer Communication Review*, *47*(1), pp.15-26.

Daniels, J., Werner, P.W. and Bahill, A.T., 2001. Quantitative methods for tradeoff analyses. *Systems Engineering, 4(3),* pp.190-212.

Dawson, C., 2009. *Introduction to Research Methods: A Practical Guide for Anyone Undertaking a Research Project* (4th ed.). Oxford: How to Books.

Day, J.D. and Zimmermann, H., 1983. *The OSI reference model. Proceedings of the IEEE, 71(12),* pp.1334-1340.

Dekeris, B., Adomkus, T. and Budnikas, A., 2006, June. Analysis of QoS assurance using weighted fair queueing (WQF) scheduling discipline with low latency queue (LLQ). In *28th International Conference on Information Technology Interfaces*, 2006. (pp. 507-512). IEEE.

Depaoli, D., Doriguzzi-Corin, R., Gerola, M. and Salvadori, E., 2014, September. Demonstrating a distributed and version-agnostic OpenFlow slicing mechanism. In *2014 Third European Workshop on Software Defined Networks* (pp. 133-134). IEEE.

Desai, A., Oza, R., Sharma, P. and Patel, B., 2013. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering, 2(3),* pp.222-225.

Doria, A., Salim, J.H., Haas, R., Khosravi, H.M., Wang, W., Dong, L., Gopal, R. and Halpern, J.M., 2010. Forwarding and Control Element Separation (ForCES) Protocol Specification. *RFC*, *5810*, pp.1-124.

Doriguzzi-Corin, R., Salvadori, E., Gerola, M., Suñé, M. and Woesner, H., 2014, September. A datapath-centric virtualization mechanism for OpenFlow networks. In *2014 third European workshop on software defined networks* (pp. 19-24). IEEE.

Duan, Q., Yan, Y. and Vasilakos, A.V., 2012. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Transactions on Network and Service Management*, *9*(4), pp.373-392.

Egilmez, H.E., Dane, S.T., Bagci, K.T. and Tekalp, A.M., 2012, December. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In *Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference* (pp. 1-8). IEEE.

Englert, M. and Westermann, M., 2009. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica, 53(4),* pp.523-548.

Essafi, M., Labed, L. and Ghezala, H.B., 2006, October. ASASI: An environment for addressing software application security issues. In *2006 International Conference on Systems and Networks Communications (ICSNC'06)* (pp. 19-19). IEEE.

Fabiano, A.J. and Qiu, J., 2015. Post-stereotactic radiosurgery brain metastases: a review. *Journal of neurosurgical sciences*, *59*(2), pp.157-167.

Fang, S., Yu, Y., Foh, C.H. and Aung, K.M.M., 2013. A loss-free multipathing solution for data center network using software-defined networking approach. *IEEE transactions on magnetics*, *49*(6), pp.2723-2730.

Fatima, A., 2016. An integrated architecture for semantic search (Doctoral dissertation, Anglia Ruskin University).

Feamster, N., Gao, L. and Rexford, J., 2007. How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communication Review*, *37*(1), pp.61-64.

Feamster, N., Rexford, J. and Zegura, E., 2014. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review, 44(2),* pp.87-98.

Fernandez, C. and Munoz, J.L., 2015. Software Defined Networking (SDN) with OpenFlow 1.3 Open vSwitch and Ryu. *UPC Telematics Department*_PhD Thesis.

Fernandez, M.P., 2013, March. Comparing openflow controller paradigms scalability: Reactive and proactive. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 1009-1016). IEEE.

Floodlight controller, http://www.projectfloodlight.org/floodlight/.

Floyd, S. and Jacobson, V., 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking, 1(4),* pp.397-413.

Foster, N., Harrison, R., Freedman, M.J., Monsanto, C., Rexford, J., Story, A. and Walker, D., 2011. Frenetic: A network programming language. *ACM Sigplan Notices*, *46*(9), pp.279-291.

Gelberger, A., Yemini, N. and Giladi, R., 2013, August. Performance analysis of software-defined networking (SDN). In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (pp. 389-393).* IEEE.

Generation and measurement. INFOCOM 2007 Demo Session, 2010.

Ghasempour, A., 2019. Internet of Things in Smart Grid: Architecture, Applications, Services, Key Technologies, and Challenges. *Inventions, 4(1),* p.22.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D. and Maglaris, V., 2014. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, *62*, pp.122-136.

Goransson, P., Black, C. and Culver, T., 2016. *Software defined networks: a comprehensive approach*. Morgan Kaufmann.

Greenfield, T. and Greener, S. eds., 2016. *Research methods for postgraduates*. John Wiley & Sons.

Gudipati, A., Li, L.E. and Katti, S., 2014, August. Radiovisor: A slicing plane for radio access networks. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 237-238).

Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S. and Chao, H.J., 2014. Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks, 68,* pp.95-109.

Hagag, S. and El-Sayed, A., 2012. Enhanced TCP westwood congestion avoidance mechanism (TCP WestwoodNew). *International Journal of Computer Applications, 45(5),* pp.21-29.

Haiyan, M., Jinyao, Y.A.N., Georgopoulos, P. and Plattner, B., 2016. Towards SDN based queuing delay estimation. *China Communications, 13(3),* pp.27-36.

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D.C. and Gayraud, T., 2014. Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, *75*, pp.453-471.

Haldar, K., and Agrawal, D., 2014. QoS Issues In Openflow/SDN. In: F., HU, ed. 2014.

Haleplidis, E., Salim, J.H., Denazis, S. and Koufopavlou, O., 2015. Towards a network abstraction model for SDN. *Journal of Network and Systems Management*, *23*(2), pp.309-327.

Han, B., Gopalakrishnan, V., Ji, L. and Lee, S., 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine, 53(2),* pp.90-97.

Harkal, V.B. and Deshmukh, A.A., 2016. Software Defined Networking with Floodlight Controller. *International Journal of Computer Applications*, *975*, pp.23-27.

Hassas Yeganeh, S. and Ganjali, Y., 2012, August. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 19-24).

He, M., Alba, A.M., Mansour, E. and Kellerer, W., 2019, May. Evaluating the Control and Management Traffic in OpenStack Cloud with SDN. In *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)* (pp. 1-6). IEEE.

He, Q. and Wang, S., 2017. A low-cost measurement framework in software defined networks. *International Journal of Communications, Network and System Sciences, 10(5),* pp.54-66.

Heller, B., Sherwood, R. and McKeown, N., 2012. The controller placement problem. *ACM SIGCOMM Computer Communication Review, 42(4),* pp.473-478.

Hill, R., Hirsch, L., Lake, P. and Moshiri, S., 2012. *Guide to cloud computing: principles and practice*. Springer Science & Business Media.

Honda, M., Huici, F., Lettieri, G. and Rizzo, L., 2015, June. mSwitch: a highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (pp. 1-13).

Hoske, M.T., 2015. Industry 4.0 and Internet of Things tools help streamline factory automation. *Control Engineering, 62(2),* pp.M7-M10.

Hu, F., Hao, Q. and Bao, K., 2014. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, *16*(4), pp.2181-2206.

Huang, H.M., Tidwell, T., Gill, C., Lu, C., Gao, X. and Dyke, S., 2010, April. Cyber-physical systems for real-time hybrid structural testing: a case study. In *Proceedings of the 1st ACM/IEEE international conference on cyber-physical systems* (pp. 69-78).

Huang, J., Xu, L., Zeng, M., Xing, C.C., Duan, Q. and Yan, Y., 2015, June. Hybrid scheduling for quality of service guarantee in software defined networks to support multimedia cloud services. In *2015 IEEE International Conference on Services Computing* (pp. 788-792). IEEE.

Huang, S., Cuadrado, F. and Uhlig, S., 2017, June. Middleboxes in the Internet: a HTTP perspective. In *2017 Network Traffic Measurement and Analysis Conference (TMA)* (pp. 1-9). IEEE.

Hubert, B., Graf, T., Maxwell, G., van Mook, R., van Oosterhout, M., Schroeder, P., Spaans, J. and Larroy, P., 2002, June. Linux advanced routing & traffic control. In *Ottawa Linux Symposium (Vol. 213). sn.*

Hung, L.H., Kristiyanto, D., Lee, S.B. and Yeung, K.Y., 2016. GUIdock: using Docker containers with a common graphics user interface to address the reproducibility of research. *PloS one, 11(4)*.

Index, C.V.N., 2017. *Forecast and methodology, 2016–2021*. White Paper, June.

IRIS: The Recursive SDN-Openflow Controller by ETRI 2016. Available from: http:// openiris.etri.re.kr/.

Ishimori, A., Farias, F., Cerqueira, E. and Abelém, A., 2013, October. Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking. In *2013 Second European Workshop on Software Defined Networks* (pp. 81-86). IEEE.

Jafarian, J.H., Al-Shaer, E. and Duan, Q., 2012, August. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 127-132).

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M. and Zolla, J., 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, *43*(4), pp.3-14.

Jammal, M., Singh, T., Shami, A., Asal, R. and Li, Y., 2014. Software defined networking: State of the art and research challenges. *Computer Networks*, *72*, pp.74-98.

Jarraya, Y., Madi, T. and Debbabi, M., 2014. A survey and a layered taxonomy of software-defined networking. *IEEE communications surveys & tutorials*, *16*(4), pp.1955-1980.

Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P. and Kellerer, W., 2014. Interfaces, attributes, and use cases: A compass for SDN. *IEEE Communications Magazine*, *52*(6), pp.210-217.

Jimson, E.R., Nisar, K. and bin Ahmad Hijazi, M.H., 2017, November. Bandwidth management using software defined network and comparison of the throughput performance

with traditional network. In *2017 International Conference on Computer and Drone Applications (IConDA)* (pp. 71-76). IEEE.

Jin, X., Gossels, J., Rexford, J. and Walker, D., 2015. Covisor: A compositional hypervisor for software-defined networks. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (pp. 87-101).

Kaljic, E., Maric, A., Njemcevic, P. and Hadzialic, M., 2019. A survey on data plane flexibility and programmability in software-defined networking. *IEEE Access*, *7*, pp.47804-47840.

Karakus, M. and Durresi, A., 2017. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, *112*, pp.279-293.

Karakus, M. and Durresi, A., 2017. Quality of service (QoS) in software defined networking (SDN): A survey. *Journal of Network and Computer Applications, 80,* pp.200-218.

Kartashevskiy, V. and Buranova, M., 2018, October. Analysis of Packet Jitter in Multiservice Network. In *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)* (pp. 797-802). IEEE.

Khuri, A.I., 2004. Applications of Dirac's delta function in statistics. *International Journal of Mathematical Education in Science and Technology, 35(2),* pp.185-195.

Kim, H. and Feamster, N., 2013. Improving network management with software defined networking. *IEEE Communications Magazine*, *51*(2), pp.114-119.

Kim, H., Schlansker, M., Santos, J.R., Tourrilhes, J., Turner, Y. and Feamster, N., 2012, October. Coronet: Fault tolerance for software defined networks. In *2012 20th IEEE international conference on network protocols (ICNP)* (pp. 1-2). IEEE.

Kleinrouweler, J.W., Cabrero, S. and Cesar, P., 2016, May. Delivering stable high-quality video: An SDN architecture with DASH assisting network elements. In *Proceedings of the 7th International Conference on Multimedia Systems* (pp. 1-10).

Koeze, E. and Popper, N., 2020. *The virus changed the way we internet*. The New York Times.

Kolahi, S.S., Narayan, S., Nguyen, D.D. and Sunarto, Y., 2011, March. Performance monitoring of various network traffic generators. In *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on* (pp. 501-506). IEEE.

Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2014. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*.

Kreutz, D., Ramos, F.M.V., Verissimo, P., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2015. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, *103*(1), pp.14-76.

Kulhari, A. and Pandey, A.C., 2016, February. Traffic shaping at differentiated services enabled edge router using adaptive packet allocation to router input queue. In *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)* (pp. 242-247). IEEE.

Kuźniar, M., Perešíni, P. and Kostić, D., 2015, March. What you need to know about SDN flow tables. In *International Conference on Passive and Active Network Measurement* (pp. 347-359). Springer, Cham.

Lantz, B., Heller, B. and McKeown, N., 2010, October. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (pp. 1-6).

Lara, A., Kolasani, A. and Ramamurthy, B., 2013. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, *16*(1), pp.493-512.

Latif, Z., Sharif, K., Li, F., Karim, M.M. and Wang, Y., 2019. A Comprehensive Survey of Interface Protocols for Software Defined Networks. *arXiv preprint arXiv:1902.07913*

Lee, I. and Lee, K., 2015. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons, 58(4),* pp.431-440.

Levin, D., Wundsam, A., Heller, B., Handigol, N. and Feldmann, A., 2012, August. Logically centralized? State distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 1-6).

Lezhepekov, A., Letenko, I. and Vladyko, A., 2017, February. Software-defined routing in convergent LTE/WiFi networks. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (pp. 173-175). IEEE.

Lin, P., Bi, J. and Hu, H., 2016. BTSDN: BGP-based transition for the existing networks to SDN. *Wireless Personal Communications*, *86*(4), pp.1829-1843.

Lin, P., Bi, J., Wolff, S., Wang, Y., Xu, A., Chen, Z., Hu, H. and Lin, Y., 2015. A west-east bridge based SDN inter-domain testbed. *IEEE Communications Magazine*, *53*(2), pp.190-197.

Lippis III, N., 2007. Network Virtualization: The new building blocks of network design. *White Paper*.

Liu, Y., Li, Y., Wang, Y. and Yuan, J., 2015. Optimal scheduling for multi-flow update in Software-Defined Networks. *Journal of Network and Computer Applications, 54*, pp.11-19.

Luong, D.H., Outtagarts, A. and Hebbar, A., 2016, June. Traffic monitoring in software defined networks using opendaylight controller. In *International Conference on Mobile, Secure, and Programmable Networking* (pp. 38-48). Springer, Cham.

MarketsandMarkets Research Private Ltd, 2019. Software Defined Networking Market by SDN Type (Open SDN, SDN via Overlay, and SDN via API), Component (Solutions and Services), End User (Data Centers, Service Providers, and Enterprises), and Region - Global

Forecast to 2023. [Online] India. Available at: <https://www.marketsandmarkets.com/Market-Reports/software-defined-networking-sdn-market-655.html> [Accessed 5 August 2018].

Marschke, D., Doyle, J. and Moyer, P., 2015. *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I* (Vol. 1). Lulu. com.

Martinez, C., Ferro, R. and Ruiz, W., 2015, October. Next generation networks under the SDN and OpenFlow protocol architecture. In *2015 Workshop on Engineering Applications-International Congress on Engineering (WEA)* (pp. 1-7). IEEE.

Masoudi, R. and Ghaffari, A., 2016. Software defined networks: A survey. *Journal of Network and computer Applications*, *67*, pp.1-25.

Maugendre, M., 2015. *Development of a performance measurement tool for SDN* (Master's thesis, Universitat Politècnica de Catalunya).

McCusker, K. and Gunaydin, S., 2015. Research using qualitative, quantitative or mixed methods and choice based on the research. *Perfusion, 30(7),* pp.537-542.

McKenney, P., 1990, January. Stochastic fairness queueing. In *IEEE INFOCOM'90* (pp. 733-734).

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J., 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, *38*(2), pp.69-74.

Medved, J., Varga, R., Tkacik, A. and Gray, K., 2014, June. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (pp. 1-6). IEEE.

Mehdi, S.A., Khalid, J. and Khayam, S.A., 2011, September. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection* (pp. 161-180). Springer, Berlin, Heidelberg.

Mell, P. and Grance, T., 2011. *The NIST definition of cloud computing*.

Mendiola, A., Astorga, J., Jacob, E. and Higuero, M., 2016. A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys & Tutorials, 19(2),* pp.918-953.

Metzler, J. and Metzler, A., 2013. *Ten Things to Look for in an SDN Controller* (p. 11). Technical Report, May.

Mirchev, A., 2015. Survey of Concepts for QoS improvements via SDN. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 33(1).*

Mirkovic, D., Armitage, G. and Branch, P., 2018. A survey of round trip time prediction systems. *IEEE Communications Surveys & Tutorials, 20(3),* pp.1758-1776.

Mishra, C., Mohanty, L., Rath, S., Patnaik, R. and Pradhan, R., 2019. Application of Backward Elimination in Multiple Linear Regression Model for Prediction of Stock Index. In *Intelligent and Cloud Computing (pp. 543-551).* Springer, Singapore.

Mishra, P., Singh, U., Pandey, C.M., Mishra, P. and Pandey, G., 2019. Application of student's t-test, analysis of variance, and covariance. *Annals of Cardiac Anaesthesia, 22(4),* p.407.

MobaXterm: Enhanced terminal for Windows with X11 server, tabbed SSH client, network tools and much more. Available at : http://mobaxterm.mobatek.net/.

Mohan, B.R. and Damodaran, D.K., 2012. *Performance measuring and comparison of VirtualBox and VMware*.

Mustafa, M.E.G. and Talab, S.A., 2016. The effect of queuing mechanisms first in first out (FIFO), priority queuing (PQ) and weighted fair queuing (WFQ) on network's routers and applications. *Wireless Sensor Network,* 8(05), p.77.

Nadeau, T. and Pan, P., 2012. *Software-Defined Network (SDN) Problem Statement and Use Cases for Data Center Applications.*

Nadeau, T.D. and Gray, K., 2013. *SDN: Software Defined Networks: an authoritative review of network programmability technologies*. " O'Reilly Media, Inc.".

Network Innovation through OpenFlow and SDN: Principles and design. CRC Press. Ch.11. NICIRA, It's Time to Virtualize the Network, 2012, Available at: <http://www.netfos.com.tw/PDF/Nicira/It%20is%20Time%20To%20Virtualize%20the%20Network%20White%20Paper.pdf>. [Accessed: 16 March 2017]

Numan, P.E., Yusof, K.M., Marsono, M.N.B., Yusof, S.K.S., Fauzi, M.H.B.M., Nathaniel, S., Onwuka, E.N. and Baharudin, M.A.B., 2019. On the latency and jitter evaluation of software defined networks. *Bulletin of Electrical Engineering and Informatics, 8(4),* pp.1507-1516.

Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K. and Turletti, T., 2014. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials, 16(3)*, pp.1617-1634.

ONF, 2009. Openflow switch specification. [pdf] Available at: <https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf>

ONF, 2012. Software-Defined Networking: the new norm for networks. [pdf] Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf> [Accessed 29 June 2016].

ONF, 2012. Version 1.3.0 (Wire Protocol 0x04). [Online]. Available at: <https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf> [Accessed: 03 April 2015]

ONF, 2014. SDN Architecture overview.[pdf] Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf>. [Accessed 21 July 2015].

ONF, 2015. Version 1.5 (Wire Protocol 0x04). [Online]. Available at: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>. [Accessed: 09 March 2016]

ONF, 2016. ONF SDN evolution. [pdf] Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-535_ONF_SDN_Evolution.pdf>. [Accessed 14 January 2017].

Open Networking Foundation (ONF), 2015. [Online]. Available: <https://www.opennetworking.org/>. [Accessed: 09 March 2016]

OpenvSwitch. [Online]. Available at: <htttp://openvswitch.org>. [Accessed: 11 May 2016]

Parniewicz, D., Doriguzzi Corin, R., Ogrodowczyk, L., Rashidi Fard, M., Matias, J., Gerola, M., Fuentes, V., Toseef, U., Zaalouk, A., Belter, B. and Jacob, E., 2014, August. Design and implementation of an OpenFlow hardware abstraction layer. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing* (pp. 71-76).

Pascual Serrano, D., Vera Pasamontes, G. and Girón Moreno, R., 2016. Modelos animales de dolor neuropático. Dolor. *Investigación Clínica & Terapéutica, 31(2),* pp.70-76.

Peterson, L.L. and Davie, B.S., 2007. Computer networks: a systems approach. Elsevier.
Pfaff, B. and Davie, B., 2013. The open vswitch database management protocol. IETF RFC 7047.

Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P. and Amidon, K., 2015. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (pp. 117-130).

Phaal, P. and Lavine, M., 2004. sFlow Specification version 5.

Phemius, K. and Bouet, M., 2013, October. Monitoring latency with openflow. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)* (pp. 122-125). IEEE.

Puente Fernández, J.A., García Villalba, L.J. and Kim, T.H., 2018. Clustering and flow conservation monitoring tool for software defined networks. *Sensors, 18(4),* p.1079.

Pujolle, G., 2015. SDN (Software-Defined Networking). *Software Networks: Virtualization, SDN, 5G and Security*, *1*, pp.15-48.

Racherla, S., Cain, D., Irwin, S., Ljungstrøm, P., Patil, P. and Tarenzio, A.M., 2014. *Implementing ibm software defined network for virtual environments*. IBM Redbooks.

Raghavendra, R., Lobo, J. and Lee, K.W., 2012, August. Dynamic graph query primitives for sdn-based cloudnetwork management. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 97-102).

Rahman, M.Z.A., Yaakob, N., Amir, A., Ahmad, R.B., Yoon, S.K. and Abd Halim, A.H., 2017. Performance analysis of congestion control mechanism in software defined network (SDN). In *MATEC Web of Conferences (Vol. 140*, p. 01033). EDP Sciences.

Raju, V.S., 2018. SDN CONTROLLERS COMPARISON. In *Proceedings of Science Globe International Conference*. [Accessed: 20 August 2019]

Ranjbar, A., Komu, M., Salmela, P. and Aura, T., 2016, April. An SDN-based approach to enhance the end-to-end security: SSL/TLS case study. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (pp. 281-288). IEEE.

Rao, S.K., 2014. SDN and its use-cases-NV and NFV. *Network*, *2*, p.H6.

Raza, M.H., Sivakumar, S.C., Nafarieh, A. and Robertson, B., 2014. A comparison of software defined network (SDN) implementation strategies. *Procedia Computer Science, 32,* pp.1050-1055.

Reitblatt, M., Foster, N., Rexford, J. and Walker, D., 2011, November. Consistent updates for software-defined networks: Change you can believe in!. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (p. 7). ACM.

Ren, T. and Xu, Y., 2014, March. Analysis of the new features of OpenFlow 1.4. In *2nd International Conference on Information, Electronics and Computer*. Atlantis Press
Rizzo, L., 2012. Netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)* (pp. 101-112).

Rothenberg, C.E., Vidal, A.L.L.A.N. and Salvador, M., 2014. Hybrid networking towards a software defined era. In *Network Innovation through OpenFlow and SDN: Principles and Design* (pp. 153-198). CRC Press.

Rotsos, C., Mortier, R., Madhavapeddy, A., Singh, B. and Moore, A.W., 2012, June. Cost, performance & flexibility in openflow: Pick three. In *2012 IEEE International Conference on Communications (ICC)* (pp. 6601-6605). IEEE.

Rowshanrad, S., Namvarasl, S. and Keshtgari, M., 2017. A queue monitoring system in openflow software defined networks. *Journal of Telecommunications and Information Technology.*

Rowshanrad, S., Namvarasl, S., Abdi, V., Hajizadeh, M. and Keshtgary, M., 2014. A survey on SDN, the future of networking. *Journal of Advanced Computer Science & Technology*, *3*(2), pp.232-248.

Ruiz, M., Coltraro, F. and Velasco, L., 2018. CURSA-SQ: a methodology for service-centric traffic flow analysis. *Journal of Optical Communications and Networking, 10(9),* pp.773-784.
Rutherford, A., 2001. Introducing ANOVA and ANCOVA: a GLM approach. Sage.

Sahay, R., Meng, W. and Jensen, C.D., 2019. The application of Software Defined Networking on securing computer networks: A survey. *Journal of Network and Computer Applications, 131*, pp.89-108.

Salman, O., Elhajj, I.H., Kayssi, A. and Chehab, A., 2016, April. SDN controllers: A comparative study. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)* (pp. 1-6). IEEE.

Salvadori, E., Corin, R.D., Broglio, A. and Gerola, M., 2011, December. Generalizing virtual network topologies in OpenFlow-based networks. In *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011* (pp. 1-6). IEEE.

Sanaei, M. and Mostafavi, S., 2019, April. Multimedia Delivery Techniques over Software-Defined Networks: A Survey. In *2019 5th International Conference on Web Research (ICWR)* (pp. 105-110). IEEE.

Scott-Hayward, S., O'Callaghan, G. and Sezer, S., 2013, November. SDN security: A survey. In *2013 IEEE SDN For Future Networks and Services (SDN4FNS)* (pp. 1-7). IEEE.

Selmchenko, M., Beshley, M., Panchenko, O. and Klymash, M., 2016, February. Development of monitoring system for end-to-end packet delay measurement in software-defined networks. In *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)* (pp. 667-670). IEEE.

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

Shah, T., Yavari, A., Mitra, K., Saguna, S., Jayaraman, P.P., Rabhi, F. and Ranjan, R., 2016. Remote health care cyber-physical system: quality of service (QoS) challenges and opportunities. *IET Cyber-Physical Systems: Theory & Applications*, *1*(1), pp.40-48.

Shankar, L. and Ambe, S., Broadcom Corp, 2003. Weighted fair queuing (WFQ) shaper. U.S. Patent Application 10/351,520.

Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.Y., Kazemian, P., Kobayashi, M., Naous, J. and Seetharaman, S., 2010. Carving Research Slices

Out of your production networks with OpenFlow. *ACM SIGCOMM Computer Communication Review, 40(1)*, pp.129-130.

Sherwood, R., Gibb, G., Yap, K.K., Appenzeller, G., Casado, M., McKeown, N. and Parulkar, G.M., 2010, October. Can the production network be the testbed?. *In OSDI (Vol. 10*, pp. 1-6).

Sherwood, R., Gibb, G., Yap, K.K., Appenzeller, G., Casado, M., McKeown, N. and Parulkar, G., 2009. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, *1*, p.132.

Shirali-Shahreza, S. and Ganjali, Y., 2013, August. Efficient implementation of security applications in openflow controller with flexam. In *2013 IEEE 21st annual symposium on high-performance interconnects* (pp. 49-54). IEEE.

Shu, Z., Wan, J., Lin, J., Wang, S., Li, D., Rho, S. and Yang, C., 2016. Traffic engineering in software-defined networking: Measurement and management. IEEE access, 4, pp.3246-3256.

Singh, M., Varyani, N., Singh, J. and Haribabu, K., 2017, August. Estimation of end-to-end available bandwidth and link capacity in sdn. In International Conference on Ubiquitous Communications and Network Computing (pp. 130-141). Springer, Cham.

Singh, S. and Jha, R.K., 2017. A survey on software defined networking: Architecture for next generation network. *Journal of Network and Systems Management*, *25*(2), pp.321-374.

Sinha, D., Haribabu, K. and Balasubramaniam, S., 2015, December. Real-time monitoring of network latency in software defined networks. In 2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS) (pp. 1-3). IEEE.

Sllame, A.M. and Aljafari, M., 2015. Performance Evaluation of Multimedia over IP/MPLS Networks. International Journal of Computer Theory and Engineering, 7(4), p.283.

Song, H., 2013, August. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 127-132).

Sonkoly, B., Gulyas, A., Nemeth, F., Czentye, J., Kurucz, K., Novak, B. and Vaszkun, G., 2012, October. On qos support to ofelia and openflow. In *2012 European Workshop on Software Defined Networking* (pp. 109-113). IEEE.

Sonkoly, B., Gulyás, A., Németh, F., Czentye, J., Kurucz, K., Novák, B. and Vaszkun, G., 2012, October. OpenFlow virtualization framework with advanced capabilities. In *2012 European Workshop on Software Defined Networking* (pp. 18-23). IEEE.

Specification, O.S., 2009. Version 1.0. 0 (Wire Protocol 0x01). *Open Networking Foundation*.

Specification, O.S., 2011. Version 1.1. 0 Implemented (Wire Protocol 0x02). Feb, 28, pp.1-56. Available at: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf>. [Accessed: 03 June 2017]

Stallings, W., 2015. *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional

Stiliadis, D. and Varma, A., 1998. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on networking, 6(5),* pp.611-624.

Suñé, M., Alvarez, V., Jungel, T., Toseef, U. and Pentikousis, K., 2014, September. An OpenFlow implementation for network processors. In *2014 Third European Workshop on Software Defined Networks* (pp. 123-124). IEEE.

Sutter, J.M. and Kalivas, J.H., 1993. Comparison of forward selection, backward elimination, and generalized simulated annealing for variable selection. *Microchemical journal, 47(1-2),* pp.60-66.

Szilágyi, S. and Almási, B., 2012. A review of congestion management algorithms on cisco routers. *Journal of Computer Science and Control Systems, 5(1),* p.103.

Tanenbaum, A., and Wetherall, D., 2011. *Computer Networks.5th ed*. Pearson Education: United States of America.

Tavakoli, A., Casado, M., Koponen, T. and Shenker, S., 2009, October. Applying NOX to the Datacenter. In *HotNets*.

Taylor, M., 2014. A guide to NFV and SDN. White paper. *Metaswitch Netw., London, UK, Dec*. [Online]. Available at: <http://www.metaswitch.com/sites/default/files/Metaswitch_WhitePaper_NFVSDN_final_rs.pdf>. [Accessed: 09 December 2015]

Thakur, S. and Shukla, A., 2013. A Survey on Quality of Service and Congestion Control. International Journal of Computer Science & Engineering Technology (IJCSET).

The VMware NSX network virtualization platform. Technical white paper, VMware, Palo Alto, CA, USA, 2015. [Online]. Available at: <https://www.vmware.com/files/pdf/products/nsx/VMware-NSXNetwork-Virtualization-Platform-WP.pdf>. [Accessed: 17 September 2017]

Tootoonchian, A. and Ganjali, Y., 2010, April. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (Vol. 3).

Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M. and Sherwood, R., 2012. On Controller Performance in Software-Defined Networks. In *Presented as part of the 2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*.

Tseng, C.W., Tseng, F.H., Yang, Y.T., Liu, C.C. and Chou, L.D., 2018. Task Scheduling for Edge Computing with Agile VNFs On-Demand Service Model toward 5G and Beyond. *Wireless Communications and Mobile Computing*, *2018*.

Van Adrichem, N.L., Doerr, C. and Kuipers, F.A., 2014, May. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS*) (pp. 1-8). IEEE.

Van der Merwe, J.E., Rooney, S., Leslie, I. and Crosby, S., 1998. The Tempest-a practical framework for network programmability. *Network, IEEE, 12(3),* pp.20-28.

Wallner, R. and Cannistra, R., 2013. An SDN approach: quality of service using big switch's floodlight open-source controller. In *Proceedings of the Asia-Pacific Advanced Network, 35(14-19),* pp.10-7125.

Wang, G., Ng, T.E. and Shaikh, A., 2012, August. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 103-108).

Wang, J.M., Wang, Y., Dai, X. and Bensaou, B., 2015. SDN-based multi-class QoS guarantee in inter-data center communications. *IEEE Transactions on Cloud Computing, 7(1),* pp.116-128.

Wang, S., Li, D. and Xia, S., 2015, April. The problems and solutions of network update in SDN: A survey. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 474-479). IEEE.

Wang, W., Tian, Y., Gong, X., Qi, Q. and Hu, Y., 2015. Software defined autonomic QoS model for future Internet. *Journal of Systems and Software, 110,* pp.122-135.

Wang, Z., 2001. *Internet QoS: architectures and mechanisms for quality of service.* Morgan Kaufmann.

Wen, H., Tiwary, P.K. and Le-Ngoc, T., 2013. Network Virtualization: Overview. In *Wireless Virtualization* (pp. 5-10). Springer, Cham.

White, J. and Pilbeam, A., 2010. A survey of virtualization technologies with performance testing. *arXiv preprint arXiv:1010.3233.*

Wu, D., Hou, Y.T., Zhu, W., Zhang, Y.Q. and Peha, J.M., 2001. Streaming video over the Internet: approaches and directions. *IEEE Transactions on circuits and systems for video technology, 11(3),* pp.282-300.

Xu, C., Chen, B. and Qian, H., 2015. Quality of service guaranteed resource management dynamically in software defined network. *Journal of Communications, 10(11),* pp.843-850.

Yamanaka, H., Kawai, E. and Shimojo, S., 2017. AutoVFlow: Virtualization of large-scale wide-area OpenFlow networks. *Computer Communications*, *102*, pp.28-46.

Yazici, V., Sunay, M.O. and Ercan, A.O., 2014. Controlling a software-defined network via distributed controllers. *arXiv preprint arXiv:1401.7651*.

Yeganeh, S.H., Tootoonchian, A. and Ganjali, Y., 2013. On scalability of software-defined networking. *IEEE Communications Magazine, 51(2),* pp.136-141.

Yin, H., Xie, H., Tsou, T., Lopez, D., Aranda, P. and Sidi, R., 2012. Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains. *IETF draft, work in progress*.

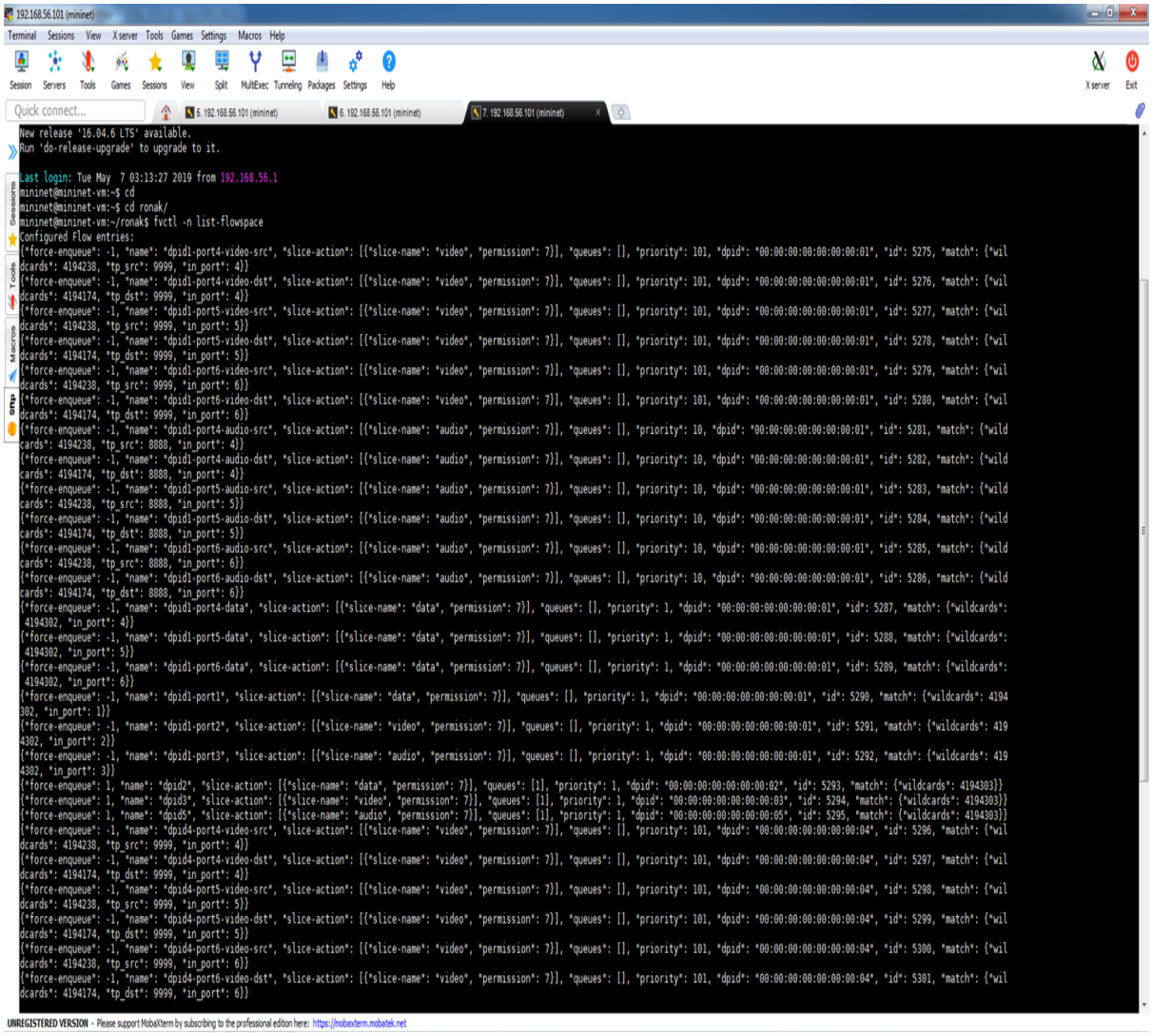
Yu, C., Lumezanu, C., Sharma, A., Xu, Q., Jiang, G. and Madhyastha, H.V., 2015, March. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement* (pp. 360-372). Springer, Cham.

Zhang, X., Hou, W., Guo, L., Zhang, Q., Guo, P. and Li, R., 2020. Joint optimization of latency monitoring and traffic scheduling in software defined heterogeneous networks. *Mobile Networks and Applications, 25(1),* pp.102-113.

Zhang, Y., Cui, L., Wang, W. and Zhang, Y., 2018. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, *103*, pp.101-118.

List of Appendices

Appendix 1. Flow space assignment for the TS algorithm



Appendix 2.A. Queuing configurations script with the LLQ algorithm for 40Mbps

Queuing configurations script with the LLQ algorithm for 40Mbps

```
#!/bin/bash
sudo ovs-vsctl -- set port s2-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s2-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s3-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s3-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s5-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s5-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s4-eth4 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=40000000
queues=1=@q1,2=@q2,3=@q3 -- \
```

```
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth5 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=40000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth6 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=40000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3
sudo ovs-vsctl list qos
```

Appendix 3.B. Queuing configurations script with the LLQ algorithm for 70Mbps

Queuing configuration script with the LLQ algorithm for 70 Mbps

```
#!/bin/bash
sudo ovs-vsctl -- set port s2-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s2-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s3-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s3-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s5-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s5-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s4-eth4 qos=@qos -- \
```

```
--id=@qos create qos type=linux-htb other-config:max-rate=70000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth5 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=70000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth6 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=70000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3
sudo ovs-vsctl list qos
```

Appendix 4.C. Queuing configurations script with the LLQ algorithm for 100Mbps

Queuing configurations script with the LLQ algorithm for 100 Mbps

```
#!/bin/bash
sudo ovs-vsctl -- set port s2-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s2-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=0

sudo ovs-vsctl -- set port s3-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s3-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=46

sudo ovs-vsctl -- set port s5-eth1 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s5-eth2 qos=@qos -- \
--id=@qos create qos type=linux-htb queues=1=@q1 -- \
--id=@q1 create queue dscp=18

sudo ovs-vsctl -- set port s4-eth4 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=100000000
queues=1=@q1,2=@q2,3=@q3 -- \
```

```
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth5 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=100000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3

sudo ovs-vsctl -- set port s4-eth6 qos=@qos -- \
--id=@qos create qos type=linux-htb other-config:max-rate=100000000
queues=1=@q1,2=@q2,3=@q3 -- \
--id=@q1 create queue other-config:priority=1 -- \
--id=@q2 create queue other-config:priority=2 -- \
--id=@q3 create queue other-config:priority=3
sudo ovs-vsctl list qos
```

Ronak Al-Haddad

United Kingdom

ARU