



UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

**SERVICE AWARE SCHEDULING POLICY FOR TIME CONSTRAINED
COMMUNICATION IN NETWORK FUNCTIONS VIRTUALIZATION AND
SOFTWARE DEFINED NETWORKS**

BY:

MUCHOKI SIMON MWANGI

P53/79176/2015

SUPERVISORS:

PROF. WILLIAM OKELO-ODONGO, DR. EVANS A. MIRITI

NOVEMBER, 2017

**Research project carried out in partial fulfillment of the requirements for the award of
Master of Science Degree in Distributed Computing Technology**

DECLARATION

This research project is original work and to the best of my knowledge has not been presented in any university or college for award of a degree.

SIGNED: _____ DATE: _____

MUCHOKI SIMON MWANGI

REG. No: P53/79176/2015

This research project has been submitted as part of fulfillment of the requirements for the award of Masters of Science degree in Distributed Computing Technology at the School of Computing and Informatics, University of Nairobi, with my approval as a university supervisor

SIGNED: _____ DATE: _____

DR EVANS A. MIRITI

SCHOOL OF COMPUTING AND INFORMATICS

UNIVERSITY OF NAIROBI

DEDICATION

This project is dedicated to my wife Jane and the entire family for great moral support and encouragements when carrying out this work. God bless you!

ACKNOWLEDGEMENT

I sincerely acknowledge the contribution of my project supervisor Prof William Okelo Odongo and Dr. Evans A. Miriti for guidance in carrying out this work, I am also grateful to the other members of the academic panel - Dr. Christopher Chepken and Dr. Mburu, your great advice and positive criticism enabled me to make continuous improvements. Thank you!

Finally I thank God for everything during this academic Journey.

ABSTRACT

Communication services have varying quality of service (QoS) requirements which the network should provide for a service to meet desired objective or user expectations. When it comes to services with strict service levels requirements and timing constraints, the need to provide quality of service management and control becomes even more important. Software defined networks (SDN) and network functions virtualization (NFV) are modern approaches in network design that offers the environment in which this control can be realized and with flexibility, worth noting is that traditionally network quality has been measured by the degree of end user satisfaction. To ensure that the users' expectations were met, a lot of effort and resources were put into the design and development of network appliances, this was to ensure resiliency and reliability. With the advent of network functions virtualization, the network appliances are implemented in software run on commercial off the shelf servers, further in software defined network, the architecture is transformed with the control being separated from forwarding and the network becoming open and programmable. Even with this shift to new networking paradigms the end user expectations hardly change - the network must meet certain levels of quality required for various use cases. In this study we looked at the concepts surrounding software defined networks and network functions virtualization with a view to understanding some of the weaknesses and challenges experienced in realizing service quality, our study revealed that there is a relationship between network latency and throughput to topology, policy, congestion and the traffic nature. We developed an SDN prototype to represent the case of vehicle tracking system, in reality GPS (global positioning system) devices fitted in motor vehicles send data to an application server; this data is carried by a multiservice network thus prone to congestion and quality degradation. To alleviate the effect of this degradation for our GPS tracking use case, we proposed and implemented a traffic shaping and priority queuing policy to ensure that in the midst of network load occasioned by multiple services sharing network resources, the information flow from the GPS devices to application server is accorded highest priority treatment. The prototype was developed and tested in a virtual network comprising of Mininet hosts, Open vSwitch (OVS) and RYU software defined network controller. Results obtained from the experimental data showed that, despite the congestion created on the network our traffic of interest in-line with our policy was not affected. The conclusion we drew from our experiment was that the policy was adequate in providing the desired quality of service functionality for our use case.

CONTENTS

DECLARATION	i
DEDICATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT.....	iv
CONTENTS.....	v
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ABBREVIATIONS.....	ix
CHAPTER ONE: INTRODUCTION.....	1
1.1 Background	1
1.1.1 Definitions of relevant terms	2
1.2 Problem Statement	2
1.3 Purpose and Objectives of the Research	5
1.4 Significance of the Research	5
1.5 Project Scope and Limitations.....	5
CHAPTER TWO: LITERATURE REVIEW	6
2.1 Introduction	6
2.2 Background	6
2.2.1 Software defined network concept.....	6
2.2.2 OpenFlow Protocol	7
2.2.3 Network functions virtualization concept.....	7
2.2.4 Industry background survey.....	9
2.3 Quality of Service in Networks.....	10
2.3.1 Service Awareness/ Application Identification	10
2.3.2 Quality of Service Policies.....	10
2.3.3 Quality of Service (QoS) Implementation	11
2.4 Related Work.....	11
2.4.1 Network Performance Challenges	11
2.4.2 Suggested Methods of Improving Network Performance	12
2.4.3 Per Flow QoS in SDN.....	13
2.5 Literature Review Summary	14
2.5.1 Proposed Architecture.....	15
CHAPTER THREE: METHODOLOGY	16

3.1	Introduction	16
3.2	Prototype Design	16
3.2.1	Experiment Requirements.....	16
3.2.2	Experimental tools overview	16
3.2.3	Experiment Design.....	18
3.2.4	Policy Implementation	20
3.2.5	Testing and Measurements	21
CHAPTER FOUR: RESULTS, DISCUSSION AND CONCLUSION		23
4.1	Results and Discussion.....	23
4.1.1	Policy evaluation.....	26
4.2	Conclusion.....	28
4.3	Study Limitations and Suggestion for Further Work	29
APPENDIX A: Experimental Data Sheets.....		30
APPENDIX B: Evaluation Results for Varying Policy Rate.....		34
APPENDIX C: Screen Shots		36
REFERENCES		39

LIST OF FIGURES

Figure 1: GPS tracking diagram	4
Figure 2: SDN concept, source ONF whitepaper. (2012).....	7
Figure 3: NFV concept, source Aaron Gember-Jacobson et al, (2014) paper	8
Figure 4: NFV reference architecture: Source ETSI report	8
Figure 5: Proposed Architecture	15
Figure 6: OpenvSwitch	17
Figure 7: Prototype Topology	19
Figure 8: Throughput vs Time, client rate 500Kbps.....	24
Figure 9: Data transferred Vs Time, client rate 500Kbps.....	24
Figure 10: Throughput Vs Time, client rate 800Kbps	25
Figure 11: Data transferred Vs Time, client rate 800Kbps.....	26
Figure 12: Throughput Vs Time, Policy 500Kbps max /284Kbps min.....	27
Figure 13: Data transferred Vs Time, policy 500Kbps min/284Kbps max	28

LIST OF TABLES

Table 1: Sender and Receiver hosts	21
Table 2: Policy rates.....	22
Table 3: Host 8 source 500Kbps.....	30
Table 4: Host 5, Host 6 and Host 7 source 500kbps.....	31
Table 5: Host 8 source 800kbps.....	32
Table 6: Host 5, Host 6 and Host 7, source 800kbps.....	33
Table 7: Policy Rate 500kbps Min 284Kbps Max, Host 8 Port 5001, Port 5002, Port 5003	34
Table 8: Policy Rate 500Kbps Min 284 Max, Host 5, Host 6, Host 7 Ports 5001	35

LIST OF ABBREVIATIONS

NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
VNF	Virtualized Network Function
USSD	Unstructured Supplementary Services Data
BTS	Base Transceiver Station
BSC	Base Station Controller
GGSN	Gateway GPRS Support Node
SGSN	Serving GPRS Support Node
HLR	Home Location Register
MGW	Media Gateway
MSC	Mobile Switching Centre
IN	Intelligent Network
OCS	Online Charging System
MWN	Mobile Wireless Network
QOE	Quality of Experience
SMSC	Short Message Switching Centre
VM	Virtual Machine
CPU	Central Processing Unit
I/O	Input /Output
SDN	Software defined networking
REST	Representational State Transfer
QOS	Quality of service
QOE	Quality of experience

CHAPTER ONE: INTRODUCTION

1.1 Background

Communications services are provided on specific hardware and software components referred to as network appliances or network elements, these components are connected through interfaces defined in a technology standard to create a communication network.

Network element performs a specific function or more to enables realization of different types of services for the end users, examples of such services are voice, short and multimedia messaging, video, data etc. thus communication network components must be dependable to meet expectations and requirements of service users, any break in service continuity cannot therefore be tolerated as it may lead to loss of revenue and reputation or to disastrous consequences.

Hassan Hawilo et al. (2014) argue that communication network elements must provide exceptional levels of reliability and performance predictability. These two characteristics are paramount especially in the design of network elements and can be considered as what differentiates the design of communication equipment and general purpose computing equipment.

The general trend in the technology industry is that end users expectations with regard to network performance keeps changing, also new service requirements emerge as the end user becomes accustomed to more and more services. Network service providers as they strive to meet varying needs of the users must ensure that network quality do not suffer, to achieve this new technologies and networking paradigms are developed and put into use.

Software defined networking (SDN) and network functions virtualizations (NFV) have emerged as such new technologies. These two approaches are being appreciated within both the academia and the industry. There is consensus that the two will play a significant role in the deployment of next generation networks. Aaron Gember-Jacobson et al. (2015) observes that though at infancy the trend for the communications industry is NFV and SDN.

1.1.1 Definitions of relevant terms

Services

In the context of this research project, service refers to applications provided to users or to end points (in-case of machine to machine scenario) of a communication network. Generally this definition applies to data and multimedia traffic that requires transmission over a network.

Service awareness

This refers to the ability of a network to distinguish between different services being carried in a network.

Software defined Network - SDN

An accurate definition was put forward by open networking foundation (ONF). According to this body responsible for the standardization of the software defined networking technology, SDN refers to a networking architecture in which control plane is separated from forwarding plane, this allows control functions to be performed by a controller referred to as an SDN controller, implemented as a software with well-defined interfaces. On the other hand the forwarding plane is implemented as devices distributed on the network with the sole role of forwarding the datagrams, the centralized controller assumes the role of determining how the datagrams are routed or handled in the network thus is referred to as the brain of the network. Further SDN-controller allows network programmability to realize desired packet and frames forwarding rules.

Network Functions Virtualizations

Researchers have written extensively on this emerging trend, according to Yoshida M. et al. (2015), in NFV network functions originally provided by physical elements or appliances are provided by virtualized entities running on general purpose commercial off the shelf servers (COTS).

1.2 Problem Statement

Networks are built for multiple use cases or in other words to carry more than one service, end users expectations for each use case with regard to performance and quality of service (QoS) are key considerations in the design of network elements and also in the integration of multiple elements. With network softwarization through the introduction of software defined networking and network functions virtualization it is imperative that the same fundamental design goals are considered.

Building a network to satisfy requirements for multiple use cases is however met by the constraints of cost and resource limitations, for example on equipment capability and capacity, therefore services which have strict service level requirements or are delay sensitive, may not meet expected quality of service (QoS) in the absence of intervening mechanisms since all services compete for the same limited network resources.

Traditionally, network quality of service have been implemented using mechanisms such as integrated services and differentiated services, each technique with its own weaknesses and strengths manifested in their mode of implementation. It is worth noting that the implementation of this traditional mechanisms is largely proprietary in nature as it is dependent on the equipment vendor specified way of implementing quality of service and performance requirements, this leaves little room for network manager's discretion on how traffic should be steered or handled on the network. One of the weakness of the traditional methods are notable lack of centralized intelligence to configure quality of service (QoS) based on changing network conditions according to Derrick Desouza et al. (2016), further the configuration process is cumbersome considering that a network may have thousands of devices. In contrast software defined networking emphasis is on open innovation such that network managers can directly manipulate network behaviour, the idea behind SDN is to introduce programmability of the network to meet desired forwarding rules and logic by the users or the information technology administrators themselves.

In real life there are many instances where service quality or timeliness is key for proper functioning of a system, this cases could range from transmission of sensors data in a factory floor or cases like location based services. With SDN approach, QoS for such cases may be guaranteed; the forwarding logic may be programmed centrally with appropriate policies applied on all network devices. This invites us to research on methods applicable in the context of software defined networks to solve the quality of service problems for such cases, in our study we looked at an example case of GPS tracking.

Description (GPS Tracking and fleet management)

In this project a case of GPS vehicle tracking and fleet management was considered. A GPS tracker is an embedded device that communicates with a GPS server; data communicated by the GPS device are position (longitude, latitude) speed and time. Also GPS tracker act as the communication hub to other sensors that may be in the vehicle, examples are fuel sensors, temperature sensors, ignition sensors, accelerometers etc. thus the device may need to send critical data whose timeliness is significant to the

GPS server residing in a private or a public cloud. Proper function of GPS tracking systems would require that commands to control some functions in the car or data originating from the GPS devices are transmitted over a network, therefore the network itself becomes a critical component whose performance and capacity must be guaranteed to be available. Worth mentioning is that this kind of services whenever they are deployed on a network may require special categorization with regard to economic significance to a service provider, this means the service provider may accord preference over other services carried on the same network.

Typical architecture of GPS systems

In all cases GPS devices contains means of communication with remote GPS server via packet data networks. For the communication to be successful the GPS devices must be configured with static public IP address of the GPS server such that the devices can reach the server from anywhere. For example to reach a GPS server a Coban model of tracker would need to be configured with IP address and port number of the GPS server using the following commands as obtained from Coban 303f manual-

adminip123456 xxx.xxx.xxx.xxx yyyy
(where xxx.xxx.xxx.xxx is the IP address of the host server and YYYY is the port number).

Typical architecture of GPS tracking is as shown in figure 1 below.

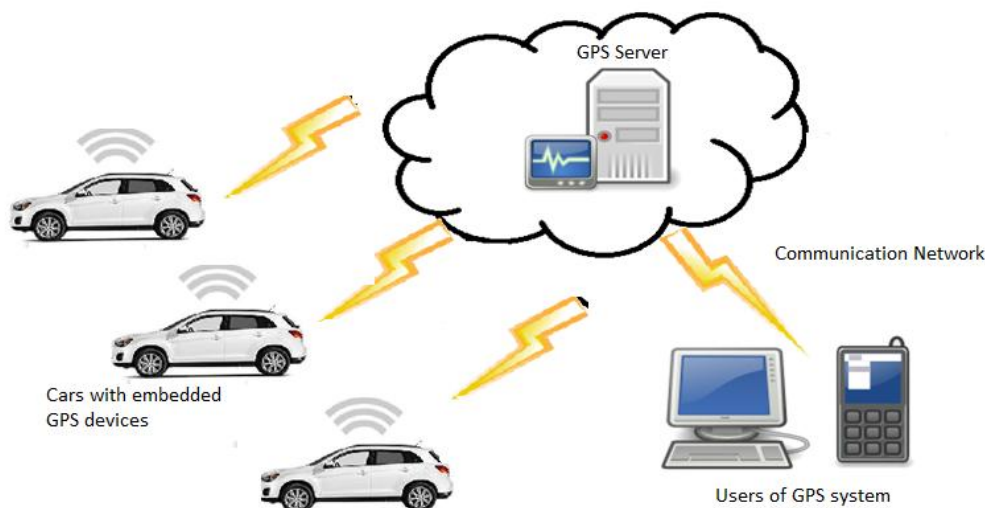


Figure 1: GPS tracking diagram

1.3 Purpose and Objectives of the Research

The main objective of this research project was to demonstrate a scheduling policy that was to assign desired throughput and minimize delay for services whose timeliness is key or service level is strict in a network with multiple services.

The specific objectives were as follows;

1. Investigate the workings and the architectures of software defined network controllers and virtualized switching network elements.
2. Develop a network prototype
3. Evaluate the effectiveness of the policy on the packets forwarding behavior of the network prototype elements with respect to traffic throughput

1.4 Significance of the Research

This research project examined approaches to QoS in networks and how this can be carried out in modern softwarized networks to realize desired service level requirements of application cases. Industry practitioners and research community who are tasked with optimization of service providers' networks to meet widely varying use cases; especially with the emergence of over the top (OTT) and internet of things (IOT) based applications that have pushed network capacities to the limits will benefit from research output that provides answers to some of the challenges they face.

In practical scenarios this research work provides suggestions to service providers seeking to meet stringent service level agreement for emerging cases of enterprise focused business model, where service providers get request for connectivity service from independent business entities i.e. third parties, e.g. fleet tracking companies, security agencies, railway companies, mobile health etc, usually this kind of third party entities demand special level of service guarantees which the operator in absence of appropriate technological capabilities will be unable to meet.

1.5 Project Scope and Limitations

There are a number of network performance metrics for example throughput, delay and delay variation /jitter, error rate etc, this research study scope focused on delay and throughput.

The limitation of this study was that software defined networking and network functions virtualization being a relatively new field suffers from inadequate research output hence literature was not in abundance.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

This chapter focuses on background survey and literature review on key concepts contained in this research proposal. Literature surrounding the concept of SDN and NFV is surveyed with a view to understanding the genesis and research output focusing on these areas. Worth noting is that there is general consensus that when it comes to new generation networks there will be need for greater flexibility in design and deployment, this is partly driven by the need to cut capital and operational expenditures of running networks and also the need to cater for the proliferation of connected things that have led to massive increase in volumes of traffic that must be carried by service providers networks.

2.2 Background

2.2.1 Software defined network concept

SDN is a networking architecture that has decoupled the network control from the forwarding functions; this is according to open networking foundation (ONF), the industry body responsible for standardization of software defined networks. This decoupling has in essence enabled the network logic to be distinct and programmable on its own. Other terms used to define SDN is -agile network, this is due to the fact that it can be dynamically and centrally adjusted to meet requirements of the networks users or administrators. The network intelligence is centralized in a software component referred to as a controller while forwarding is done by devices distributed across the network and generally referred to as forwarders. A key description by ONF is that SDN is programmatically configured hence network managers can write programs themselves and deploy them on the network with adherence to open standards. SDN controllers issues instructions to manipulate the design and the operations of a network via standard interfaces in contrast with traditional approach where such operations were hardwired in proprietary equipments and networking protocols or based on vendor specific methods. The figure 2 below shows the concept of SDN, network applications communicate via northbound applications programming interface (API) of the controller, the controller then via the southbound interface effect the desired logic on the forwarding devices spread across the network.

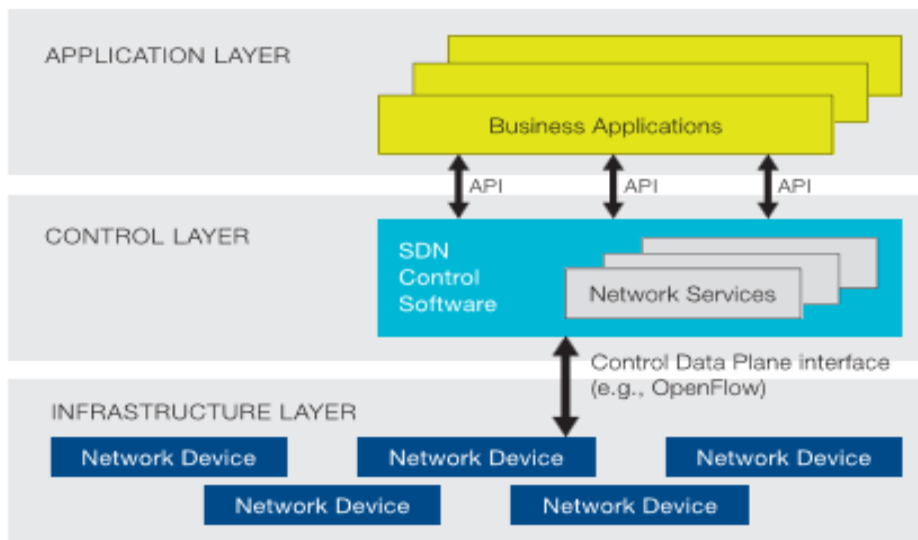


Figure 2: SDN concept, source ONF whitepaper. (2012)

2.2.2 OpenFlow Protocol

The OpenFlow was first proposed by Nick Mc Keown et al. (2008) for use in campus networks, the idea was to enable network researchers have a way of testing their ideas and innovations. OpenFlow has now become the first open standard interface to be defined between the controlling and forwarding layers of SDNs. Through open-flow interface, network devices can be directly manipulated by control layer to realize desired network behavior.

2.2.3 Network functions virtualization concept

According to Aaron Gember -Jacobson et al. (2014) there is a growing interest in replacing dedicated network hardware with softwarized ones that can be run on general purpose computing resources, a trend they described as network functions virtualization. Figure 3 below explains the concept of NFV, worth noting are traditionally known network appliances being replaced by virtualized entities running on general purpose cloud servers.

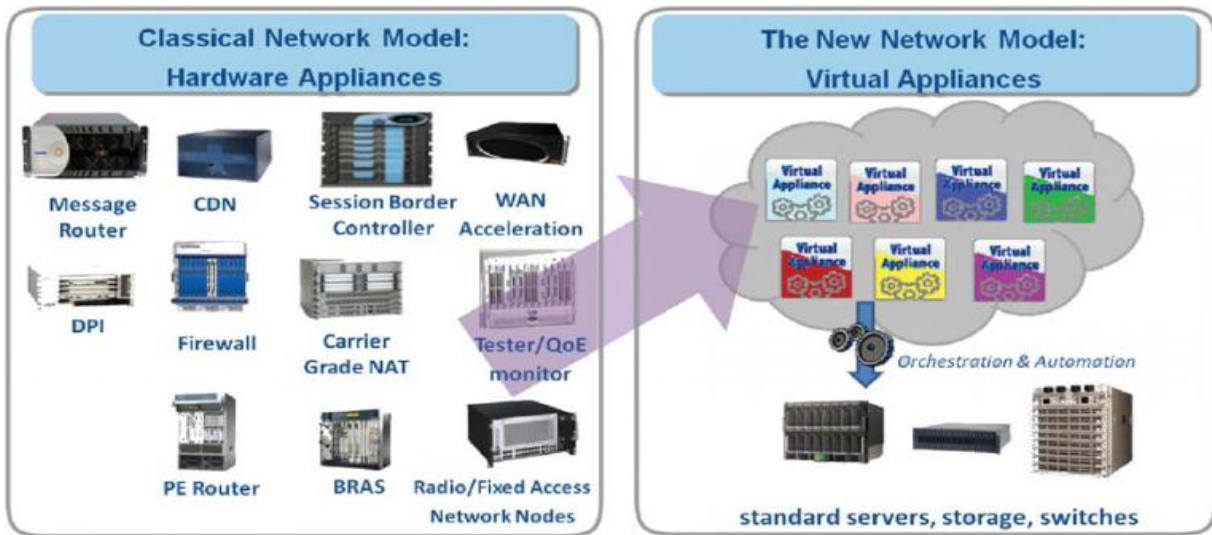


Figure 3: NFV concept, source Aaron Gember-Jacobson et al, (2014) paper

2.2.3.1 Reference architecture

The most significant effort towards standardization of carrier grade NFV was carried out by European telecommunications standards institute (ETSI), figure 4 below is the reference architecture put forward by ETSI for adoption by the industry practitioners and researchers

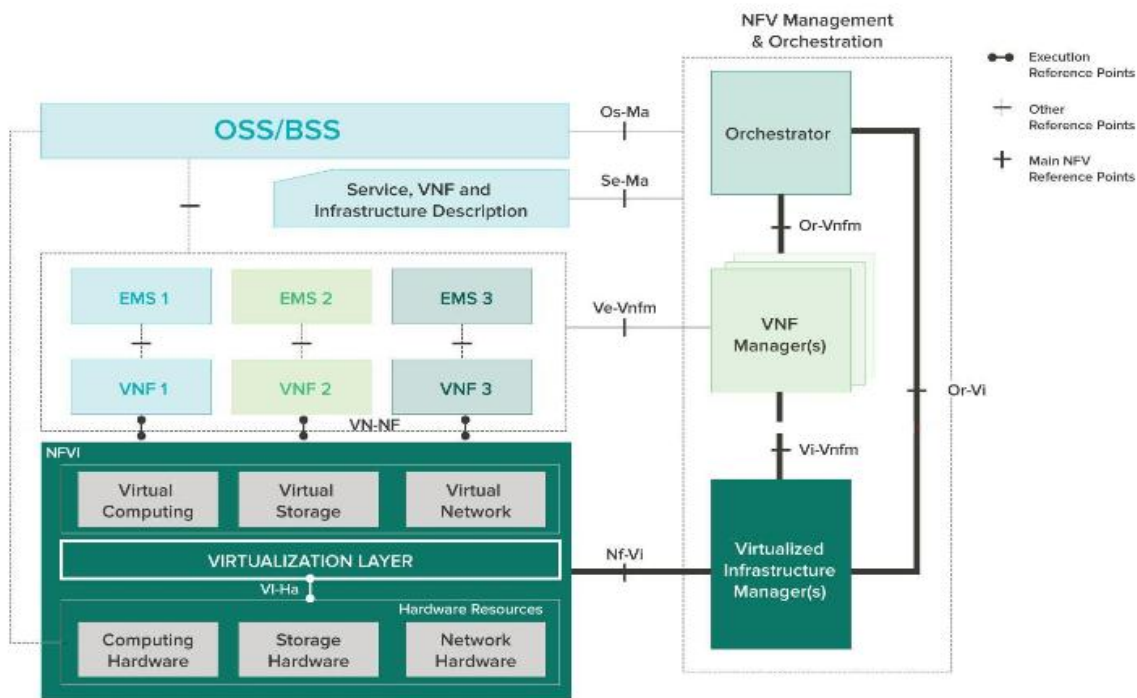


Figure 4: NFV reference architecture: Source ETSI report

According to Nathalie Omnes et al. (2015) the functionality of the components captured in the above reference architecture are as follows; the compute, storage and network infrastructure constitutes the network functions virtualization infrastructure (NFVI), which is managed and controlled by virtualized infrastructure manager (VIM). The network application that are the direct replacement for network appliances forms the next layer and are referred to as the virtual network functions (VNF) controlled and managed by VNF manager. For everything to work in harmony orchestration is performed by network functions virtualization orchestrator (NFVO).

2.2.4 Industry background survey

In our background check we came across notable initiatives supporting the shift in networking technologies, some of them are by technology vendors themselves while others are by standardization bodies. ETSI is one such organization who in their report (NFV ISG Published Documents, 2014) observes that service providers' networks have variety of vendor specific and non-standard hardware equipments which makes running a network more and more difficult, further there are energy costs and skills requirements associated with hardware-based appliances. ETSI claims that some of the challenges faced by network providers may be overcome if network virtualization comes into play. The key emphasis is that standard commodity computing resources may be utilized to run networks in points of presence or in data centers for both mobile and also for fixed networks.

Other industry perspectives points to similar approach, according to Ericsson white paper (Network functions virtualizations and software management, 2015), network functions virtualization is an architecture concept in deployment of networks with particular emphasis on virtualization technologies, through virtualization it becomes possible to create software nodes that can be connected to virtualize entire classes of network node functions into building blocks that may be connected or chained together to create an end to end network.

So far initiatives to ensure adoption of NFV and SDN in a harmonized and standardized manner are ongoing, as a result open platforms and reference architectures have been proposed, some of those we came across are open daylight which has become popular SDN controller preferred by network operators. Linux foundation in September 2014 launched an open platform for NFV (OPNFV) with standard hardware and software interfaces based on ETSI reference model discussed earlier in this

document. Another initiative mentioned earlier is the open networking foundation (ONF) that is responsible for advocating and leading the standardized approach in open-flow networks development.

2.3 *Quality of Service in Networks*

In our study we looked into quality of service and how it is implemented in networks, considering that in the context of software defined networks the basic idea is programmability of networking logic, we concentrated on how this flexibility can be exploited to realize desired quality of service levels for our use case, according to international telecommunications union (ITU-TE.800, 09/2008) quality of service takes many dimensions but have everything to do with end users perception of the service being consumed, in practical sense it deals with delay, delay variation, throughput and packet loss as important metrics that determine how users experience a certain service.

2.3.1 Service Awareness/ Application Identification

Networks are made to carry multiple services and all have varying quality of service requirements, therefore it is important that services are distinguishable so that policies may be applied accordingly. Service awareness refers to the ability of a network to distinguish between different types of traffic carried. The significance of this is to ensure that appropriate mechanism is applied to certain traffic types to ensure that end users expectations are met.

Literature mentions techniques like domain name system (DNS) lookup, others are deep packet inspection (DPI) that looks into layer 4- layer 7 to extract protocols information therefore determine what type of traffic is being carried. Other mechanism that is applied is looking into source and destination IP address and port numbers of the application.

2.3.2 Quality of Service Policies

There are various mechanisms for providing QoS in a network; this includes admission policies, traffic shaping, packet scheduling among others, in our study we looked at traffic shaping since it's of great relevance to this proposal. Basically traffic shaping is the act of controlling bandwidth to ensure that there is prioritization of critical traffic, it is implemented on core routers/switches to provide a mechanism to control the amount and volume of data being sent into the network as well as the rate at

which it is sent. Advances in traffic shaping more-so in the context of software defined network allows prioritization of traffic by IP address, host name or application

2.3.3 Quality of Service (QoS) Implementation

Quality of service policies may be implemented using differentiated services code point (DSCP) where traffic at the entry into a network domain is marked with a DSCP code, traffic is then accorded priority based on its code. Other mechanisms are per-flow QoS where traffic is categorized into flows and priority level is accorded to flows.

2.4 Related Work

2.4.1 Network Performance Challenges

As network functions virtualization and software defined concepts enters the mainstream multiple challenges have been identified ,a lot of available literature seems to focus on the aspect of carrier grade performance, in essence carrier grade performance refers to the exceptional level of reliability and availability provided by network equipments, traditionally communication network equipments are known to provide resiliency since the services they carry are largely mission critical and need to be available at all times.

Replacing the traditional appliances with softwarized appliances has presented challenges to both the academia and the industry, in our study we looked at key concern areas and what has been done by researchers. We noted that key concerns revolves around performance and carrier grade service assurance , Hassan Hawilo et al. (2014) mentions some of the concerns as security of software based networks , operation and management, interconnection of elements, performance, carrier grade service assurance and co-existence with legacy networks. Hassan Hawilo et al. (2014) argues that for services to be considered carrier grade, hardware and software components manufacturers must ensure that high availability and reliability are key factors in design. Thus to meet carrier-grade service requirements technology should be resilient to failure to guarantee service quality that is acceptable to users.

In other observations by ETSI through its base document (NFV ISG Published Documents, 2014) network functions virtualization approach to building networks is based on standard computing resources, this means that there is a possibility of performance decrease that must be given due

consideration. The authors argue that, the challenge is keeping the effects of performance decline minimal and seems to suggest that hypervisors and software technologies choice is a key decision if the effects on network performance is to be minimized. Therefore it can be argued that the choice of technologies is key in enhancing the performance of softwarized networks.

The argument by ETSI is supported by Wenyu Shen et al. (2015) who argues that the management of a network based on virtual network elements is more difficult than management of past generation networks. Wenyu shen et al. (2015) further argues that the implementation of software-based network has its own complexities with regard to management of logical links and software based components. Further observation in their paper is that virtualization leads to dramatic increase in number of managed objects since not only the physical infrastructure is managed but also the virtual infrastructure, as well as the interaction between them.

2.4.2 Suggested Methods of Improving Network Performance

In networks, it is imperative that end users quality of service and quality of experience is maintained, according to international telecommunications union (ITU) standards document (definition of terms related to quality of service ITU-T. E800, 2008) quality of service takes many dimensions, notable according to ITU is the need for the network to deal with delay, delay variation, bandwidth and packet loss. This definition of quality of service/experience must be carried along even into the era of softwarized networks.

Many researchers in the SDN and NFV domain have looked into probable mechanisms of realizing quality of service and experience. Worth noting is the advantage of network programmability that has been presented by software defined networks.

In our study we looked into some of the suggested methods, according to Mao Yang et al. (2014) it is possible to dynamically adjust network configurations through the control plane to realize desired QoE and QoS levels. Mao Yang et al. (2014) further points out the weaknesses in current traditional networks architectures whereby different service types are accorded the same network characteristics, therefore naturally the QoE and QoS for end users deteriorates, a glaring weakness in such mechanism is that it is assumed that all services carried on the network have similar requirements but in reality this is not the case. Mao Yang et al. (2014) proposes a solution that emphasizes on having different services

running on different virtual networks, further they suggest an example where a virtual network may carry voice traffic and provide low latency required in voice communication while provisioning another network that provides high data rates required by video services. This service customization according to the Mao Yang et al. (2014) may meet quality of service and enhance experience of the users; however the authors note that despite this solution there are issues to do with services middle boxes placement on the network, optimization of traffic paths and dynamic allocation of resources.

Mao Yang et al. (2014) view is supported by Navid Nikaein et al. (2015) who argues that softwarized networks can provide the tools to collapse the silo network architectures into a horizontal architecture whose elements can be combined as network functions chained together to provide a service with specific targeted performance. Navid Nikaein et al. (2015) arguments show that it is possible to assign target performance parameters with flexibility in softwarized network architectures in sharp contrast with traditional ones.

2.4.3 Per Flow QoS in SDN

In our study we looked at works related to per-flow QoS implementation mechanisms to understand some of the challenges, today due to big data and proliferation of internet of things, various types of applications impose diverse quality of service (QoS) requirements on bandwidth, latency, error rate and so on.

For time constrained communications or strict service level applications the QoS and QoE problem is more pronounced since besides transmitting information between end points, this must be done while meeting deadlines and required service levels. In the context of software defined networks the answer seems to lie in the ability to control the traffic flows to provide end to end performance guarantees.

OpenFlow protocol opens the network to programmability by providing the capability to directly manipulate the network behavior according to Liwei Quang et al. (2016), OpenFlow interface thus helps in the realization of QoS control though implementation is still dependent on northbound applications of the controllers; there also has been criticism of some implementation of openflow, one being that its performance scalability with increasing network processing load is usually not predictable. In some of the works we came across Michael Jarschel et al. (2015) in their paper come up with a performance model, a simulation and also a real experiment, to analyze the behavior of OpenFlow controller, with

respect to packets delay and packets drop under heavy load. The model used probability feedback queue theory, in this model Michael Jarschel et al. (2015) concluded that sojourn time depends on the speed of the open flow controller, the higher the rate of arrival of new flows at the switch the longer the processing time, the conclusion was that the controller performance is thus important consideration in the configuration of new flows. In their augments, today controllers' implementations are not able to handle huge number of flows which have been occasioned by high speed links of 10G and above. The weakness of some openFlow controllers is thus apparent from this author's point of view.

Despite the NFV&SDN providing ability to provision a flexible and cost effective network Marcelo Caggiani Luizelli et al. (2015 argues that there are still underlying problems, in particular and of relevance to this study is the problem of placement of network functions as it has a direct bearing on processing time and latencies.

2.5 Literature Review Summary

In our study we looked at the concepts of softwarized networks and the various challenges encountered in the shift to this new networking paradigms, many authors agree that the future networks will be virtualized and will be directly programmable, the authors also points out that issues around networks performance and quality of service must be addressed in the new networking architectures just like it has been done in the past, this is in order to meet the fundamental design goals of resiliency and reliability of communication networks.

We looked at various techniques applicable in solving QoS problems and viewed the same in the context of software defined networks, many authors agree that the new networking paradigm promise of making networks programmable is a huge step towards empowering the technology managers to implement their own logic on how they want their networks to behave, when it comes to quality of service literature suggests that traditional mechanism like traffic shaping and priority queuing may be programmatically implemented in the new networking paradigm but leaves it to the discretion of the practitioner to know how to implement such.

We saw this as a gap in knowledge and we sought to demonstrate how we can capitalize on network programmability provided by software defined networks to solve the problem of quality of service for important user applications. We proposed an architecture for the purposes of our demonstration.

2.5.1 Proposed Architecture

The architecture that we proposed comprised of software defined network controller and virtualized network forwarding elements, the architecture was in such a way that, forwarding rules to ensure that service quality requirements for the traffic of interest, in the midst of changing network conditions and congestion may be communicated to the forwarding elements thus manipulating network behaviour.

In our proposed architecture we choose to use static IP address and port numbers to identify the traffic of interest in the midst of several traffics types being carried on the network, the choice for this was solely based on the reality of our GPS tracking case whereby a GPS server need to have a static IP address through which all the GPS devices in the field can send collected data. figure 5 below shows our proposed architecture.

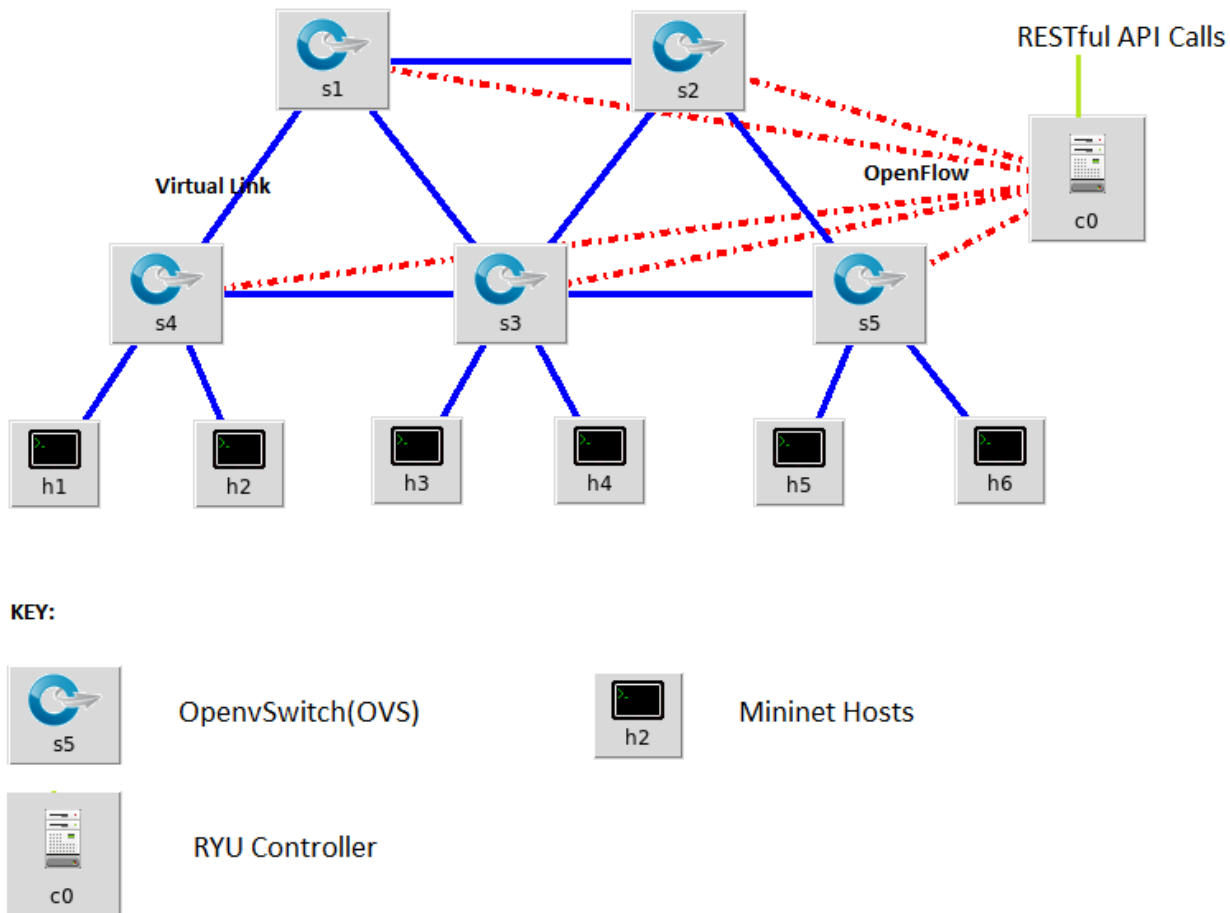


Figure 5: Proposed Architecture

CHAPTER THREE: METHODOLOGY

3.1 Introduction

This chapter describes the methodology that was used in carrying out the study; the chapter describes the tools used and the process followed in the design, implementation and evaluation of the prototype.

3.2 Prototype Design

The research design adopted for this study was a prototype experiment; this choice was guided by the need for flexibility in topology design and also cost consideration. The tools used for this research project were free and open-source widely used in research and prototyping of networks.

3.2.1 Experiment Requirements

To meet our prototype design objective we identified the design requirements to be an SDN controller, a network emulator and traffic generator. A basic PC computer was found to be adequate for the hardware platform therefore we used an Intel core i5, 6GB RAM computer, for the software, we installed Ubuntu 14.04 Linux operating system running on Oracle Virtual Box virtual machine. We created Linux virtual machines and installed RYU SDN controller, Mininet 2.2.1 network emulator and iPERF traffic generation and analysis tool for network performance measurements.

3.2.2 Experimental tools overview

3.2.2.1 Mininet

Mininet network emulator was chosen as the most ideal tool for the networks design, first, with Mininet large scale networks can be created without incurring the huge costs of hardware platform and secondly it runs real code, further open-flow switches can forward traffic at line rate in a way similar to physical hardware switches. With Mininet, topologies can be created at will and hence ideal for experimenting different topologies and scenarios with flexibility. Many researcher view Mininet as a network emulator mainly used to support research, prototyping, and testing of experimental networks.

Mininet hosts are Linux based clients/servers, the switches fully support open-flow protocol thus may be used for custom routing as required in software defined networking. In his doctoral dissertation Brandon

Heller (2013), described Mininet as an emulator that uses lightweight operating systems containers to run a complete network within a single operating system instance. Further information on Mininet was obtained from its documentation, worth mentioning is that Mininet provides an easy way to get correct system behavior and to experiment with topologies. The key emphasis on the documentation is that the code that runs on Mininet can be moved real hardware switches and work same way with regard to packet forwarding.

3.2.2.2 *Open vSwitch*

OpenvSwitch (OVS) performs multilayer switching and can be programmatically controlled by SDN controllers, this switching platform was used in this project since it works well with RYU controller and is available within Mininet platform.

The architecture of OVS is as shown below.

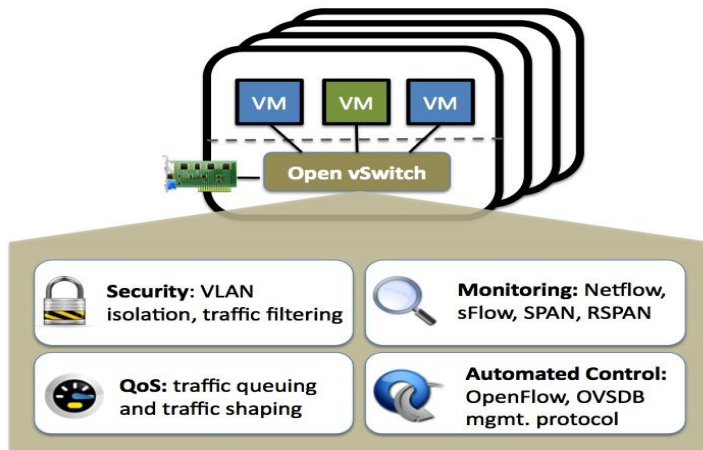


Figure 6: OpenvSwitch

Source (Openvswitch.org)

3.2.2.3 *iPERF*

iPERF is a network performance testing tool that is adequate to report throughput, delays and jitter in IP networks, this tools was useful in generating traffic from the clients side required for experimentation and also was an affective measurement tool on the server side.

3.2.2.4 RYU

RYU is a controller widely used in software defined networks, its architecture is such that it provides components with defined APIs useful to network programmers. A variety of network protocols are supported by RYU, examples are OpenFlow. In our study RYU was selected due to its ease of use and its availability as an open source SDN controller.

3.2.3 Experiment Design

In this project a policy for setting queuing and rate limiting was developed and implemented on the controller by making use of RYU framework RESTful APIs, rather than carry out custom application programming this project relied on modern approach of API calls for queuing and rate limiting via common HTTP methods acting on resources identified by URIs to program the forwarding plane with the logic of assigning strict priority to desired traffic identified by destination address and port

3.2.3.1 Topology creation

The traffic of interest representing our GPS case was modeled as a packet transmission from Host 1 (client) to Host 8 (Server) in an experimental network that was loaded (congested) with traffic.

The topology was created on Mininet via the following commands.

```
$ sudo mn -controller remote --mac --switch ovsk -topo tree, depth=3, fanout=2 -x
```

The switches south bound interface needed to support a version of open flow protocol and in our case the switches were set to use open flow version 1.3 via the following command on every switch terminal.

```
# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

The switches manager needed to listen to commands from the SDN controller thus the ports for this were set on every switch via the following command on every switch terminal.

```
# ovs-vsctl set-manager ptcp:6632
```

Now the preparation of the forwarding plane was complete and the next focus was on the SDN controller.

The first step was to prepare the controller flow processing pipeline to register entry into the flow table via the following command.

```
# sed '/OFFlowMod(././s)/, table_id=1)' ryu/ryu/app/simple_switch_13.py >  
ryu/ryu/app/qos_simple_switch_13.py
```

The controller applications were then started via the following command on the controller terminal.
`# ryu-manager ryu.app.rest_qos ryu.app.qos_simple_switch_13 ryu.app.rest_conf_switch`

The figure 7 below shows the experimental network as designed in our study.

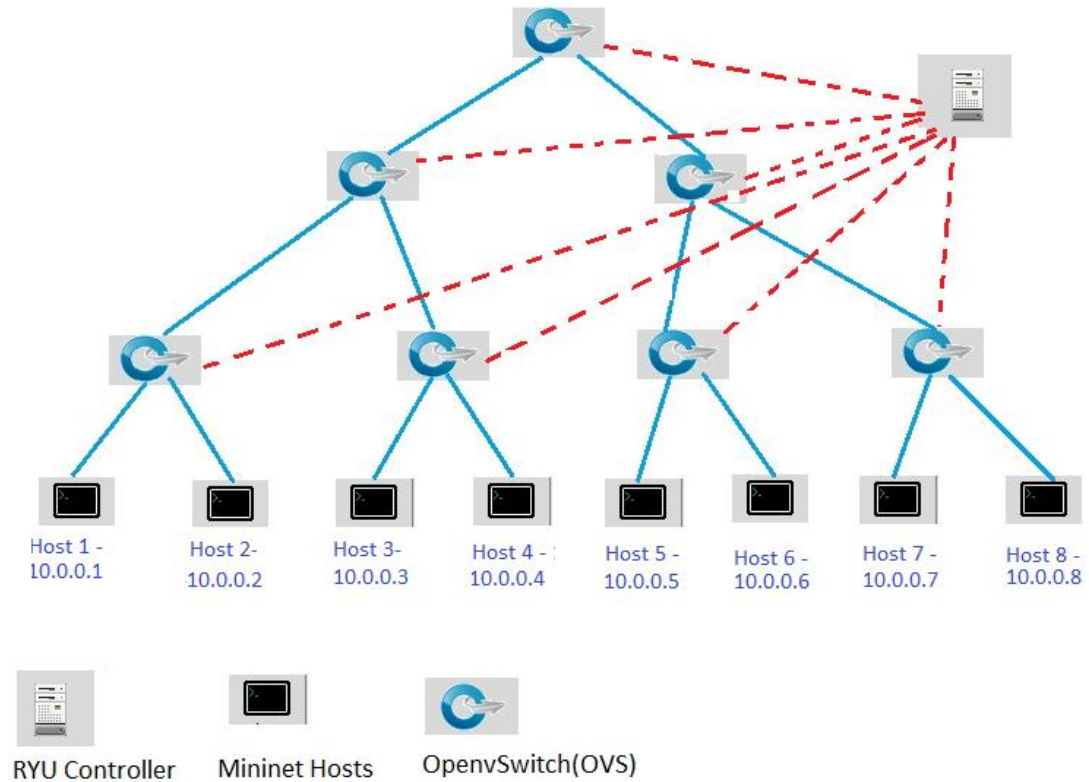


Figure 7: Prototype Topology

In this topology Hosts 5, 6, 7 and 8 were the servers to receive traffic coming from clients Host 1, Host 2 Host 3 and Host 4. The traffic considered to be of interest was coming from Host 1 destined to server Host 8 application bearing port number 5001. The role of other Hosts 2, Host 3 and Host 4 was for stressing the network with traffic hence referred to as loaders.

3.2.3.2 *The policy*

The policy to minimize network latency and maintain higher throughput for desired service was determined; the primary objective of the policy was to provide QoS guarantees to critical traffic identified by destination IP address and port number. First the desired throughput for critical service was determined; the service was identified by layer 4 port number and destination IP address.

We stated our policy as follows;

All traffic traversing the switches was to fall in to either of two queues defined and communicated to the multilayer switches via the controller. The priority queue was assigned Queue ID 1 and was assigned a minimum rate of 600Kbps while the other was assigned Queue ID 0 with a maximum rate of 384Kbps. The forwarding rule was that all traffic destined to host 8 IP address 10.0.0.8 and application port 5001 was to be put on the priority queue.

In our study the assumption was that to guarantee service quality in the case of GPS tracking, at least 600Kbps data rate would be sufficient, however we varied this in later experiments since required data rate for our case may vary in some instances.

3.2.4 Policy Implementation

Having defined the policy as above, we based our implementation on queuing and traffic shaping to accord special treatment to critical service of GPS tracking on the network. We defined the queues on the multilayer switches as per our policy using REST API calls that made use of common HTTP methods acting on resources represented by uniform resource identifiers (URIs).

The following are the HTTP methods we used to implement our policy;

To set the address of the switch database we used the following method;

```
# curl -X PUT -d "'tcp:127.0.0.1:6632'" http://localhost:8080/v1.0/conf/switches/all/ovsdb_addr
```

We further defined the rates and the queues via the following method.

```
# curl -X POST -d '{"port_name": "sx-ethx", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "384000"}, {"min_rate": "600000"}]}' http://localhost:8080/qos/queue/0000000000000000x
```

We finally installed the forwarding rule to determine how the switches queues traffic destined to our host servers via the following method.

```
# curl -X POST -d '{"match": {"nw_dst": "10.0.0.8", "nw_proto": "UDP", "tp_dst": "5001"}, "actions": {"queue": "1"}}' http://localhost:8080/qos/rules/0000000000000000x
```

3.2.5 Testing and Measurements

After implementation of the prototype the next step was to evaluate and take measurements, we proceeded as follows;

The servers were set to listen to the incoming traffic and measure bandwidth, jitter and data transferred via the following commands run each server host.

Iperf -s -u -i time -p port

On the client terminals Host 1 (acting as the source of traffic of interest coming from GPS device) Host 2, Host 3 and Host 4 (acting as loaders to congest the network) traffic destined to the servers was generated using iPERF traffic generator via the following command run on each client host.

Iperf -c -u ipaddress of server -p portnumber -t time

We carried out several rounds of experiments using different source data rates generated at the client hosts i.e. 300Kbps, 400Kbps, 500Kbps, 700Kbps and finally 800Kbps. We obtained experimental data via iPERF on the receiving server hosts. In our setup traffic of interest representing our case of GPS tracking was the one generated on Host 1 (GPS device) and sent to Host 8 port 5001 (GPS Server), role of the other generated traffic on Host 1, Host 2, Host 3, Host 4 was for congesting the network. For clarity table 1 below shows the sending hosts and corresponding receiving host of the network prototype;

SENDER (SOURCE CLIENT)		CORRESPONDING RECEIVER(SERVER)		
HOST NAME	IP ADDRESS	HOST NAME	IP ADDRESS	UDP PORT
Host 1	10.0.0.1	Host 8	10.0.0.8	5001
Host 2	10.0.0.2	Host 7	10.0.0.7	5001
Host 3	10.0.0.3	Host 6	10.0.0.6	5001
Host 4	10.0.0.4	Host 5	10.0.0.5	5001
Host 1	10.0.0.1	Host 8	10.0.0.8	5002
Host 2	10.0.0.2	Host 8	10.0.0.8	5003

Table 1: Sender and Receiver hosts

Our choice of 600Kbps rate for our case of GPS tracking was based on assumption that it is adequate to provide the required service level in the midst of network congestion arising from other competing traffics. We recognize that the ideal rate varies from case to case since as discussed before a GPS device also acts as a hub for other sensors as well, meaning the more the sensors the more the requirement for higher rates. We therefore used a range of different rates to evaluate our policy, in this case we used the range of 200Kbps to 700Kbps for the higher priority queue, we also varied the maximum rate for less priority queue in the range of 484Kbps to 10Kbps as shown in table 2 below and re-run the experiment in turns.

Evaluation test number	Minimum rate (Kbps)	Maximum rate(Kbps)	Remark
1	700	484	
2	600	384	Carried out in first round
3	500	284	
4	400	184	
5	300	84	
6	200	10	

Table 2: Policy rates

In both cases we recorded the data transferred between hosts, throughput and jitter as measured by iPERF on the receiving Hosts 8, Host 7, Host 6 and Host 5. To obtain more realistic results, on Host 8 we measured traffic arriving to other processes on the same server identified by port numbers 5001, 5002 and 5003.

We further computed data transferred between hosts cumulatively in intervals of five seconds. The relevant results obtained were tabulated and are shown in the tables 5, table 6, table 7 and table 8 appearing on the appendices section of this report for some of the experiments carried out in the first and second instance.

CHAPTER FOUR: RESULTS, DISCUSSION AND CONCLUSION

4.1 Results and Discussion

We obtained results from all rounds of experiments and represented some of it graphically. First was when the the policy rate was fixed at 600Kbps minimum for the traffic of interest assumed to be coming from the GPS device and 384Kbps maximum for the other non critical traffic, Secondly is when the policy rate was varied to 500Kbps minimum for traffic on interest and 284Kbps maximum for the other traffics, similiary we varied the policy rate to 400Kbps minimum and 184Kbps maximum, 300Kbps minimum and 84Kbps maximum, 200Kbps minimum and 10Kbps maximum and made further observations.

We plotted on a graph throughput Vs time interval and Cumulative data transferred Vs time interval for some of the experiments for the purposes of clarity and for ease of discussion since the observed trend was the same.

First when we sent 500Kbps from all the hosts clients and measured on the servers hosts we found that the bandwidth of traffic originating from Host 1 and destined for Host 8 port 5001 remained steady at around the same rate of 500Kbps, all the other traffic suffered degradation but did not exceed 384Kbps as shown in the figure 8 below. This shows that there was preferential treatment of the traffic which we considered to be of interest and coming from the GPS device despite the heavy network load caused by traffic from other clients in the network prototype.

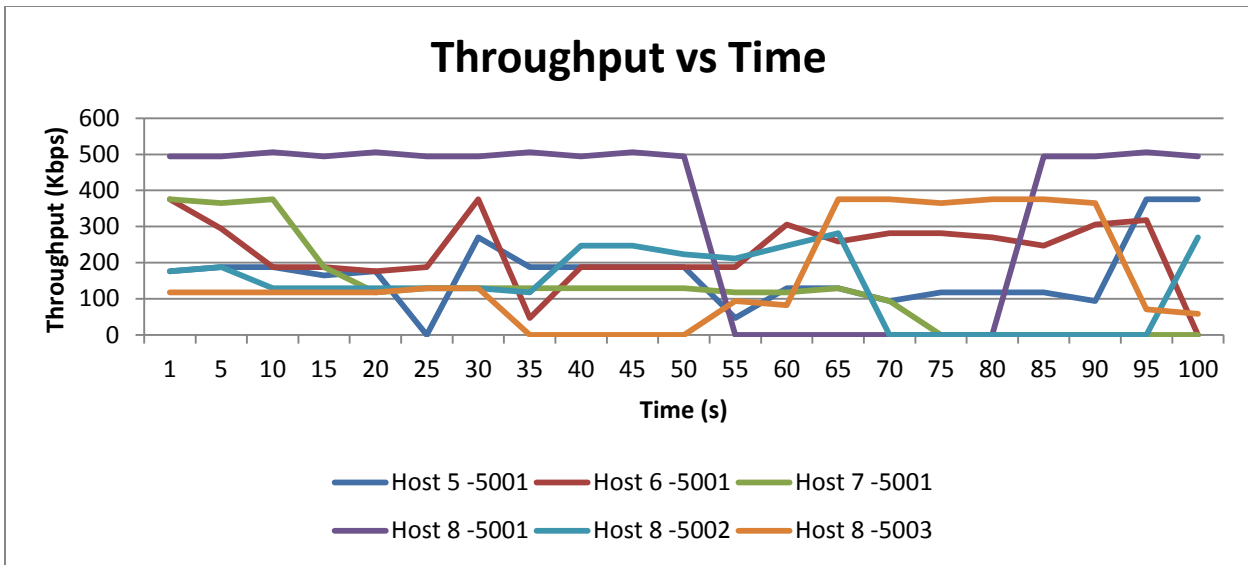


Figure 8: Throughput vs Time, client rate 500Kbps

For the same experiment we computed the cumulative data transferred measured in an interval of 5 seconds at the receiving host servers, we found that as shown in figure 9 below more data was cumulatively transferred between Host 1 and Host 8 port 5001 as compared to the rest, meaning that when the network was congested by traffic from other hosts, priority was accorded to our traffic of interest deemed to be originating from the GPS device and that is the reason more data cumulatively transferred.

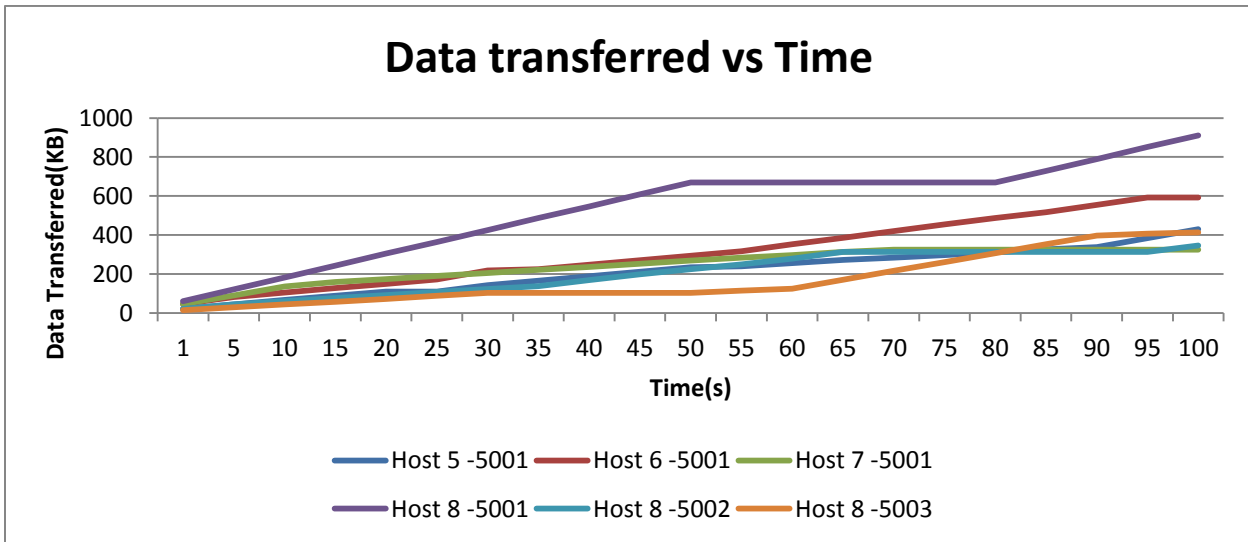


Figure 9: Data transferred Vs Time, client rate 500Kbps

Similarly when we changed the clients sending rate to 800Kbps we observed that bandwidth of traffic originating from Host 1 and destined for Host 8 port 5001 remained steady at around the same rate of 800Kbps, all the other traffic suffered degradation but did not exceed 384Kbps as shown in figure 10 below , this again pointed to preferential treatment of our traffic of interest in the network.

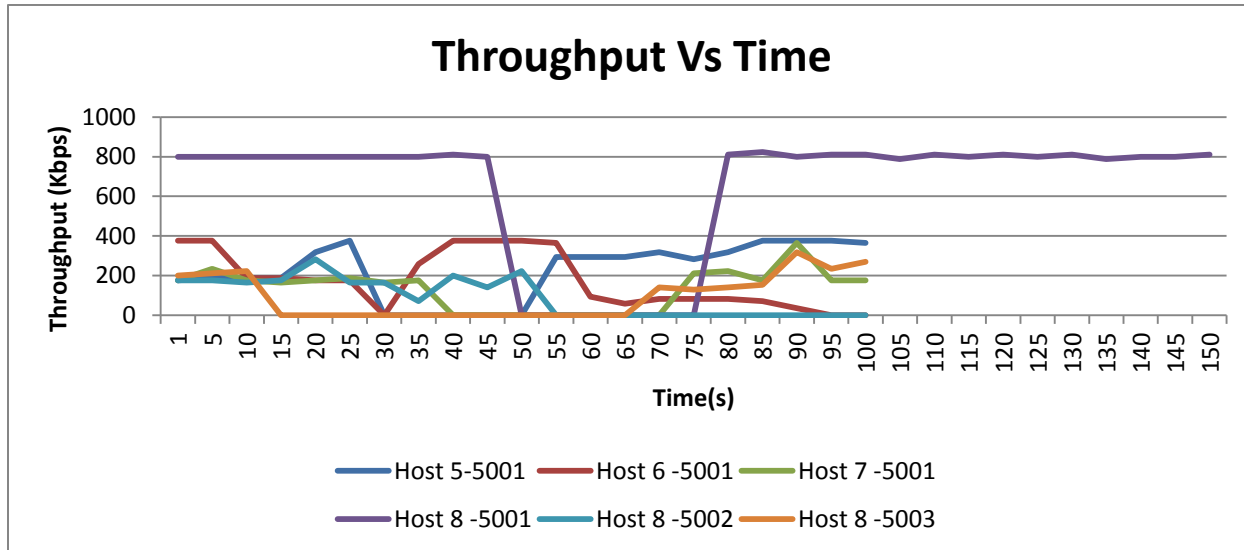


Figure 10: Throughput Vs Time, client rate 800Kbps

We also observed that, like in the case of client sending rate at 500Kbps, when we sent 800Kbps the cumulative data transferred measured in an interval of 5 seconds at the receiving host servers was highest for the traffic of interest which is between Host 1 and Host 8 port 5001 as shown in figure 11 below, again showing that our traffic of interest was accorded priority on the network.

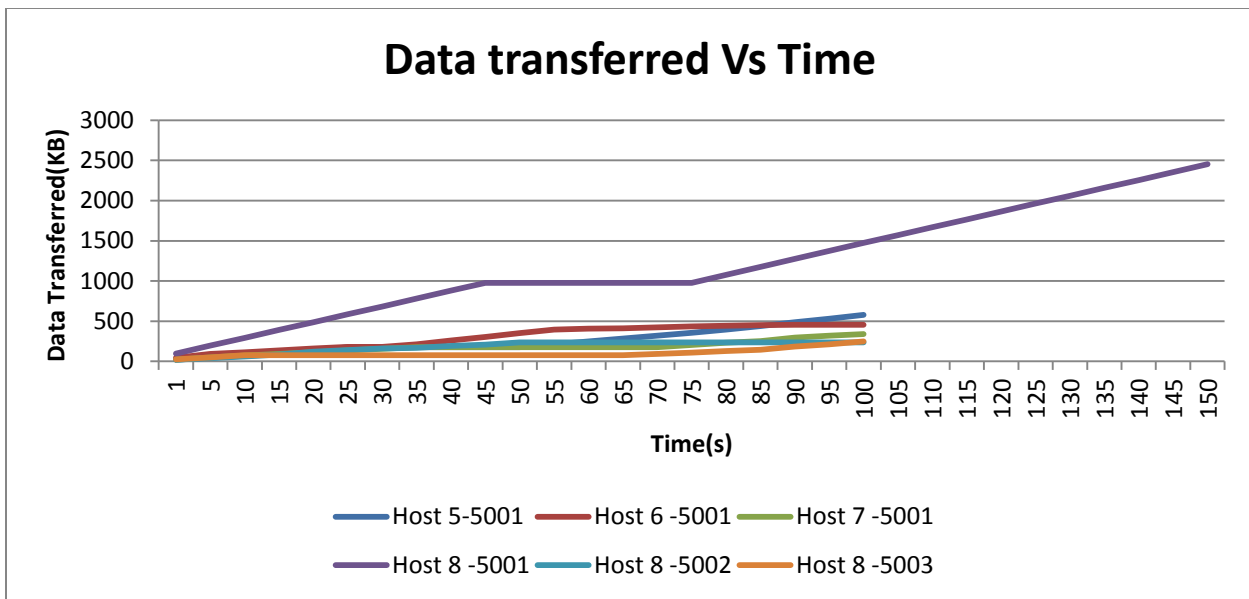


Figure 11: Data transferred Vs Time, client rate 800Kbps

4.1.1 Policy evaluation

In the second round of experiments where we sought to know if the observed trend will change when policy data rate is different, when we had set 500Kbps to be the minimum for the high priority queue and 284Kbps to be the maximum for the low priority queue. We found that as shown in figure 12 below the bandwidth of traffic originating from Host 1 and destined for Host 8 port 5001 remained steady at around 800Kbps, all the other traffic suffered degradation but did not exceed 284Kbps, this showed that the traffic deemed to be originating from the GPS device was not degraded like other traffics in the midst of heavy network load, further it shows that if the required data rate for GPS tracking is different from the assumed 600Kbps minimum then the policy would still enforce the ideal rate.

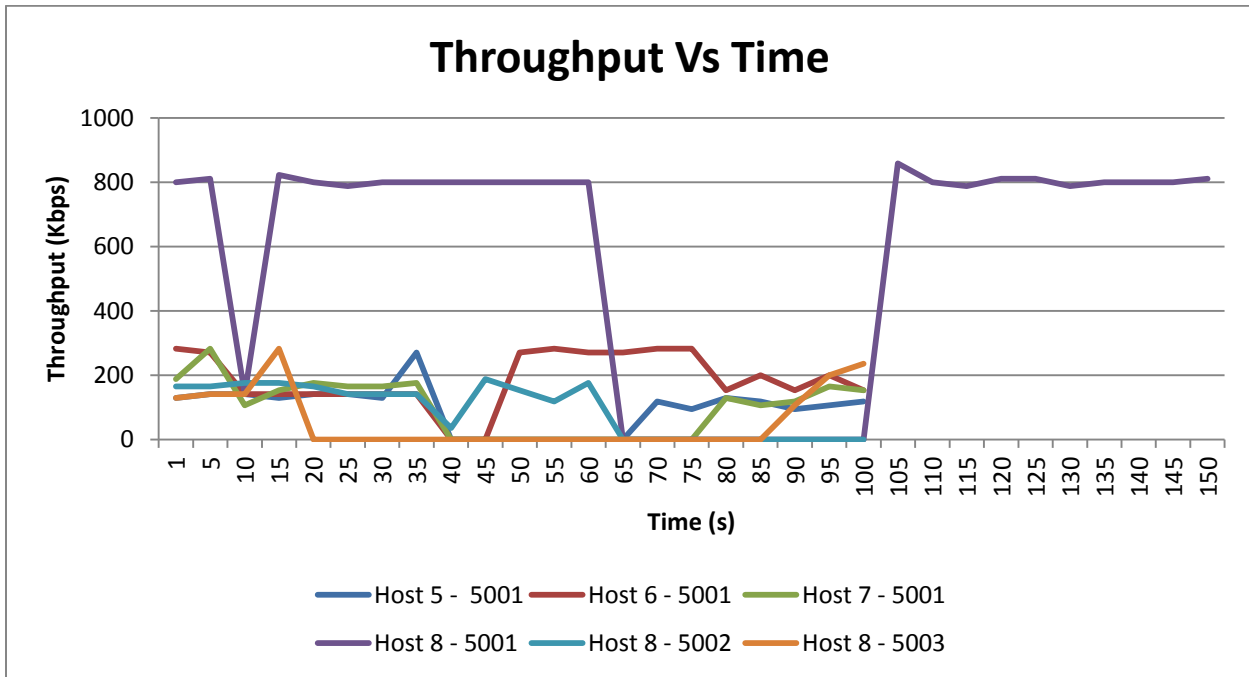


Figure 12: Throughput Vs Time, Policy 500Kbps max /284Kbps min

Similarily as shown in figure 13 below we found that more data was cumulatively transferred between Host 1 and Host 8 port 5001 as compared to the rest meaning regardless of the required rate the priority policy would still work.

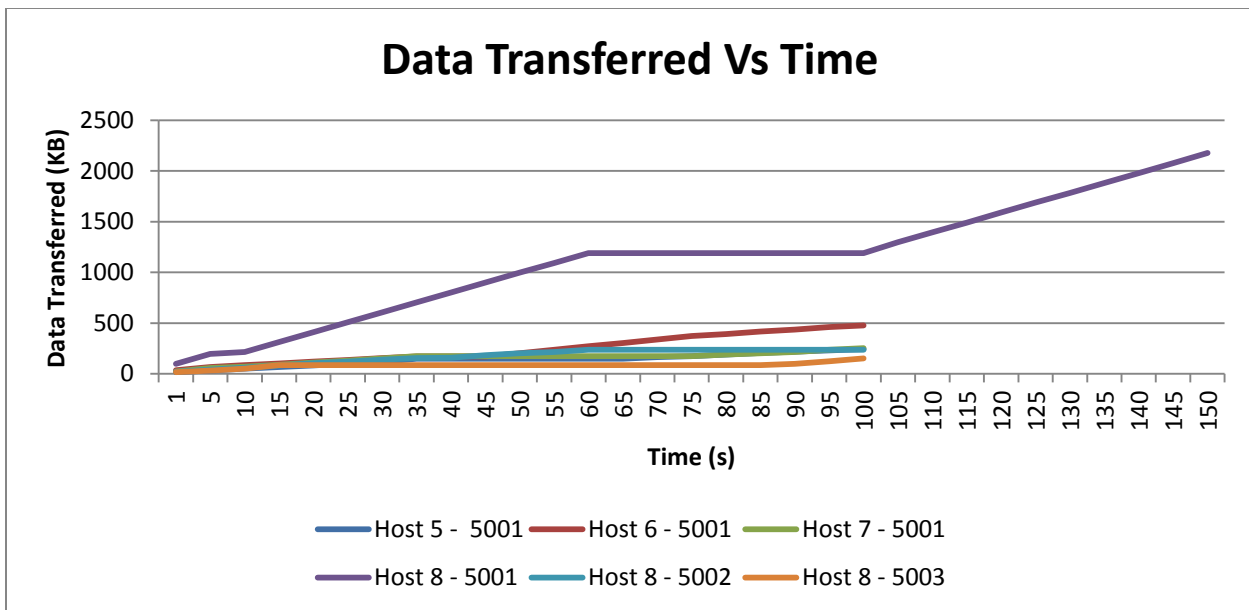


Figure 13: Data transferred Vs Time, policy 500Kbps min/284Kbps max

4.2 Conclusion

In this research project fundamental concepts of software defined networks and network functions were investigated with a view to understanding the technological foundations, further the current state of affairs in the adoption of the same was studied with a view to understanding the problem areas and the challenges faced. Since our work focused on network performance we looked at traditional approaches and techniques employed in improving quality of service of networks, we further looked into how the new approach to networks architectures occasioned by the shift to network softwarization affects network performance. We sought to demonstrate how network designers can take advantage of SDN to deploy their own network policies, in particular to improve the performance of networks and most importantly meet requirements for some of the services or applications carried on networks. In our study we capitalized on the concept of traffic shaping and priority queues as the key techniques to realize acceptable quality of service for our GPS tracking case.

We deployed a network prototype on Mininet network prototyping platform therefore enabling open vSwitch a popular virtualized switch to be used for the experiment. RYU software defined network controller was deployed to act as the control plane for our prototype. A priority queue policy was developed, implemented and tested. In our study we placed emphasis on programmability of the control plane as one of the avenues that can be followed to solve the quality of service problem, in contrast to

traditional approaches our work revealed that it was possible to programmatically apply policies centrally from the controller to meet our desired network behavior. In our study the traffic of interest was accorded the highest priority on the network, despite the congestion experienced the result shows that as per our policy there was no degradation of our traffic of interest as compared to other traffic flows, this shows that by exploiting the programmability of network provided by new network architectures it is possible to create and deploy with flexibility policies that enable meeting varying QoS needs for different types of services or applications. This is essence further shows that in live commercial networks the quality of service for various communication services can be dealt with by network managers own policies with flexibility, this is in complete departure from the past approaches where the issues of QoS was addressed in a proprietary fashion with sometimes the algorithms hardwired in the network equipment.

4.3 Study Limitations and Suggestion for Further Work

The limitation of this study is that the scalability of the proposed solution was not evaluated in this experiment, We noted that there was characteristic drop in performance of the emulated network under heavy network load, worth noting is that even the traffic considered priority suffered this performance problem at some instant of time, we suggest further study to ascertain the cause of this. In our opinion the scalability of the controller to process flows in the midst of huge traffic load may need to be studied.

APPENDIX A: EXPERIMENTAL DATA SHEETS

Results for Fixed policy rate

Table 3: Host 8 source 500Kbps

Host 8 (10.0.0.8) port 5001, Port 5002, Port 5003 Source 500Kbps												
Time	Data Transferred			Bandwidth			Jitter			Cumulative data Transferred(Kbytes)		
	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003
1	60.3	21.5	14.4	494	176	118	0.015	23.587	31.287	60.3	21.5	14.4
5	60.3	23.0	14.4	494	188	118	0.057	39.222	68.368	120.6	44.5	28.8
10	61.7	15.8	14.4	506	129	118	0.045	67.465	70.921	182.3	60.3	43.2
15	60.3	15.8	14.4	494	129	118	0.065	70.875	71.026	242.6	76.1	57.6
20	61.7	15.8	14.4	506	129	118	0.058	71.021	70.971	304.3	91.9	72
25	60.3	15.8	15.8	494	129	129	0.034	70.997	71.083	364.6	107.7	87.8
30	60.3	15.8	15.8	494	129	129	0.040	70.985	71.068	424.9	123.5	103.6
35	61.7	14.4	0.0	506	118	0	0.039	70.964	70.995	486.6	137.9	103.6
40	60.3	30.1	0.0	494	247	0	0.062	22.514	70.995	546.9	168	103.6
45	61.7	30.1	0.0	506	247	0	0.053	9.424	70.995	608.6	198.1	103.6
50	60.3	27.3	0.0	494	223	0	0.054	7.536	70.995	668.9	225.4	103.6
55	0.0	23.80	11.50	0	212	94.1	0.035	8.194	33.889	668.9	249.2	115.1
60	0.0	30.1	10.0	0	247	82.3	0.035	7.772	8.736	668.9	279.3	125.1
65	0.0	34.5	45.9	0	282	376	0.035	9.316	11.253	668.9	313.8	171
70	0.0	0.0	45.9	0	0	376	0.035	12.282	10.982	668.9	313.8	216.9
75	0.0	0.0	44.5	0	0	365	0.035	12.282	10.574	668.9	313.8	261.4
80	0.0	0.0	45.9	0	0	376	0.035	12.282	12.318	668.9	313.8	307.3
85	60.3	0.0	45.9	494	0	376	0.049	12.282	12.375	729.2	313.8	353.2
90	60.3	0.0	44.5	494	0	365	0.035	12.282	12.459	789.5	313.8	397.7
95	61.7	0.0	8.61	506	0	70.6	0.047	12.282	12.253	851.2	313.8	406.31
100	60.3	33.0	7.18	494	270	58.8	0.058	9.840	6.981	911.5	346.8	413.49

Table 4: Host 5, Host 6 and Host 7 source 500kbps

Host 5 (10.0.0.5) port 5001, Host 6 (10.0.0.6) port 5001, Host 7 (10.0.0.7) port 5001 Source 500Kbps												
Time(s)	Data Transferred(Kbytes)			Bandwidth(Kbps)			Jitter(ms)			Cumulative data Transferred(Kbytes)		
	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7
1	21.5	45.9	45.9	176	376	376	23.593	6.667	6.663	21.5	45.9	45.9
5	23	35.9	44.5	188	294	365	39.247	18.071	7.953	44.5	81.8	90.4
10	23	23	45.9	188	188	376	39.487	39.345	7.959	67.5	104.8	136.3
15	20.1	23	23	165	188	188	52.07	39.506	38.726	87.6	127.8	159.3
20	21.5	21.5	14.4	176	176	118	39.586	34.544	62.865	109.1	149.3	173.7
25	0	23	15.8	0	188	129	39.507	39.46	70.418	109.1	172.3	189.5
30	33	45.9	15.8	270	376	129	55.695	7.984	70.987	142.1	218.2	205.3
35	23	5.74	15.8	188	47	129	39.505	59.023	71.024	165.1	223.94	221.1
40	23	23	15.8	188	188	129	39.485	39.614	71.003	188.1	246.94	236.9
45	23	23	15.8	188	188	129	39.499	39.332	70.969	211.1	269.94	252.7
50	23	23	15.8	188	188	129	39.494	39.497	70.962	234.1	292.94	268.5
55	5.74	23	14.4	47	188	118	16.638	39.603	4.674	239.84	315.94	282.9
60	15.8	37.3	14.4	129	306	118	6.722	10.079	2.332	255.64	353.24	297.3
65	15.8	31.6	15.8	129	259	129	2.187	10.119	2.122	271.44	384.84	313.1
70	11.5	34.5	11.5	94.1	282	94.1	4.12	9.359	3.862	282.94	419.34	324.6
75	14.4	34.5	0	118	282	0	5.294	9.313	2.834	297.34	453.84	324.6
80	14.4	33	0	118	270	0	2.164	10.598	2.834	311.74	486.84	324.6
85	14.4	30.1	0	118	247	0	4.447	10.335	2.834	326.14	516.94	324.6
90	11.5	37.3	0	94.1	306	0	4.144	10.329	2.834	337.64	554.24	324.6
95	45.9	38.8	0	376	318	0	11.993	10.161	2.834	383.54	593.04	324.6
100	45.9	0	0	376	0	0	12.175	10.659	2.834	429.44	593.04	324.6

Table 5: Host 8 source 800kbps

Host 8 (10.0.0.8) port 5001, Port 5002, Port 5003 Source 800Kbps												
Time(s)	Data Transferred(Kbytes)			Bandwidth(Kbps)			Jitter			Cumulative data Transferred(Kbytes)		
	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003
1	97.6	21.5	24.4	800	176	200	0.032	27.991	25.935	97.6	21.5	24.4
5	97.6	21.5	25.8	800	176	212	0.036	45.291	40.608	195.2	43	50.2
10	97.6	20.1	27.3	800	165	223	0.041	45.772	40.144	292.8	63.1	77.5
15	97.6	21.5	0	800	176	0	0.047	47.756	46.863	390.4	84.6	77.5
20	97.6	34.5	0	800	282	0	4.99	26.008	46.863	488	119.1	77.5
25	97.6	20.1	0	800	165	0	4.445	50.614	46.863	585.6	139.2	77.5
30	97.6	20.1	0	800	165	0	4.156	53.288	46.863	683.2	159.3	77.5
35	97.6	8.61	0	800	70.6	0	4.256	123.384	46.863	780.8	167.91	77.5
40	99.1	24.4	0	811	200	0	4.463	70.417	46.863	879.9	192.31	77.5
45	97.6	17.2	0	800	141	0	3.826	79.167	46.863	977.5	209.51	77.5
50	0	27.3	0	0	223	0	4.291	59.123	46.863	977.5	236.81	77.5
55	0	0	0	0	0	0	4.291	72.084	46.863	977.5	236.81	77.5
60	0	0	0	0	0	0	4.291	72.084	46.863	977.5	236.81	77.5
65	0	0	0	0	0	0	4.291	72.084	46.863	977.5	236.81	77.5
70	0	0	17.2	0	0	141	4.291	72.084	46.512	977.5	236.81	94.7
75	0	0	15.8	0	0	129	4.291	72.084	43.916	977.5	236.81	110.5
80	99.1	0	17.2	811	0	141	4.893	72.084	39.297	1076.6	236.81	127.7
85	100	0	18.7	823	0	153	6.149	72.084	44.013	1176.6	236.81	146.4
90	97.6	0	38.8	800	0	318	6.173	72.084	4.965	1274.2	236.81	185.2
95	99.1	0	28.7	811	0	235	6.144	72.084	57.443	1373.3	236.81	213.9
100	99.1	0	33	811	0	270	5.999	72.084	49.638	1472.4	236.81	246.9

Table 6: Host 5, Host 6 and Host 7, source 800kbps

Host 5 (10.0.0.5) port 5001, Host 6 (10.0.0.6) port 5001, Host 7 (10.0.0.7) port 5001 Source 800Kbps												
Time(s)	Data Transferred(Kbytes)			Bandwidth(Kbps)			Jitter(ms)			Cumulative data Transferred(Kbytes)		
	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7
1	21.5	45.9	21.5	176	376	176	28.036	14.333	32.637	21.5	45.9	21.5
5	23	45.9	28.7	188	376	235	47.941	16.788	36.985	44.5	91.8	50.2
10	23	23	21.5	188	188	176	48.305	45.196	76.126	67.5	114.8	71.7
15	23	23	20.1	188	188	165	48.387	42.262	52.332	90.5	137.8	91.8
20	38.8	21.5	21.5	318	176	176	24.439	48.44	54.687	129.3	159.3	113.3
25	45.9	21.5	23	376	176	188	16.875	48.311	51.863	175.2	180.8	136.3
30	0	0	20.1	0	0	165	16.789	48.388	55.388	175.2	180.8	156.4
35	0	31.6	21.5	0	259	176	16.789	57.155	52.101	175.2	212.4	177.9
40	0	45.9	0	0	376	0	16.789	16.842	50.801	175.2	258.3	177.9
45	0	45.9	0	0	376	0	16.789	16.835	50.801	175.2	304.2	177.9
50	0	45.9	0	0	376	0	16.789	16.792	50.801	175.2	350.1	177.9
55	35.9	44.5	0	294	365	0	5.256	4.398	50.801	211.1	394.6	177.9
60	35.9	11.5	0	294	94.1	0	4.213	6.321	50.801	247	406.1	177.9
65	35.9	7.18	0	294	58.8	0	5.009	6.696	50.801	282.9	413.28	177.9
70	38.8	10	0	318	82.3	0	4.059	7.028	50.801	321.7	423.28	177.9
75	34.5	10	25.8	282	82.3	212	4.946	7.51	266.894	356.2	433.28	203.7
80	38.8	10	27.3	318	82.3	223	3.514	5.765	63.418	395	443.28	231
85	45.9	8.61	21.5	376	70.6	176	3.748	3.842	72.068	440.9	451.89	252.5
90	45.9	4.31	44.5	376	35.3	365	5.309	5.217	15.253	486.8	456.2	297
95	45.9	0	21.5	376	0	176	4.369	5.217	44.143	532.7	456.2	318.5
100	44.5	0	21.5	365	0	176	3.557	5.217	37.88	577.2	456.2	340

APPENDIX B: EVALUATION RESULTS FOR VARYING POLICY RATE

Table 7: Policy Rate 500kbps Min 284Kbps Max, Host 8 Port 5001, Port 5002, Port 5003

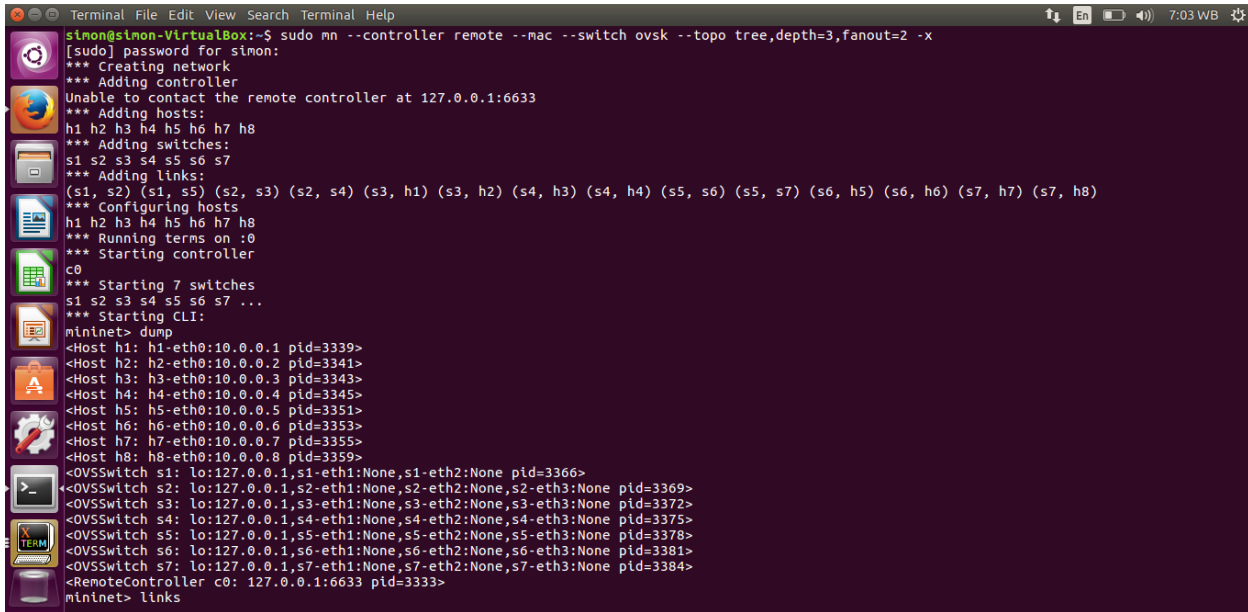
Host 8 (10.0.0.8) Source 800Kbps												
Time(s)	Data Transferred (Kbytes)			Bandwidth (Kbps)			Jitter (ms)			Cumulative data Transferred (Kbytes)		
	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003	Port 5001	Port 5002	Port 5003
1	97.6	20.1	15.8	800	165	129	0.010	29.770	31.891	97.6	20.1	15.8
5	99.1	20.1	17.2	811	165	141	5.655	51.133	69.578	196.7	40.2	33
10	17.2	21.5	17.2	141	176	141	16.745	53.164	70.319	213.9	61.7	50.2
15	100.0	21.5	34.5	823	176	282	3.935	52.762	39.908	313.9	83.2	84.7
20	97.6	20.1	0.0	800	165	0	4.126	53.359	39.597	411.5	103.3	84.7
25	96.2	17.2	0.0	788	141	0	3.947	62.438	39.597	507.7	120.5	84.7
30	97.6	17.2	0.0	800	141	0	3.565	70.529	39.597	605.3	137.7	84.7
35	97.6	17.2	0.0	800	141	0	3.902	70.632	39.597	702.9	154.9	84.7
40	97.6	4.3	0.0	800	35.3	0	4.553	145.719	39.597	800.5	159.2	84.7
45	97.6	23.0	0.0	800	188	0	5.257	44.572	39.597	898.1	182.2	84.7
50	97.6	18.7	0.0	800	153	0	5.570	47.812	39.597	995.7	200.9	84.7
55	97.6	14.4	0.0	800	118	0	6.311	60.846	39.597	1093.3	215.3	84.7
60	97.6	21.5	0.0	800	176	0	6.074	46.627	39.597	1190.9	236.8	84.7
65	0.0	0.0	0.0	0	0	0	7.370	49.950	39.597	1190.9	236.8	84.7
70	0.0	0.0	0.0	0	0	0	7.370	49.950	39.597	1190.9	236.8	84.7
75	0.0	0.0	0.0	0	0	0	7.370	49.950	39.597	1190.9	236.8	84.7
80	0.0	0.0	0.0	0	0	0	7.370	49.950	39.597	1190.9	236.8	84.7
85	0.0	0.0	0.0	0	0	0	7.370	49.950	39.597	1190.9	236.8	84.7
90	0.0	0.0	12.9	0	0	106	7.370	49.950	257.569	1190.9	236.8	97.6
95	0.0	0.0	24.4	0	0	200	7.370	49.950	4.746	1190.9	236.8	122
100	0.0	0.0	28.7	0	0	235	7.370	49.950	49.074	1190.9	236.8	150.7

Table 8: Policy Rate 500Kbps Min 284 Max, Host 5, Host 6, Host 7 Ports 5001

Host 5 (10.0.0.5) port 5001, Host 6 (10.0.0.6) port 5001, Host 7 (10.0.0.7) port 5001, Source 800Kbps												
Time(s)	Data Transferred (Kbytes)			Bandwidth (Kbps)			Jitter (ms)			Cumulative data Transferred (Kbytes)		
	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7	Host 5	Host 6	Host 7
1	15.8	34.5	23	129	282	188	33.762	21.344	31.457	15.8	34.5	23
5	17.2	33	34.5	141	270	282	69.651	27.894	33.314	33	67.5	57.5
10	17.2	17.2	12.9	141	141	106	70.477	67.017	87.495	50.2	84.7	70.4
15	15.8	17.2	18.7	129	141	153	70.504	70.408	55.043	66	101.9	89.1
20	17.2	17.2	21.5	141	141	176	70.504	70.487	53.671	83.2	119.1	110.6
25	17.2	17.2	20.1	141	141	165	70.51	70.509	53.632	100.4	136.3	130.7
30	15.8	17.2	20.1	129	141	165	70.517	70.594	53.755	116.2	153.5	150.8
35	33	17.2	21.5	270	141	176	28.036	70.497	51.343	149.2	170.7	172.3
40	0	0	0	0	0	0	27.989	70.498	55.448	149.2	170.7	172.3
45	0	0	0	0	0	0	27.989	70.498	55.448	149.2	170.7	172.3
50	0	33	0	0	270	0	27.989	28.297	55.448	149.2	203.7	172.3
55	0	34.5	0	0	282	0	27.989	27.918	55.448	149.2	238.2	172.3
60	0	33	0	0	270	0	27.989	27.946	55.448	149.2	271.2	172.3
65	0	33	0	0	270	0	27.989	27.913	55.448	149.2	304.2	172.3
70	14.4	34.5	0	118	282	0	113.307	4.078	55.448	163.6	338.7	172.3
75	11.5	34.5	0	94.1	282	0	10.232	2.61	55.448	175.1	373.2	172.3
80	15.8	18.7	15.8	129	153	129	51.373	7.018	178.262	190.9	391.9	188.1
85	14.4	24.4	12.9	118	200	106	7.381	4.5	67.892	205.3	416.3	201
90	11.5	18.7	14.4	94.1	153	118	4.308	9.789	61.953	216.8	435	215.4
95	12.9	24.4	20.1	106	200	165	6.359	2.75	40.022	229.7	459.4	235.5
100	14.4	18.7	18.7	118	153	153	9.269	4.001	55.02	244.1	478.1	254.2

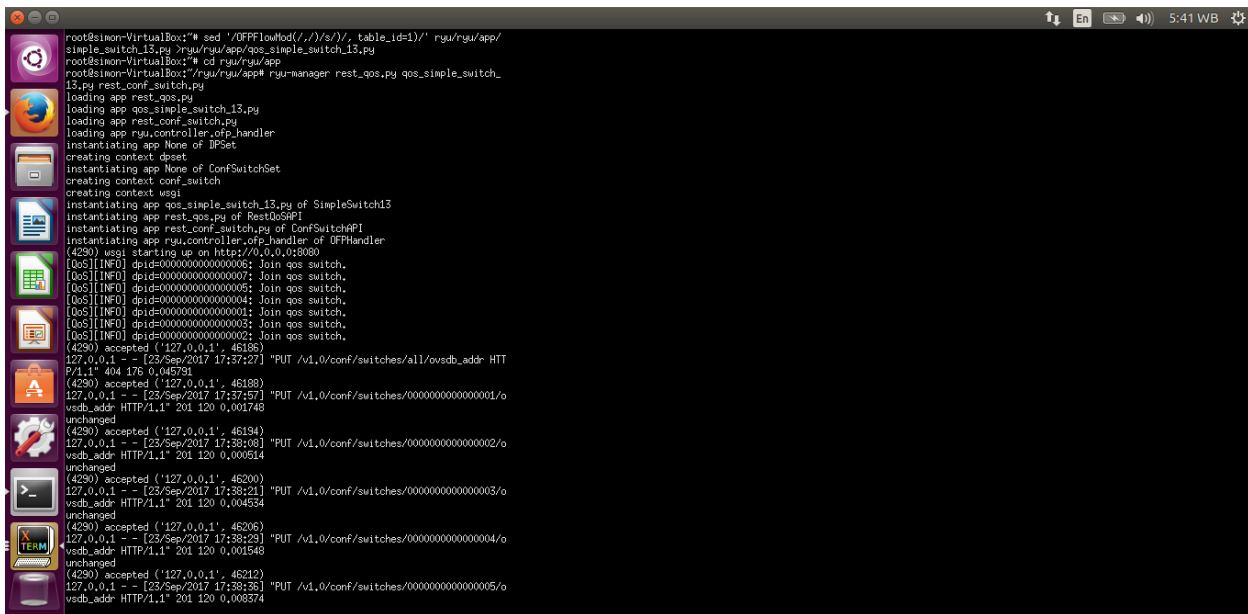
APPENDIX C: SCREEN SHOTS

Topology creation



```
simon@simon-VirtualBox:~$ sudo mn --controller remote --nac --switch ovsk --topo tree,depth=3,fanout=2 -x
[sudo] password for simon:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Running terms on :0
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3339>
<Host h2: h2-eth0:10.0.0.2 pid=3341>
<Host h3: h3-eth0:10.0.0.3 pid=3343>
<Host h4: h4-eth0:10.0.0.4 pid=3345>
<Host h5: h5-eth0:10.0.0.5 pid=3351>
<Host h6: h6-eth0:10.0.0.6 pid=3353>
<Host h7: h7-eth0:10.0.0.7 pid=3355>
<Host h8: h8-eth0:10.0.0.8 pid=3359>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3366>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=3369>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=3372>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=3375>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None pid=3378>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None pid=3381>
<OVSSwitch s7: lo:127.0.0.1,s7-eth1:None,s7-eth2:None,s7-eth3:None pid=3384>
<RemoteController c0: 127.0.0.1:6633 pid=3333>
mininet> links
```

Controller



```
root@simon-VirtualBox:~# sed -i /OFFFlowMod(/,/)s/\/, table_id=1/' rsu/ryu/app/
simple_switch_13.py >rsu/ryu/app/qos_simple_switch_13.py
root@simon-VirtualBox:~# cd rsu/ryu/app
root@simon-VirtualBox:~/rsu/ryu/app# rsu-manager rest_qos.py qos_simple_switch_
13.py rest_conf_switch.py
loading app rest_qos.py
loading app qos_simple_switch_13.py
loading app rest_conf_switch.py
loading app rsu_controller_ofp_handler
Instantiating app None of IPSet
creating context dpset
Instantiating app None of ConfSwitchSet
creating context conf_switch
creating context wsgi
Instantiating app qos_simple_switch_13.py of SimpleSwitch13
Instantiating app rest_qos.py of RestQoSAPI
Instantiating app rest_conf_switch.py of ConfSwitchAPI
Instantiating app rsu_controller_ofp_handler of OFPHandler
(4290) wsgi starting up on http://0.0.0.0:8080
[0oS][INFO] dpid=000000000000000000: Join qos switch.
[0oS][INFO] dpid=00000000000000000002: Join qos switch.
[0oS][INFO] dpid=00000000000000000005: Join qos switch.
[0oS][INFO] dpid=00000000000000000004: Join qos switch.
[0oS][INFO] dpid=00000000000000000001: Join qos switch.
[0oS][INFO] dpid=00000000000000000003: Join qos switch.
[0oS][INFO] dpid=00000000000000000002: Join qos switch.
(4290) accepted ("127.0.0.1", 46186)
127.0.0.1 - - [23/Sep/2017 17:37:27] "PUT /v1.0/conf/switches/all/ovsdb_addr HTTP
/1.1" 404 176 0,045791
(4290) accepted ("127.0.0.1", 46188)
127.0.0.1 - - [23/Sep/2017 17:37:37] "PUT /v1.0/conf/switches/0000000000000001/o
vsdb_addr HTTP/1.1" 201 120 0,001748
unchanged
(4290) accepted ("127.0.0.1", 46194)
127.0.0.1 - - [23/Sep/2017 17:38:08] "PUT /v1.0/conf/switches/0000000000000002/o
vsdb_addr HTTP/1.1" 201 120 0,000514
unchanged
(4290) accepted ("127.0.0.1", 46200)
127.0.0.1 - - [23/Sep/2017 17:38:21] "PUT /v1.0/conf/switches/0000000000000003/o
vsdb_addr HTTP/1.1" 201 120 0,004534
unchanged
(4290) accepted ("127.0.0.1", 46206)
127.0.0.1 - - [23/Sep/2017 17:38:29] "PUT /v1.0/conf/switches/0000000000000004/o
vsdb_addr HTTP/1.1" 201 120 0,001546
unchanged
(4290) accepted ("127.0.0.1", 46212)
127.0.0.1 - - [23/Sep/2017 17:38:36] "PUT /v1.0/conf/switches/0000000000000005/o
vsdb_addr HTTP/1.1" 201 120 0,006374
```

```

packet in 4 36:07:30:f5:5f:ef 33:33:00:00:00:fb 3
packet in 2 7a:4f:9a:1c:4b:9e 33:33:00:00:00:fb 3
packet in 3 7a:4f:9a:1c:4b:9e 33:33:00:00:00:fb 3
packet in 6 4a:59:cb:4b:37:22 33:33:00:00:00:fb 3
packet in 7 4a:59:cb:4b:37:22 33:33:00:00:00:fb 3
packet in 4 7a:4f:9a:1c:4b:9e 33:33:00:00:00:fb 3
packet in 3 82:dd:e7:76:27:5c 33:33:00:00:00:fb 3
packet in 5 8a:e0:69:1d:d3:ab 33:33:00:00:00:fb 3
packet in 6 8a:e0:69:1d:d3:ab 33:33:00:00:00:fb 3
packet in 7 8a:e0:69:1d:d3:ab 33:33:00:00:00:fb 3
packet in 5 0a:02:56:04:a9:13 33:33:00:00:00:fb 1
packet in 7 0a:02:56:04:a9:13 33:33:00:00:00:fb 3
packet in 1 0a:02:56:04:a9:13 33:33:00:00:00:fb 2
packet in 2 0a:02:56:04:a9:13 33:33:00:00:00:fb 3
packet in 3 0a:02:56:04:a9:13 33:33:00:00:00:fb 3
packet in 4 0a:02:56:04:a9:13 33:33:00:00:00:fb 3
packet in 2 42:68:25:aa:25:9d 33:33:00:00:00:fb 2
packet in 3 42:68:25:aa:25:9d 33:33:00:00:00:fb 3
packet in 1 42:68:25:aa:25:9d 33:33:00:00:00:fb 1
packet in 5 42:68:25:aa:25:9d 33:33:00:00:00:fb 3
packet in 6 42:68:25:aa:25:9d 33:33:00:00:00:fb 3
packet in 7 42:68:25:aa:25:9d 33:33:00:00:00:fb 3
packet in 5 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 2
packet in 6 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 3
packet in 1 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 2
packet in 2 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 3
packet in 3 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 3
packet in 4 6a:e8:d5:eb:0a:cf 33:33:00:00:00:fb 3
packet in 2 a6:77:38:ae:09:0f 33:33:00:00:00:fb 1
packet in 1 a6:77:38:ae:09:0f 33:33:00:00:00:fb 1
packet in 4 a6:77:38:ae:09:0f 33:33:00:00:00:fb 3
packet in 5 a6:77:38:ae:09:0f 33:33:00:00:00:fb 3
packet in 6 a6:77:38:ae:09:0f 33:33:00:00:00:fb 3
packet in 7 a6:77:38:ae:09:0f 33:33:00:00:00:fb 3
packet in 7 ae:61:ff:e4:70:f7 33:33:00:00:00:fb 3
(429) accepted ('127.0.0.1', 46244)
unchanged
success
127.0.0.1 - [23/Sep/2017 18:03:49] "POST /qos/queue/000000000000007 HTTP/1.1" 200 280 0,953228
(429) accepted ('127.0.0.1', 46250)
unchanged
success
127.0.0.1 - [23/Sep/2017 18:04:05] "POST /qos/queue/000000000000007 HTTP/1.1" 200 280 0,763679
(429) accepted ('127.0.0.1', 46256)
127.0.0.1 - [23/Sep/2017 18:04:21] "GET /qos/queue/all HTTP/1.1" 200 1006 0,012896
(429) accepted ('127.0.0.1', 46259)
127.0.0.1 - [23/Sep/2017 18:05:16] "POST /qos/rules/all HTTP/1.1" 200 907 0,028931

```

Policy setting

```

:8080/v1.0/conf/switches/000000000000003/qvsdb_addr
root@simon-VirtualBox:~# curl -X PUT -d '{"top:127.0.0.1:16632"}' http://localhost
:8080/v1.0/conf/switches/000000000000004/ovsdb_addr
root@simon-VirtualBox:~# curl -X PUT -d '{"top:127.0.0.1:16632"}' http://localhost
:8080/v1.0/conf/switches/000000000000005/ovsdb_addr
root@simon-VirtualBox:~# curl -X PUT -d '{"top:127.0.0.1:16632"}' http://localhost
:8080/v1.0/conf/switches/000000000000006/ovsdb_addr
root@simon-VirtualBox:~# curl -X PUT -d '{"top:127.0.0.1:16632"}' http://localhost
:8080/v1.0/conf/switches/000000000000007/ovsdb_addr
root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X GET http://localhost:8080/qos/queue/all
[{"switch_id": "000000000000001", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000002", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000003", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000004", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000005", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000006", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000007", "command_result": {"result": "failure", "details": "Queue is not exists."}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X POST -d '{"port_name": "s5-eth2", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "384000"}, {"min_rate": "600000"}]}' http://localhost:8080/qos/queue/000000000000005
[{"switch_id": "000000000000005", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X POST -d '{"port_name": "s5-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "384000"}, {"min_rate": "600000"}]}' http://localhost:8080/qos/queue/000000000000005
[{"switch_id": "000000000000005", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X POST -d '{"port_name": "s7-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "384000"}, {"min_rate": "600000"}]}' http://localhost:8080/qos/queue/000000000000007
[{"switch_id": "000000000000007", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X POST -d '{"port_name": "s7-eth2", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "384000"}, {"min_rate": "600000"}]}' http://localhost:8080/qos/queue/000000000000007
[{"switch_id": "000000000000007", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X GET https://localhost:8080/qos/rules/all
[{"switch_id": "000000000000001", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000002", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000003", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000004", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000005", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}, {"switch_id": "000000000000006", "command_result": {"result": "failure", "details": "Queue is not exists."}, {"switch_id": "000000000000007", "command_result": {"result": "success", "details": {"0": {"config": {"max_rate": "384000"}}, "1": {"config": {"min_rate": "600000"}}}}]root@simon-VirtualBox:~#
root@simon-VirtualBox:~# curl -X POST -d '{"match":{"nw_dst":"10.0.0.8"}, "m_proto": "UDP", "tp_dst": "5001", "actions":{"qos": "1"}}' http://localhost:8080/qos/rules/all
[{"switch_id": "000000000000001", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000002", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000003", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000004", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000005", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000006", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}], {"switch_id": "000000000000007", "command_result": [{"result": "success", "details": "QoS added, ; qos_id=1"}]}]root@simon-VirtualBox:~#

```

Hosts

The screenshot shows an XTerm window with a dark background and a vertical toolbar on the left. The terminal output is split into two panes. The left pane shows the output of an 'iperf' test from host h1 to various other hosts and nodes. The right pane shows the output of an 'iperf' test from host h8 to various other hosts and nodes. The output includes details such as client connections, UDP buffer sizes, transfer rates, and bandwidth.

```
Client connecting to 10.0.0.8, UDP port 5001
Send [ 37] local 10.0.0.2 port 51164 connected with 10.0.0.8 port 5003
[ 37] Interval: Transfer Bandwidth
[ 37] 0.0-20.1 sec 979 KBytes 400 kbits/sec
[ 37] Sent 682 datagrams
[ 37] Server Report:
[ 37] local 10.0.0.2 port 51164 connected with 10.0.0.8 port 5003
[ 37] Interval: Transfer Bandwidth
[ 37] 0.0-20.1 sec 979 KBytes 400 kbits/sec
[ 37] Sent 682 datagrams
[ 37] WARNING: did not receive ack of last datagram after 10 tries.
root@simon-VirtualBox:~#
```

```
[ 37] 0.0-20.0 sec 1.67 MBytes 700 Kbits/sec 3.786 ms 0/ 1192 (0%)
[ 38] local 10.0.0.8 port 5001 connected with 10.0.0.1 port 37120
[ 38]
[ 38] "host: h7"
[ 38] [ 38] 19.0-20.0 sec 23.0 KBytes 188 Kbits/sec 33.511 ms 0/ 16 (0%)
[ 38] [ 38]
[ 38] "host: h6"
[ 37] 28.0-30.0 sec 23.0 KBytes 188 Kbits/sec 33.668 ms 0/ 16 (0%)
[ 38] [ 38] 20.0-21.0 sec 23.0 KBytes 188 Kbits/sec 33.511 ms 0/ 16 (0%)
[ 38] [ 38] "host: h5"
[ 37] 0.0-1.0 sec 53.1 KBytes 435 Kbits/sec 2.829 ms 0/ 37 (0%)
[ 37] 1.0-2.0 sec 45.9 KBytes 376 Kbits/sec 2.213 ms 0/ 32 (0%)
[ 38] [ 38] "Node: h8"
[ 37] 19.0-20.0 sec 14.4 KBytes 118 Kbits/sec 65.094 ms 0/ 10 (0%)
[ 38] [ 38] "Node: h8"
[ 38] [ 38] 1.0-2.0 sec 15.8 KBytes 123 Kbits/sec 47.362 ms 0/ 11 (0%)
[ 38] [ 38] 2.0-3.0 sec 15.8 KBytes 123 Kbits/sec 54.387 ms 0/ 11 (0%)
[ 38] [ 38] 3.0-4.0 sec 14.4 KBytes 118 Kbits/sec 60.532 ms 0/ 10 (0%)
[ 38] [ 38] 4.0-5.0 sec 14.4 KBytes 118 Kbits/sec 62.761 ms 0/ 10 (0%)
[ 38] [ 38] 5.0-6.0 sec 15.8 KBytes 123 Kbits/sec 63.953 ms 0/ 11 (0%)
[ 38] [ 38] 6.0-7.0 sec 15.8 KBytes 123 Kbits/sec 64.526 ms 0/ 11 (0%)
[ 38] [ 38] 7.0-8.0 sec 14.4 KBytes 118 Kbits/sec 64.792 ms 0/ 10 (0%)
[ 38] [ 38] 8.0-9.0 sec 15.8 KBytes 123 Kbits/sec 64.999 ms 0/ 11 (0%)
[ 38] [ 38] 9.0-10.0 sec 14.4 KBytes 118 Kbits/sec 65.054 ms 0/ 10 (0%)
[ 38] [ 38] 10.0-11.0 sec 15.8 KBytes 123 Kbits/sec 65.121 ms 0/ 11 (0%)
[ 38] [ 38] 11.0-12.0 sec 14.4 KBytes 118 Kbits/sec 65.131 ms 0/ 10 (0%)
[ 38] [ 38] 12.0-13.0 sec 15.8 KBytes 123 Kbits/sec 65.091 ms 0/ 11 (0%)
[ 38] [ 38] 13.0-14.0 sec 15.8 KBytes 123 Kbits/sec 65.119 ms 0/ 11 (0%)
[ 38] [ 38] 14.0-15.0 sec 14.4 KBytes 118 Kbits/sec 65.148 ms 0/ 10 (0%)
[ 38] [ 38] 15.0-16.0 sec 15.8 KBytes 123 Kbits/sec 65.069 ms 0/ 11 (0%)
[ 38] [ 38] 16.0-17.0 sec 14.4 KBytes 118 Kbits/sec 65.155 ms 0/ 10 (0%)
[ 38] [ 38] 17.0-18.0 sec 15.8 KBytes 123 Kbits/sec 65.099 ms 0/ 11 (0%)
[ 38] [ 38] 18.0-19.0 sec 15.8 KBytes 123 Kbits/sec 65.102 ms 0/ 11 (0%)
[ 38] [ 38] 19.0-20.0 sec 14.4 KBytes 118 Kbits/sec 65.152 ms 0/ 10 (0%)
[ 38] [ 38] 20.0-21.0 sec 21.5 KBytes 176 Kbits/sec 47.901 ms 0/ 15 (0%)
[ 38] [ 38] 21.0-22.0 sec 21.5 KBytes 176 Kbits/sec 41.563 ms 0/ 15 (0%)
[ 38] [ 38] 22.0-23.0 sec 21.5 KBytes 176 Kbits/sec 39.921 ms 0/ 15 (0%)
[ 38] [ 38] 23.0-24.0 sec 21.5 KBytes 176 Kbits/sec 37.106 ms 0/ 15 (0%)
```


REFERENCES

Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella OpenNF (2015). Enabling Innovation in Network Function Control University of Wisconsin-Madison

Chin may Vaishampayan, Sarvesh Bidkar, Saurabh Mehta, Deval Bhamare, Raviraj Vaishampayan and Ashwin Gumaste. (2015). Demonstrating Open Flow over a Carrier Ethernet Switch Router (CESR) Services Perspective, Bombay, India

Cloud radio retrieved from http://labs.chinamobile.com/cran/wp-content/uploads/CRAN_white_paper_v2_5_EN.pdf

Derrick D'souza, Krishna Prabhu Sundharan, Savithru Lokanath, Vivek Mitta (2016): Improving QoS in a Software-Defined Network, University of Colorado Boulder

Ericsson. (2015). Network functions virtualizations and software management: Retrieved from <http://www.ericsson.com/res/docs/whitepapers/network-functions-virtualization-and-software-management.pdf>

Hassan Hawilo, A. Shami ; M. Mirahmadi ; R. Asal (2014) NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC), Canada, http://archive.openflow.org/wk/index.php/Flow-Service_Classification

ITU, ITU-TE.800 (09/2008) Telecommunication Standardization Sector of ITU, Quality of telecommunication services: concepts, models, objectives and dependability planning – Terms and definitions related to the quality of telecommunication services

Jordi Ferrer Riera, Eduard Escalona, Josep Batall'e, Eduard Grasa, Joan A.(2014) Virtual Network Function Scheduling: Concept and Challenges, Barcelona.

Liwei Kuang, Laurence T. Yang, Xiaokang Wang, Puming Wang and Yaliang Zhao. (2016). A Tensor-Based Big Data Model for QoS Improvement in Software Defined Networks

Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Saete Buriol Marinho Pilla Barcellos, Luciano Paschoal (2015) Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions, Gaspar Institute of Informatics

Mao Yang , Yong Li , Depeng Jin , Lieguang Zeng , Xin Wu , Athanasios V, Vasilakos. (2014.) Software-Defined and Virtualized Future Mobile and Wireless Networks:

Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia (2015) Modeling and Performance Evaluation of an OpenFlow Architecture , University of Würzburg, Institute of Computer Science, Würzburg, Germany

Mininet overview (2017) retrieved from <https://Mininet.org>

M. Yoshida, W. Sheen, T. Kawabata, K. Minato and W. Imajuku. (2015). MORSA: A Multi-objective Resource Scheduling Algorithm for NFV Infrastructure, Japan

Nathalie Omnes, Marc Bouillon, Gael Fromentoux, Olivier Le Grand. (2015) A Programmable and Virtualized Network & IT Infrastructure for the Internet of Things, How Can NFV & SDN Help For Facing The Upcoming Challenges. Orange Labs Lannion, France

Navid Nikaein, Eryk Schiller, Romain Favraud, Kostas Katsalis, Donatos Stavropoulos, Islam Alyafawi, Zhongliang Zhao, Torsten Braun, Thanasis Korakis (2015): Network Store: Exploring Slicing in Future 5G Networks, Paris, France

Nick McKeown, Tom Anderson, Hari Balakrishnan , Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner. (2008) Openflow: Enabling innovation in campus networks.

NFV ISG Published Documents: (2014) Retrieved from <http://www.etsi.org/technologies-clusters/technologies/nfv>

Open networking foundation (2012). Software-Defined Networking: The New Norm for Networks

Open daylight, opendaylight resources, viewed April 2016, <<https://www.opendaylight.org/>>

OPNF, SDN definition, viewed April 2016, < <https://www.opennetworking.org/sdn-resources/sdn-definition>>

Open stack, viewed may 2016, <<https://www.openstack.org>>

Production Quality, Multilayer Open Virtual Switch (2017) retrieved from <http://openvswitch.org/>

OPNFV, NFV platform, viewed April 2016, <<https://www.opnfv.org>>

Pål Grønsund, Kashif Mahmood, Geir Millstein Ariel Noy, Gadi Solomon, Ajay Sahai(2015) , A solution for SGI-LAN Services Virtualization using NFV and SDN Norway

Roy Thomas Fielding(2000): Architectural Styles and the Design of Network-based Software Architectures, California, US.

Wenyu Sheen, Masahiro Yoshida, Taichi Kawabata, Kenji Minato, Wataru Imajuku. (2015) Conductor: An NFV Management Solution for Realizing End-to-End Virtual Network Services, Kanagawa Japan.

What is RYU? (2017) retrived from . <https://osrg.github.io/ryu/>