TESTING THE FUNCTIONALITY AND EFFECTIVENESS OF SOFTWARE DEFINED NETWORKS

by

Adebayo Oluwaseun Adedayo

Thesis submitted in partial fulfilment of the requirements for the degree
Master of Engineering (Magister Ingeneriae)

in

Electrical and Electronic Engineering Science
Faculty of Engineering and Built Environment

at the

University of Johannesburg

Supervisor: Prof Bhekisipho Twala

Co-Supervisor: Dr Nnamdi Nwulu

December 2017

DECLARATION OF AUTHORSHIP

I, Adebayo Oluwaseun Adedayo, declare that this thesis titled, 'Testing the Functionality and Effectiveness of Software Defined Networks and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at the University of Johannesburg.
- This work has not been submitted anywhere else or by anyone else for academic purposes.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work.

Signed:____________________

Date: 18/06/2018

ABSTRACT

An important part of Information Technology is networking which has made communication between two or more computers or devices to be possible. Traditional network architecture is not able to meet the challenges of the current trends in networking due to the complexity of handling devices and its non-scalable nature. Software Defined Networking (SDN) is an emerging dynamic approach to networking that makes use of logically centralized controllers in managing a network thus simplifying network design and operation. The project involves the design of a prototype network based on SDN architecture. An analysis of the network is conducted by considering various aspects of the Software Defined Networks that affects its functionality such as transfer of data between the control plane and the data plane. Furthermore, we analyse the use of virtualization technology, troubleshooting and verification of the behaviour of SDN. Since SDN is a new networking approach, there are various aspects of the technology that still needs to be understood and improved. The aim of this research is to test the functionality and effectiveness of SDN and to investigate various aspects of the architecture that affects its operation. Recommendations and conclusions emerging from the analysis are made to enhance the understanding and functionality of SDN.

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof Bhekisipho Twala for his continuous support, motivation, knowledge and patience. My appreciation also goes to Dr Nnamdi Nwulu for his support and encouragement during the last few months of completing this thesis.

I would like to convey my gratitude to the National Research Foundation (NRF) for scholarship provided during my research.

I thank my colleagues Mokesioluwa Fanoro and Vitalis Aguba for their support and productive discussions.

Lastly, I would like to thank my parent and siblings for their support and my profound gratitude to my lovely, understanding and patient wife Oluwasolami, my queen whom have been of great help, support and relentless effort in every aspect and also my wonderful gifts from God Inioluwa and Josiah for being part of the success. All the glory goes to God Almighty for making this research a reality and for giving me support all round, Lord without you this will not be possible.

DEDICATION

I would like to dedicate this thesis to God Almighty who save me and gives me the grace to complete this degree;

My parent and sibling;

My lovely wife and wonderful children for their support.

TABLE OF CONTENTS

KEYWORDS

Software Defined Networks (SDN), OpenFlow, Mininet, OpenvSwitch, Ryu, Virtualization.

ABBREVIATIONS AND NOTATIONS

API	Application Programming Interface
ARP	Address Resolution Protocol
CAPEX	CAPital EXpenditure
CLI	Command Line Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Control Protocol
DNS	Domain Name System
DoS	Denial of Service
DPID	DataPath Identifier
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IT	Information Technology
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JSON	JavaScript Object Notation
KVM	Kernel based Virtual Machine
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
MAC	Medium Access Control
MPLS	Multiple Protocol Label Switching
NAT	Network Address Transaction
OFP	OpenFlow Protocol
ONF	Open Networking Foundation

OSI	Open System Interconnection
OPEX	Operating EXpenses
OSPF	Open Shortest Path First
PC	Personal Computer
QoS	Quality of Service
REST	REpresentational State Transfer
RIP	Routing Information Protocol
SAN	Storage Area Network
SDN	Software Defined Networks
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wireless Area Network

1. INTRODUCTION

The explosion of connected devices on the network has led to network expansion which has led to complexity in managing today's network environment. Devices such as smartphones, tablets, laptops and desktop computers in businesses, education sectors and industries are increasing everyday which contributes to increase in bandwidth consumption. Management and configuration complexity are also experienced by the system and network administrator's due to increase in the number of switches and routers in the network which are the result of exponential growth of internet usage.

These problems were partially solved with the invention of virtualization and using load balancer across the servers to control the load and bandwidth usage. Server virtualization and cloud computing are the latest technologies used in datacentres which process a lot of data per second. Some of the organizations with large data centres include Google, Apple, Facebook, E-bay and Amazon. These data centres process and store large amount of data that rely on speed and performance. Lack of consistency and dependency on various vendors still affect the quality of server virtualization and cloud computing.

To meet the current demand and plan for future expansion, new and improved network architecture is needed. This should address the complexity, scalability and consistency problems in today and future networks. It will also need to be easily managed and easily adaptable to changes in network topology, protocols and traffic control. The new networks need to be vendor-independent which will enhance better network management and facilitate dynamic control of the network.

In lieu of this, a new concept was developed in networking named Software Defined Networking (SDN). A Software Defined Network is an emerging norm in networking that promises to address issues affecting the traditional network architecture and enable innovation in network management and design [1]. SDN is also a new network architecture that separates the control plane from the data plane making the split control plane to be programmable. To achieve this, the control plane inside the networking device such as switches and routers are separated from the device and is logically centralised and controlled.

This logically centralised control plane is called the controller. The controller consists of a software that controls multiple data-plane, and is responsible for traffic forwarding from the data plane of networking devices in the network and uses a standard protocol like OpenFlow. One of the most important goals of SDN is to control all the networking devices from a logically centralised controller without the need of controlling each device [1], [2].

The controller controls the state of the data plane using an Application Programming Interface (API) such as OpenFlow. OpenFlow is the first standard protocol for SDN architecture and gives a straight forward access to the forwarding plane. It also administers information on the data plane networking devices [3]. OpenFlow was created at Stanford University and aimed to replace layer 2 and layer 3 protocols on the OSI model [4]. OpenFlow protocol serves as a bridge communication between the data plane and the logical centralised controller using a secured channel connection [5]. Open Networking Foundation (ONF) was created by Google, Facebook, Microsoft, Verizon, Yahoo and Deutche Telekom in 2011 to develop SDN and the use of OpenFlow architecture in networking.

The foundation operates as a non-profit organization and it maintains OpenFlow standard. The organization has been upgrading the specification of OpenFlow to add more functions and better performances which will be effective in the software switches and hardware switches that are OpenFlow compatible. The controller makes use of OpenFlow protocol to discover OpenFlow enabled switch, their topology, collect statistics from the networking device and make forwarding decision for the data plane.

1.1 Motivation

In testing the functionality and effectiveness of SDN as the new paradigm in networking, some of the supported techniques and methodology on the traditional architecture will be examined. The motivation behind this research is to test if SDN can be deployed in today's networking environment. Configuration of different networking devices from different vendors have always been a tedious task today, likewise, the management of such networking devices have been cumbersome as well. This research is to test the functionality and effectiveness of SDN and its usefulness to overcome the tedious task, complex framework management, expensive capital expenditure and operating expenditure faced using traditional network architecture. This is done by running some experiments to check some networking scenario by analysing their functionality and effectiveness.

Exponential growth of connected devices such as smartphones, tablets, laptop and desktop computers to sites like Google, Facebook, Amazon, Netflix, Apple, Twitter, Instagram etc. has given rise to constant upgrading of data-centres been used by such firms to accommodate the load on their servers. This also gives rise to Big-data which evolves from the enormous amount of processed and stored data been handled daily by data centres. Data centres need to handle very high-speed connections across its networking device and distribute the load evenly across the servers.

Data centres need a new way to network their devices across the networks and to make the management of such devices easier without losing security, functionality and effectiveness of the current network technologies. This research will also show how data centres can benefit from SDN in the techniques used to network their devices and in managing deployed virtual machines instances in the network. As more data are uploaded and downloaded every day, with new applications coming up, cloud computing has helped developers to deploy their applications in the cloud and encourage users to connect to websites running such applications. Cloud computing in data centres has contributed immensely to IT development and SDN have been invented to make their services better.

Considering the number of connected devices in campuses in different higher institutions of learning today, managing the devices on the networks are difficult and cumbersome and increases the operating expenditure of the schools. Each node on the network can hardly be monitored due to limited overview of networking devices and nodes on the campus network. Network administrators will need to implement different policies and protocol services to monitor and manage their networks. This comes at a cost to the schools and the objectives are not attained some of the time.

A test-bed laboratory will be built on the laptop for testing the functionality and effectiveness of SDN. Some experiments will be conducted to examine the objectives of this research on a LAN with same subnet, VLAN, VLAN interconnectivity, QoS, traffic monitoring, and the application of SDN in data centres. Though additional features are added to OpenFlow protocol to enhance the use of SDN in networks, current features will enable dynamic IT-level architectures, for example, multi-tenancy in Cloud computing organization [6].

1.2 Importance of SDN

SDN is important to many organisations and academic institutions depending on the sub-sector of the institution which will define the method of deployment to be used. Various network spheres such as wireless access networks, enterprises, ISP, home, small business and data centres can adapt SDN technology into their system [7], [8]. The use of OpenFlow protocol in SDN makes it easier for SDN enabled networking device to interface with traditional networking devices.

Software Defined Networking is cheaper to run and it's cost effective. Organizations with large data centres will benefit more from the cost effectiveness. This is because only one protocol is to be used to control flows in the network which makes the network to be easily managed. New services and policies deployment are faster in SDN than in traditional network architecture and this is a great importance of SDN to organizations.

SDN offers a simple and easy to use service management for network monitoring, QoS, firewall and other services. When we consider an IT organization, the administrator in a lower level can easily understand the importance and use of SDN on a network, and can also adapt to implementation of the technology in a network environment. Ability to view all the networking devices in the network with the use of API in SDN is another importance that makes SDN a technology to be embraced as it offers a better a network environment.

1.2.1 Benefits of SDN to Higher Institutions

SDN started as a university research centre where research was being conducted to find a new way of testing new services and policies without the need to shut down the networking devices and to master or use all the protocols in networking. One of the first benefits of SDN to higher institutions of learning is the provision of a new research field in networking. Universities have one of the highest connected devices in the network which is growing higher as more students enrol.

Managing such a big network has not been easy because traditional network is difficult to manage, and very costly to run.

SDN will benefit Universities by making it easier to configure all the networking devices on the network. SDN also make network management easier than traditional network architecture, and without the need for specialized proprietary hardware to be used in the network. Higher institutions will also benefit from SDN with the reduction of capital expenditure (CapEx) incurred to purchase new networking devices which are expensive, the use of OpenFlow enabled devices in the network will reduce CapEx because these devices are not proprietary and are cheaper than the proprietary systems [9].

Deployment of new services on the school data centres can be virtualised and provided to the users whenever the services are required. Virtualization of services on the servers will eliminate the need for Universities to purchase new hardware to install the required services. These services can be first tested on virtual machines and deployed to the virtual servers. The load across the virtualized servers can then be balanced using OpenFlow protocols [10].

SDN gives flexibility and innovations to the University's network manager to explore different applications of SDN to enhance better service on the network and easier management of departmental local area network. For instance, faculty administration and finance departments can make use of virtual networking in SDN without the need for VLAN which will involve the configuration of each networking devices in the respective departments. SDN will also enable the network and system administrator to have a full overview of the whole University network which may comprise of one or more campuses with a city across different geographical locations. This will make management of devices on the network less cumbersome and new network capabilities can be easily deployed. SDN will also increase uptime and reduce downtime, it will also improve planning because of using centralised overview of the entire network.

1.2.2 Benefits of SDN to Industries

Industrial sectors that benefit most from SDN are Cloud computing service providers with services such as Amazon Elastic Computer Cloud (EC2), Windows Azure, Google App Engine and Rackspace Cloud servers [11], [12]. They provide infrastructure for service providers which in turn provide services to end users by building different services which include application, hosting, content delivery, search engine etc. This is achieved using multiple virtual machines created which holds services to be deployed to end-users [10]. Apart from the Cloud computing giant named above, every organisation with internet facility can benefit from SDN in different ways ranging from savings on their expenditure to simplicity in running their services. Industries can benefit immensely from SDN in a variety of ways, some of which are listed below:

- Complexity Reduction: SDN reduces the complexity associated with network configuration and management using the traditional network architecture. SDN reduces complexity by offering a flexible network management framework. A framework can be designed to automate many management tasks that are done manually in today's network. The use of automated network in the network will reduce operational overheads, decrease network instability and makes SDN a number one choice in managing Cloud-based application [3].

- Low Capital Expenditure: SDN with the use of OpenFlow enabled networking devices will lower the capital expenditure (CapEx) in an organisation by making use of available equipment and only purchase required OpenFlow enabled devices. Using vendor-neutral devices are cheaper and easier to maintain than vendor-dependent devices because the producers of the former are increasing these days which is one of the reasons responsible for the low prices. The use of virtualization also lowers the CapEx to be incurred in purchasing new devices. Virtual networking of switches and servers can be used with SDN to save cost of purchasing new network devices [13].
- Lower Operating Expenditures: The survey performed with network administrators, Operating Expenditures (OpEx) has a higher impact on the organisational cost than CapEx. There is lower operating cost due to improved network management with the use of logically centralised control management. SDN reduces the need to hire highly specialised skilled engineers to maintain and manage the network, the less complex framework used in SDN will only require lesser highly skilled engineers to manage the network [13].
- Centralised control of multi-vendor environments: SDN controller software makes the management of the entire network to be easier by providing tools for faster configuration and updating of devices on the network. SDN makes this to be easily achievable by controlling any vendor OpenFlow enabled networking device [3].
- Policies are faster and easier to deploy: Deployment of policies is faster and easier in SDN because of using virtualised networking environment. Network policies can be tested easily and later deployed after achieving the desired goal or can be deploy as at when needed by making use of automatic framework that can be used with SDN.
- Higher rates of Innovation: SDN adoption by IT industries and other organisations will give rise to reprogramming of the network in real time to meet specific needs. SDN makes use of virtualization to achieve most of its purpose, the use of virtualization in SDN will increase the eagerness of engineers to explore different applications of SDN and to investigate new services, deploy new services and new network capabilities using SDN [14].
- Virtualize Network Environment: The use of SDN in virtualization of networking device on the networking and data centres has helped IT department and organisation to reduce the capital expenditure. First, the need to purchase new device has been drastically reduced. Secondly, the time needed to configure each physical networking device has been reduced because virtual networking device and virtual machines are used to perform the same operations needed and its deployment is even faster than physical machines [14].
- Downtime reduction: Downtime will be drastically reduced with the use of SDN. This can be achieved from the virtualization of networking device which reduces the time for

upgrade of such devices. The use of snapshot of device configuration in SDN will help system administrators to quickly recover devices back to previous state in case there is a failure after upgrading [14].

- Multi-tenancy and Isolation: Servers and networking devices can be isolated from one another in an SDN environment without the need for commonly used VLAN. Servers, applications and networking device can be virtualised and deployed within minutes saving time and capital expenditures. SDN can use latest technology to virtualise and isolate each of these virtualised services on the network.

1.3 Economic Impacts of SDN

The economic impacts of SDN also need to be reviewed to have knowledge ahead before adopting SDN into the networking environment. Considering demand and supply of networking devices, SDN will offer more benefits to the economy. Some of which are listed below:

- As demand for OpenFlow enabled networking devices are increasing, so also is the supply which in turn makes the networking device vendors to supply more to their distributors. Using the law of demand and supply, this will generate more profits for networking device vendors.
- For the consumers, SDN based devices will come at a cheaper rate than the proprietary traditional networking devices. This will make consumers to purchase required devices needed at a cheaper rate. It will also give room to homes and smaller offices to use a standard OpenFlow enabled devices, the standard which they cannot afford before.
- There will be more markets for non-popular vendors and new vendors will also emerge to take their share of the market as well. These vendors may target personal use and small organizations to sell their product and even provide assistants in setting up the devices.
- Lower price of OpenFlow enabled SDN devices will help large organisations to save more on their CapEx which will be a huge savings considering replacement of devices or upgrading of their devices.
- Savings on OpEx will also be huge because SDN will allow organisations to deploy policy faster, shorten development and test cycles, and improve operational efficiency.
- Another economic impact of SDN is the reduction of energy consumption which means huge savings for large data centres. Power management feature can be implemented in data centres to lower power consumption during off-peak periods.

1.4 Problem Statement

The main objective of this research is to test the functionality and effectiveness of Software Defined Networks. Functionality means quality of being functional or quality of being suited to serve a purpose well, while effectiveness means doing the right thing or the capability of producing the desired result. In other words, this research is testing how functional SDN is and the capability of producing desired results of computer network communication using SDN. These desired results include the use of SDN in virtual environment to emulate physical network environment that will be used to perform different functionalities testing on the network using multiple scenarios.

The research also involves the use of OpenFlow protocols which is the protocol used in SDN for communication among the network devices on the network. OpenFlow protocol use OpenFlow table which consist of flow entries that are used for forwarding decision making whenever there is a match on the flow table. OpenFlow works by encapsulating other protocols such as ARP and ICMP. Forwarding of packets with the use of OpenFlow tables will be analysed. This will entail the use of Linux commands and applications to achieve the desired goal. To have a good network environment, some network functionality should be in place; some of this functionality will be examined. These include:

- Virtual Local Area Network: VLAN is used in schools, organisations and industries to segregate the network and to reduce congestion on the network by creating smaller networks with lesser congestion on the network
- Firewall: Firewall is used to secure the network to avoid unnecessary guest, hackers or intruders on the network
- Quality of Service (QoS): QoS is used in networking to manage the network resources. The research will run experiment on how a good quality of service can be implemented in the network to manage the network resources better.
- Traffic management: This include traffic monitoring and measurement. Traffic management provides a way of monitoring and measuring the traffic in the network to provide better services. This can be used in conjunction with QoS to provide a better service.

With the use of the available resources and techniques, this thesis will lay a foundation upon which future researches in SDN can be and shows how various techniques can be used in testing the functionality and effectiveness of SDN to provide a better service in future network

1.5 Scope of the Research

The scope of this research will be on LAN, although there are other possible aspects of networking all other ways or approach are not the focus of this research. The focus of this research is to test the use of SDN technology in networking by analysing the functionality in Local Area Network environment as well as its effectiveness.

Software Defined Networks is becoming the future technology in networking which most of the network vendors, telecommunication companies, ISP, search engines, data centres and cloud service providers are adopting. SDN is aiming to be the main technology in networking in organisations network which include server networking, network across physical and virtual networking devices and in data centres.

The functionality and effectiveness of SDN in a LAN environment will be explored. Although it is possible to adapt this testing to other forms of network environment, this possibility is not investigated in this thesis. This thesis will consider the use of some applications and operating systems to perform the required experiments. Other applications that can be used will not be discussed in detail.

1.6 Methodology

To achieve the objectives of this research, focus will be given to understanding of SDN and how a Local Area Network (LAN) based on SDN can be developed. This involves the dedicated survey of the available literature on SDN. An overview concept will also be presented to highlight the root of SDN.

It is highly essential to know the importance of network monitoring, Quality of Service (QoS) and Address Resolution Protocol (ARP). After going through different literature and background study on SDN, some suggested applications and techniques that are easier to deploy are implemented.

A prototype network based on SDN architecture will be designed for evaluating the functionality and the effectiveness of the network with focus on the operation of SDN on a LAN environment. The prototype network will make use of OpenFlow protocol since this is the first and the most used and readily available protocol for now. The implementation of the prototype will serve as a test bed for the research and provide usage of the network and the programmability of the network. The RYU controller will be used as the main controller which will also serve as the brain of the network. RYU is chosen because it is written in Python programming language and it is easier to deploy and readily available. More details are shown in the first section of Chapter four which gives more into materials, applications and software that are used.

1.7 List of Publications

1. A. O. Adedayo, B. Twala, "Testing the Functionality of Firewall in Software-Defined Networking," Artificial Intelligence and Evolutionary Computations in Engineering Systems, Singapore, 2018, pp. 1-4. doi: 10.1007/978-981-10-7868-2_1
2. A. O. Adedayo, B. Twala, "QoS functionality in software defined network," *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, 2017, pp. 693-699. doi: 10.1109/ICTC.2017.8191068

1.8 Structure of the thesis

This thesis is structured into six chapters with references of the literatures cited in the thesis. Below are the details of the chapters.

Chapter 1 briefly introduces the thesis and outlines the research. It also includes the problem statement that describe the main objectives of the research and methodology that talk about the approach used in solving the problem.

A survey of current works relating to computer network, including some key fundamentals in networking is given in chapter 2. Background works on SDN is also shown reflecting the history of SDN, benefit of SDN over conventional networks and other details.

Chapter 3 presents OpenFlow protocol as the major protocol used in SDN. OpenFlow protocol background and its effect, as well as support in SDN are discussed in detail. These include versions created and the current supported version in this thesis is being discussed as well.

The prototype network testbed which involve the use of different network scenario to test the functionality and effectiveness of SDN is shown in chapter 4. It describes experiments that test major network functions that are available on current network which include flooding, pinging, Address Resolution Protocol (ARP), forwarding, and VLAN implementation in SDN.

Chapter 5 is the continuation of the research that shows more depth into advanced functionality of SDN and their effectiveness with experiments on network which include firewall and Quality of Service (QoS).

Chapter 6 shows SDN monitoring and visualization with webGUI. This involves monitoring with OpenFlow and statistical method.

The conclusion and future works identified during the research is given in chapter 6.

The references are in the last sections that contain details of information used.

2. LITERATURE REVIEW

2.1 Networking

A computer network can be defined as a set of connected devices or nodes which can be a computer, laptop, printer, mobile devices or other devices that can send and receive data on the network. Networking can be the connection of two or more devices together with the ability to communicate with each other [15].

Computer networks can be categorized mainly according to the geographical area they cover. The two main categories are Local Area Network (LAN) which covers a few kilometres and Wide Area Network (WAN) that span across countries and continents. There is also Metropolitan Area Network (MAN) span tens of kilometres and the network size is in between LAN and MAN.

Local Area Networks (LANs) are usually privately-owned networks within a single building or campus of up to few kilometres in size. LANs are used in the connection of personal computer and workstation in office and factories to share resources (such as printers and scanners) and to exchange information [16]. Metropolitan Area Network (MAN) covers tens of kilometre or a city. An example of MAN is cable television network service available in many cities. Wide Area Network (WAN) spans across large geographical area, these may be countries or continents. It may consist of two or more LANs connected.

The connection of interconnected networks is referred to as INTERNET. Earlier before the 1970s, network device manufacturer had their own communication standard on their devices. This makes devices made by other vendor incompatible, thus leaving a big problem in the communication of devices in the network. The lack of communication standard among various network devices has led to the creation of Open System Interconnection (OSI) model in the late 1970s. The model was created as a standard for vendors to make all network devices compatible with each other.

2.2 The OSI Model

The International Standards Organisation (ISO) created a reference model to be used by all network vendors and to serve as a communication stack, and it's called Open Systems Interconnection (OSI) [17]. OSI model was created in the late 1970 as a standard that covers every area of network communication and it is designed as a network architecture that is robust, interoperable and flexible [15]. Its main goal is to create an avenue for vendors to manufacture interoperable network devices and make use of protocols that can make communication between the networking devices possible [17].

It is a layered structure that shows the process of communication of data and network information between application on one computer system through network media and application on another system [17].

The OSI has seven layered structure namely:

- Application layer (layer 7)
- Presentation layer (layer 6)
- Session layer (layer 5)
- Transport layer (layer 4)
- Network layer (layer 3)
- Data link layer (layer 2)
- Physical layer (layer 1)

A summary of each layer is listed below:

- Physical layer – The physical layer moves bits between devices. It also specifies voltage, wire speed and pin-out cables.
- Data link layer – The data link layer transforms the physical layer raw transmission to a reliable link. It combines packets into bytes and bytes into frames. It also provides access to media using MAC address. It performs error detection and not correction [15], [17].
- Network layer – The network is responsible for the source-to-destination delivery of packet across multiple networks. It manages device addressing, track the location of devices on the network and determines the best way to move data [17].
- Transport layer – Transport layer provides reliable or unreliable delivery. It also performs error correction before retransmitting. TCP and UDP are protocols at the transport layer.
- Session layer – Session layer provides dialogue control between devices or nodes and basically keeps different applications data separate. It is also responsible for setting up, managing and then tearing down sessions between presentation layer entries [17].
- Presentation layer – Presentation layer prevents data to the application layer and is also responsible for data encryption compression and translation services.
- Application layer – Application layer enables the user to communicate to the computer. It provides user interface and support for services such as electronic mail, remote file access and transfer and other forms of information services [15].

Some key technologies and management techniques used in networking which include virtualization, VLAN, NFV and QoS are discussed below.

2.3 Virtualization

Virtualization is the process of creating a virtual, rather than physical version of something; this may include operating systems, servers, networks, applications and storage devices [18]. IBM implemented virtualization to partition their mainframe computers in the 1960s to save cost of buying more hardware. Virtualization was more popular in the 1980s and 1990s but uses declines due to the lower cost of hardware available at the time. However, virtualization became popular again in the 2000s to reduce cost of hardware and the carbon emission in datacentres and to save wasted processing power in the datacentres.

The major forms of virtualization include:

- Operating system virtualization is the use of virtualization software to allow a piece of hardware or a host with an operating system to run multiple operating system images at the same time [19].
- In server virtualization, the server resources (such as number and identity of physical servers, processors and operating systems) are concealed from server users [19].
- Storage virtualization involves the pooling of physical storage from multiple network storage devices into what appears to be a single storage device that is managed from a central console.
- Network virtualization is a method of combining the available resources in a network by splitting up the available bandwidth into channels, each of which is independent from the others, and each of which can be assigned (or reassigned) to a server of devices in real time. It also allows multiple virtual networks over a physical network. Some of the technologies that use network virtualization include VLAN, VxLAN, NVGRE and Nicira NVP [19], [20], [21].

Virtualization software that is used to execute virtualization simply decouples the software from the hardware by creating an abstract layer known as hypervisor, between the virtual machine and the host operating system. Most popular virtualization software includes Oracle VirtualBox, VMware player and VMware workstation [22].

A Virtual Machine (VM) is an independent virtual computer system with an operating system running inside the virtualization software and uses the available hardware resources of the host machine [18]. Multiple VM can run inside one host using the virtualization software. The hypervisor acts as a controller between the hardware and the virtual machines and it's used to monitor the virtual machines by assigning resources to the virtual machine as at when required [22].

Considering server virtualization, there are three main technologies which are listed below:

- Full virtualization uses a technology that allows different operating systems to be installed on the virtual machine without the need for modification. The installed guest operating system has no knowledge of the host operating system. The hypervisor is used to manage the hardware resources allocated to the guest OS which include CPU instruction, memory and hard-disk. Oracle VMserver and VMware both use this technology.
- Para-virtualization technology is based on the modification of the guest operating system which makes it to be aware that it is running on a virtual machine. The virtual machine is

aware of the hypervisor and collaborates with the hypervisor to reduce processor overhead. Xen is an example of Para-virtualization [19].

- Virtualization at the OS layer works differently; the technology makes the host to run a single OS kernel as its core and exports operating system functionality to each of the guests. To achieve this, the guest must use the same operating system as the host due to OS kernel export functionality between them. This architecture eliminates system calls between layers, which also reduces CPU usage overheads. Oracle Solaris zones, docker, lxc, lxd and virtuozzo are examples of OS-level virtualization [19].

2.3.1 VLAN

Virtual Local Area Network (VLAN) is a logical group of devices or users grouped by function, department or application irrespective of their physical location on the LAN [23]. VLAN is used to create smaller broadcast domain; with the introduction of VLAN, this broadcast domain may be broken into smaller broadcast domain thereby reducing broadcast in the network. This is because each VLAN is treated like separate subnet with one broadcast only to switch ports on the VLAN. Hosts on one VLAN cannot communicate with hosts on another VLAN, for inter-VLAN communication a router is needed to route the traffic among the VLANs. VLAN uses an identifier to recognize each VLAN in each network and uses 12 bits. The maximum usable VLANs on a given Ethernet network are 4094 because of the 12 bits used.

2.3.2 Network Function Virtualization (NFV)

NFV made his first appearance in 2012 and was proposed by European Telecommunications Standards Institute (ESTI). It was formed to address the need for additional hardware appliances whenever a new network service is required. The space to accommodate the appliances and increase cost of power consumption, capital investment toward the possession of the appliance and scarce skills required to incorporate for network operators [24].

In NFV, network functions such as firewalls, routers, load balancers and network address translation which are normally deployed on dedicated physical appliances, will be partially or completely implemented as software. Their functions are virtualized and installed on universal servers [25].

NFV is used to reduce high Capital Expenses (CAPEX) for initiating services provided by the network service provider as well as reduction of Operation Expenses (OPEX) for servicing and maintaining the equipment and power consumption of the network equipment [24], [26]. Some of the benefits of Network Function Virtualization include:

- Lower cost of devices and lower power consumption by devices.
- It helps in reducing cost by providing more better efficient test and integration on the same infrastructure.
- It enables operators to provide support and services to multiple users using multi-tenancy system.

- It makes use of power management features to reduce energy consumption on the servers.
- It enhances operational efficiency of the network and provide robust support for other platforms.

2.4 Quality of Service (QoS)

Quality of Service (QoS) is a set of techniques used to manage network resources. QoS is a technique used to control traffic priority on the network such as managing delay, delay variation, packet loss and bandwidth. QoS policing feature is used to regulate the maximum bandwidth that any user can use. This is useful on networks where some users are using high-speed broadband connection and consume almost all the available bandwidth on the network leaving the user with lesser bandwidth [27], [28].

With the advent of Voice over IP (VOIP) and video conference, QoS has been a useful tool to deliver a better service efficiently on the network. VOIP is a higher priority service on the network that should be void of packet loss, delay, jittering and high latency which will have great effect on the quality of the voice call.

QoS is used to provide optimal performance for a variety of applications and services on the network and it's also useful in giving access to the right host or application to use the required amount of network resources which may be real-time or non-real-time [29]. Some of the ways QoS can be implemented include:

- Policing involves dropping traffic over a specified limit. It uses a logic of ensuring that no traffic exceeds the maximum rate (in bits/sec) that the network administrator configures, which also ensures that no one traffic flow can take over or consume the entire network resource.
- Traffic Shaping is used to match device and link speeds, which controls packet loss, variable delays and link saturation, which can cause jitter and delay [28].
- Priority queuing involves placing a specific class of traffic in the Low Latency Queue (LLQ), which is processed before the standard queue.

2.5 Data Centres

Prior to the advent of data centres regarding features as it is today, organisations make use of servers to perform their computing, storage and networking which is normally kept in the server room. This server can possibly be called data centre considering some of the features it possesses [30]. The server room consists of enterprise desktop PC with higher performance than PC used individual. This server has the functionality to compute and store data being used by the organisation.

Due to the exponential growth of data being processed and stored by today's organisation, the enterprise server used earlier in the server room cannot handle the computing speed and power, the increased data processing and storage, and the highly-demanded network speed that is required to deliver the service needed. Data centres came into existence to handle the required services and demand in today's IT infrastructure.

A data centre can be described as a centralised IT infrastructure that can perform operations and services either in physical form or virtual form for management, dissemination and storage of data and information [30], [31]. Data centres run servers and storage equipment that can run application software, process and store data needed for the organisational needs or needed by other organisations to run their businesses [32]. Data centres were originally created to physically separate traditional computing elements, their storage elements and the networks that interconnect them with the client users [4]. As demand increases, virtualization of the computer and storage section in the data centres come into place to manage available resources effectively and to reduce capital expenditure and operational expenditure.

Data centres can be classified into internet or enterprise. Internet-facing data centres are browser-based and support averagely few applications, it also has many users which are usually unknown, while enterprise (or internal) data centres on the other hand host more applications and services to fewer users [31]. A modern data centre comprises of computer, storage and network resources, and it may exist as a private data centres within an organisation's facility or may be maintained by service providers with specialised facilities such as Amazon and Google that provide services to individuals or organisations in the form of cloud services.

2.5.1 Multitenant Data Centres

Service providers provide multitenant services to their customers. Multitenancy is the terminology used in data centres which means the provision of services such as own (virtual) network to each tenant given them the ability to manage the virtual network in a similar way that they would manage a physical network.

Multitenancy gives the data centres provider the ability to host hundreds or even thousands of tenants or customers on their hardware in the same location or across multiple locations. This is done through virtualization of its resources i.e. a single set of physical hardware can host multiple tenants with access to the server, storage and the network. Virtualization is done using virtual machines that run on the hardware to provide components needed by the tenants and must be within the available resources of the underlying equipment.

2.6 History and the Need for SDN

2.6.1 Traditional Network Architecture

Different networking devices operate at the specific layer of the OSI model. Considering a traditional Local Area Networking (LAN), there are layer 2 devices such as switches and layer 3 devices like routers. These devices have three planes which are data plane, control plane and management plane. Networking equipment vendor has created network operating system or firmware that operates on these devices. This firmware enables the device to perform accordingly on the OSI model level in which it belongs to and thus process the data by analysing the packet header and forward or drop the packet based on the decision made.

A typical networking device consist of data plane, control plane and management plane. The management plane is part of control plane used for network management such as monitoring and configuration.

The control plane is the brain of the device where the decisions are made. It is the firmware developed by network device vendors to process the operations and switching of packets in the network. The control plane is responsible for making sure the information in the forwarding table are current and processing of multiple protocol that may have effect on the forwarding table. Network-wide active topology is also the responsibility of the control protocols to keep which is an important role that cannot be neglect [33].

The data plane is the hardware with various ports which are used for transmission and reception of packets and it also contains a forwarding table in its affiliated logic. The data plane is responsible for packet buffering, packet scheduling, and header modification and forwarding. Figure2.1 below shows the roles of each plane in a network device.

Regarding SDN, data plane and control plane will be our focus. The operation involves the use of a routing information base (RIB) in the control plane as a data set in storing the network topology. The RIB passes the information to the forwarding information base (FIB) which is mirrored between the control and data plane of a typical device [4].

The forwarding table makes use of the information provided by the FIB and uses it in forwarding. As shown in figure 2.1, the relationship between the data plane, the control plane and the management plane has been displayed with key features of each plane. It shows functions of the control plane and the data plane and the use of the management plane in monitoring the status and the statistics of the network using compatible protocols.

The forwarding decision determined as per information in the RIB is communicated internally to the data plane FIB which is used in forwarding of packets. Networking device vendors such as Cisco, HP and Juniper has their own proprietary Operating Systems (OS) running on their devices. This led to rigidity in devices used in network and complexity. This makes the network administrator to learn how to operate and configure different vendor devices on the network. It has also made the introduction of different new device into the network to be cumbersome.

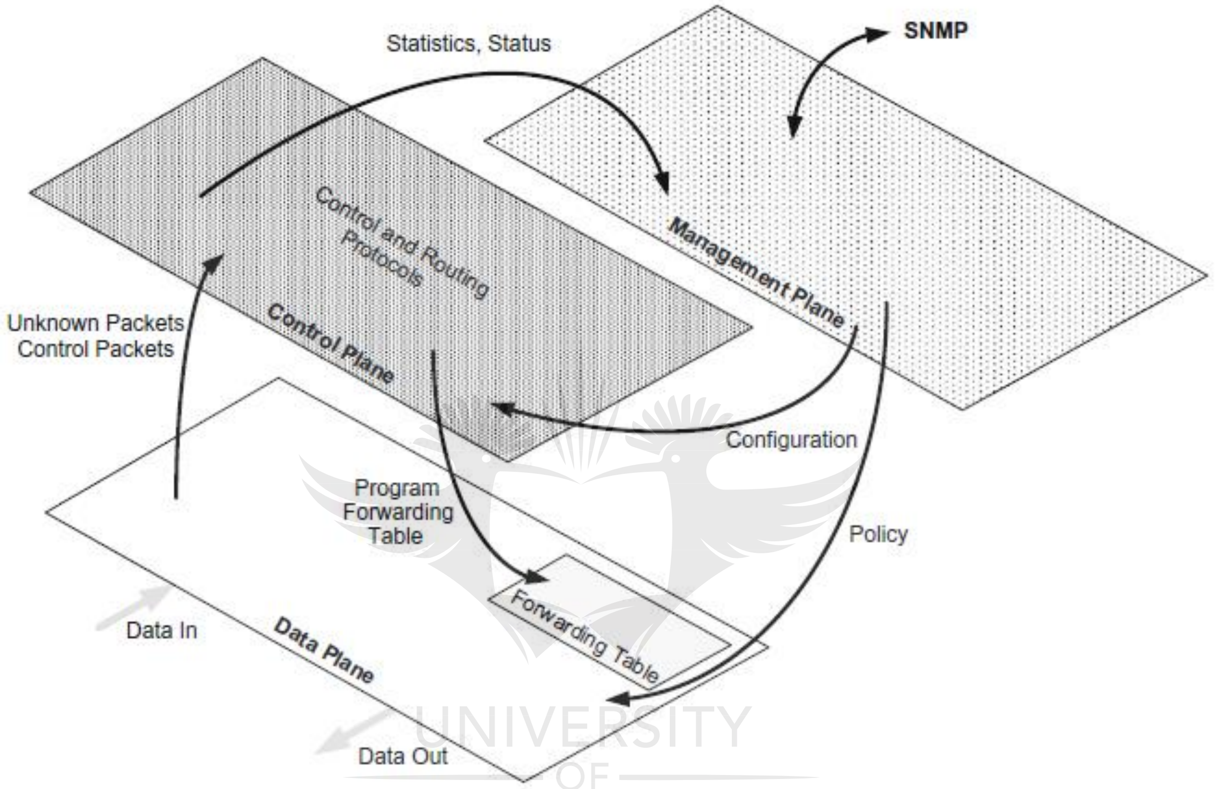


Figure 2.1: Roles of control, management and data plane [33]

2.6.2 Limitations of Traditional Network Architecture

The current network architecture has some limitations which makes it hard to provide solutions to new technology development, increasing connected device and enormous data processed by data centres on daily basis. Data centres are becoming congested as the number of online connected devices are increasing. Administration of network devices in organization and data centres equipment is becoming more tedious because the traditional network architecture is not designed to meet the current requirements by enterprises, organizations and today's users. Some of the limitations of current network include:

- Complexity – The process of adding more devices to the network have become complex and time consuming. There will be need for the network administrator to configure multiple switches, routers, and firewall and update the access control lists and other protocol-based

mechanism. Network topology, software version on the device and vendor switch model must also be taken into consideration, all this has made today's network relatively static whereas the today's server environment is dynamic in nature.
In addition, server virtualization is increasing, with the use of load-balancer to access applications distributed across multiple virtual machines. This has not been an easy task to implement for service providers without considering the static nature of current network. The current network does not conform to the dynamic changing traffic patter which arose from the use of IP voice and data service and different QoS to maintain the reliability of the service.

- Inconsistent policies – In current network technology, implementing a network-wide policy will require the configuration of new devices; bring up new virtual machines and reconfiguration of access control list across the entire network. This complexity has led to inconsistency in the security, QoS and other policies that need to be implemented across the network.
- Inability to scale – To deliver higher value, better services to customers, the network need to be scalable. The use of multi-tenancy technological services to customer also complicate scalability as multi-tenancy is created to serve group of users with different applications and different performance needs. To meet client requirements, some specialized devices at the network edge is needed which thus increases the capital expenditure (CAPEX) and operational expenditure (OPEX).
- Vendor dependence – Deployment of new capabilities and services to adapt to changing business needs has been hindered by different vendor equipment that lacks a standard that can make the device to easily communicate with each other.

A new network architecture is needed to be created that will be simple, independent, with non-proprietary software and be an open standard accepted by all network device vendors. The architecture should also have provision for future expansion without the need for expensive network edge devices. This new architecture is called Software Defined Networking (SDN). The new network architecture is needed to drive the latest computing trends that the old network architecture cannot address. These driving trends include the following [3]:

- Changing traffic pattern
- Rise of cloud service
- Big data

SDN was created as a solution to the limitation of the old network architecture and to address and proffer broader ways to the above latest computing trends.

2.7 Software Defined Network

Software Defined Networking is an emerging network architecture where the control plane is separated from the data plane. SDN makes the network to be programmable and it isolate the control plane from the data plane and provides a logically centralized controller that acts as the brain of the network. The physical abstraction of the control plane from the data plane is a great innovation that gives room for network programming, flow control, forwarding decision and policies to be implemented from the control plane. This also makes each layer to be independent and enable faster, easier and simpler programming of the networking devices in the network [1], [2].

The history of SDN can be traced back to 2003 when ForCES (Forwarding and Control Element Separation) was proposed by two Intel employees H. Khosravi and T. Anderson. The proposition was to separate the forward element (data plane) from the control element (control plane) in the network device [34].

"Clean Slate" program was formed in 2005 in Stanford University in the United States of America. It was formed to investigate if there are any fault in the current network architecture and to make the current network architecture better. Clean Slate concluded that the network should contain a separated plane to make decision in controlling the whole network. Clean Slate proposed a centralised system that will be able to monitor the whole network and make traffic forwarding decision based on the information provided in the control plane tables [35].

The breakthrough that made SDN what it is today started as part of Clean Slate program at Stanford University. The program was called Ethane. Ethane was first implemented in 2006 as a new network architecture that provides a simple powerful management model. Ethane gives the network administrator to set policy rules for different protocols and users. The two biggest advantages of Ethane are the use of very simple hardware switches that are cheap to manufacture and a central controller that makes all the decision to be used in forwarding by the connected switches. Ethane uses a different protocol for communication between the controller and switches, this protocol later became OpenFlow protocol which is the major protocol used by controllers in SDN today [35], [36]. OpenFlow protocol will be further discussed in Chapter 3.

To implement Ethane in a larger network, the distance and location of the controller to the switches must be put into consideration. To address this issue, Software Defined Networking was formed and it uses a logically centralised technology to communicate with the switches. SDN has been designed to make abstraction of the control plane from the data plane easier. ISP and cloud computing firms will benefit a lot from SDN as it has made deployment of services easier, faster and cost effective.

2.7.1 SDN Architecture

The main objective of SDN is to decouple the binding between the software and hardware existing on the same networking device. This is achieved by decoupling the control plane from the data plane, and the control plane is logically centralised in software based SDN controllers. These controllers are used to maintain global view of the entire network. With the use of the controller, the whole network appears as a single logical switch [3].

With SDN, network administrators manipulate traffic from a logically centralised controller by gaining vendor-independent control over the entire network. This will eliminate the need to configure each switch in the network. It also eliminates the need for different protocols in the network as the communications between the controller and switches are simplified using the protocol they both understand.

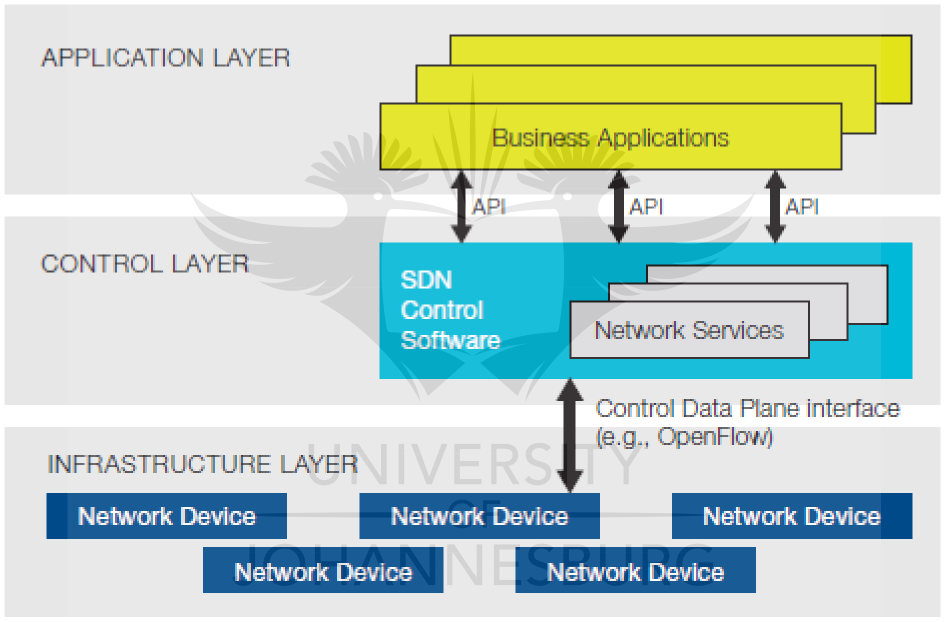


Figure 2.2: SDN Architecture [3]

SDN architecture shown in figure 2.2 basically consists of three layers namely: Application layer, control layer and Infrastructure layer.

The application layer refers to all applications in the network, this include business application, analytical, networking management applications and other SDN applications. The control layer refers to the control plane. The SDN controller resides in this layer. The controller acts as the brain of the network and it connects to the application layer and infrastructure layer. The infrastructure layer is basically the data plane where the network devices are located.

The control layer connects to the application layer via the application programming interface (API). This connection interface is called Northbound API. Northbound API is programmable and is used in abstraction of the network devices and topology. It has not been standardized yet but it is exposed using Representational State Transfer (ReST) which also makes the interaction easier using JavaScript Object Notation (JSON) or eXtensible Markup Language (XML). Some developer also uses Java and Python programming language. The ReST is used as a tool to write set rules that are installed on the networking devices as flows and are used for forwarding, dropping, or analysing the incoming packets [33].

The infrastructure layer refers to the entire physical network infrastructure. The data plane resides in this layer. Network device such as switch that performs packet-forwarding is independent and only receive forwarding decisions regarding processes from the control plane which is logically centralised in the network. The Southbound API connects to the infrastructure layer using the OpenFlow. The OpenFlow protocol is used to program the network devices and it is a standard protocol, unlike the Northbound. The Southbound interface makes multi-vendor communication possible in the network.

SDN brings simplicity into the networking making the need to understand and process different protocols by different networking devices and standards that are no longer needed as all protocols are encapsulated in OpenFlow protocol, networking device thereby receive instructions from the controller for decision making. Logically centralised SDN controllers make it easier to deploy new policy or to alter the network behaviour in real-time. With the use of automated SDN programs in the logically centralised controllers, network administrators are more flexible to configure, manage, secure and optimise network resources. In conclusion, SDN makes use of intelligent co-ordination to simplify the management of the entire network.

2.7.2 SDN Operation

The basic components of SDN are the SDN devices, the controllers and the applications. The SDN devices such as switch contain forwarding rules to make decision on the received incoming packets. This forwarding rule is called flow entry and it is used to match the incoming packet with the available flow entries in the switches. Flow entries are made up of flows which are also data defined by the controller. Flows are set of packets transferred from one network endpoint or sets of endpoints to another endpoint or sets of endpoints [33].

A flow table contains series of flow entries and the action to perform when the packet matching the flow arrives at the SDN device. The process of operation of SDN is illustrated in the figure 2.3. Basically, the switches are controlled by the controllers. When a packet is received by the switch, it checks the flow table for matching in the flow entries. Flow entries may either be installed reactively or proactively. Reactive flows are setup dynamically for each new flow forwarded to the controller by the switch while proactive flows are setup statically earlier before the arrival of the packet by the network administrator [37].

After the switch checks for a match in its flow table, if a match is found, the switch decides to forward the packet as per instructions and actions configured in the flow table. If no match is found, the switch may either drop the packet or pass it to the controller to make decision on the packet, either to drop it or to install a new flow on the switch which will then forward the packet.

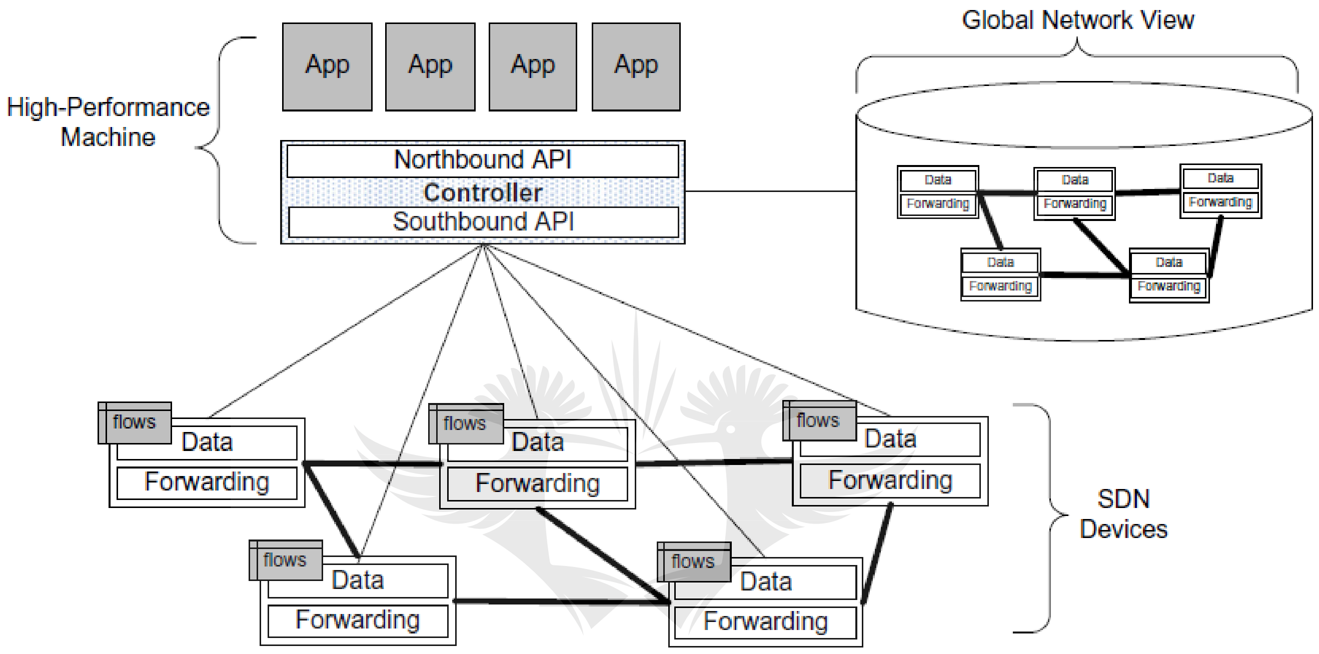


Figure 2.3: SDN Operation [33]

The SDN controller is the brain of the network and it functions by giving right to SDN application to define flow on the network devices. It also helps the application to respond to forwarded packets by the SDN devices to the controller. The SDN applications interface with the controller and are built on top of the controller. It is primarily used to set flows on the device [33]. The process involves the SDN application instructing the controller to respond to the incoming packets by adding new flow entries in the flow table. This will enable the device to respond to the packet next time it sees the packet that belongs to the flow [33], [37].

2.7.3 SDN Controller

The controller is the central nervous system in SDN and it manages all network devices residing in the data plane. It also offers the calling of the network resource service for the applications in the upper application plane which is the northbound API [38]. The controller is used in the implementation of policy decisions, this includes forwarding, redirecting, load balancing and monitoring. Most controllers in use today have some common application modules in-built, this include learning switch, a router, a basic firewall and load balancer [33].

The SDN controller anatomy showed in figure 2.4 shows the controller's core functionality including the northbound API with few applications, the modules and a southbound API. The southbound API uses OpenFlow and it is standardised. It interfaces with the SDN devices. Some other proprietary standards also exist apart from OpenFlow and they all exist on the same controller. The northbound API, on the other hand, has not been standardised which is an issue for the controller-to-application interface in SDN. Some of the major and popularly used API in SDN is REST, Python, and Java [33].

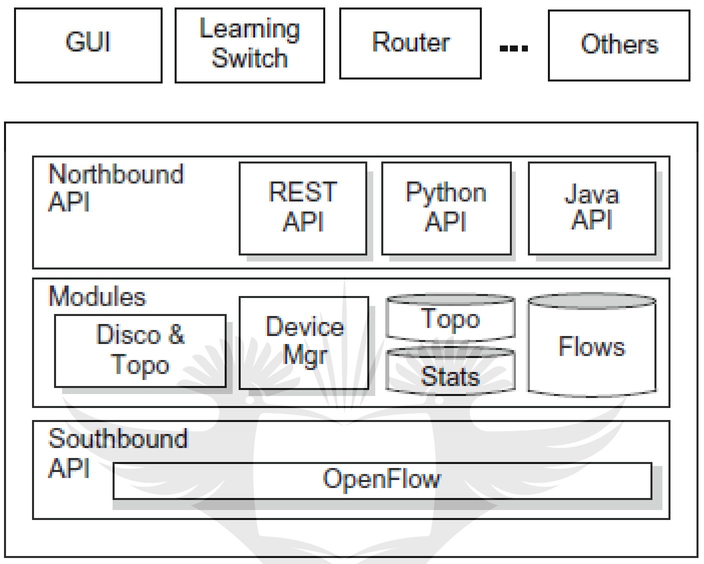


Figure 2.4: SDN Controller Anatomy [33]

2.7.4 SDN Controller Core Module

The SDN controller is the core components of SDN and it is responsible for decision making regarding forwarding actions of the network devices and processing of various information.

The major core features in the controller include:

- End-user device discovering – End user devices such as laptops, desktops, mobile devices and printers are discovered.
- Link discovery module – Controller discover the entire network information which include network devices such as switches, routers and wireless access points [38].
- Network device topology management – The module generates the network topology, store the topology, calculate the forwarding path and monitors the network status [33], [38]
- Flow management – these modules is responsible for maintaining the flow database being managed by the controller. It also coordinates synchronisation of device flow entries with the database [38].

2.7.5 SDN Controller Implementation

One of the earlier problems of SDN and SDN controller design is the controller location. The separation of control plane from the data plane is the strength and major fundamental of SDN and it is also the main controversial issue as well. The separation which is an advantage of SDN over the current network enables the network administrator to control, monitor and manage the entire network by having a full overview of the network [4]. The controller location has effect on the resiliency and high-availability of the network which led to three different types of separation which are explained in full details below:

Centralised Control Plane

The major advantage of centralised control plane is to have a full view of the network and its ability to deploy application faster. The size of the network and location of the controller will be a determining factor in the performance of the network. The listed factors below will affect the performance of centralised control plane:

- Scale – Increase in the network size and volatility will require increase in per-session input-output and processing. Management and data analysis of the network as it grows bigger will also be a great burden.
- High Availability – Perhaps the scale problem can be handled by one controller, but a single controller makes the network to have a single point of failure which will have impact on the network. Therefore, a standby combination of controllers will be required to have a high availability network.
- Geography – The location between the controller and the controlled device must be taken into consideration. City, state or national separation between two will result in transmission and increase the risk of separating the controller from the device.

According to Nadeau [4], it can be deduced that strictly centralised controller will perform very well in research and experimental SDN.

Distributed Control Plane

Distributed control plane uses a model that makes the reconfiguration of the network easier as it does not have a single point of failure but rather incorporate individual element or their proxies to make reachability possible and to have an over view of the entire network [4]. The use of multiple control planes which are interconnected together to have a view of the network will give rise to some problems which are listed below:

- Possible convergence delay and difficulty
- Difficulty in horizontal scaling to add new device and to view the network
- Increasing number of control plane to configure and manage.

The major benefit is the instance of one control plane per device and it is proved to be highly resilient to failure.

Logically-Centralized Control Plane

Due to the problems with strictly centralised and distributed control plane, a conclusion has been reached to balance the merits and demerits of the two to have a better scalable, secure and better performance network. A logically-centralised control plane was implemented. It also has the advantage single configuration points which will then synchronise to all other interconnected controllers in the network. It is much easier to scale horizontally; new instances are only needed to achieve this [4].

The major demerit is that it is also resilient to many failure points, but this only happen during synchronisation which can be limited by using secured and fast convergence. It can be concluded that centralising the control plane in a logically centralised but physically distributed model makes more sense in respect to scale, high-availability and geographical perspective.

2.7.6 Types of Available SDN Controller

Controllers come in different forms based on the programming language used and market focus. Different controllers have emerged since the inception of SDN and OpenFlow. Earlier controller focused on building API support while this was changed recently to include performance enhancement and scalability [39]. Some of the available open-source controllers are:

- NOX: NOX was the first OpenFlow controller created by Nicira networks. It is based on C++ programming language [37], [40].
- POX: POX is a variant of NOX and it is a Python based development platform. The topology can be queried and has support for visualization [40].
- MUL: MUL is a C based multi-threaded controller. It is designed for performance and reliability [40].
- Trema: Trema is a full-stack, easy-to-use framework for developing OpenFlow controllers in Ruby and C. it was created by NEC electronics [37], [41].
- Beacon: Beacon is a Java-based controller created in Rice University and it supports both event-based and threaded operations [37], [40].
- Floodlight: Floodlight is a Java-based OpenFlow controller forked out from Beacon controller. It is an enterprise-class, Apache-licensed and was developed at Stanford University [42].
- Ryu: Ryu is a component based SDN framework developed by NTT in Japan. It provides a well-defined API that allows developers to make new network management and control applications [43].
- OpenDaylight: OpenDaylight was founded by Linux Foundation and is a Java-based controller. It is a highly modular, extensible, scalable and multi-protocol controller built for SDN [44].

Some of the available special purpose controllers that perform specific functions include the following:

- FlowVisor is a network slicer that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controller [40], [45].
- RouteFlow is an open source project that is used to provide virtualised IP routing services over OpenFlow networks [46].
- SNAC: SNAC is a C++ based OpenFlow controller built on NOX, which uses a web-based policy manager to manage the network [40].
- Oflops: Oflops is an acronym that stands for OpenFlow Operations per second. It is a standalone controller that benchmarks various aspects of OpenFlow switch [40].
- Resonance is a Network Access Control application built using NOX and OpenFlow [40].

2.8 SDN in the Data Centres

The sudden increase in the data processed by data centre, the size of the data centre and the speed of operation needed in processing such high volume of data in todays' data centres have been a difficult task in the operations of data centres because the current topology was not designed to handle such enormous data processing.

2.8.1 The Need for SDN in Data Centres

Today's data centre requirements include efficiency, agility, scalability, easier to deploy and easier to manage multitenancy, network virtualization and simplicity. All these are the reason for adoption of SDN in data centres to provide solution to these problems which are complex and difficult to solve using today's current network architecture and other problems that may arise in the future [47]. SDN with OpenFlow can be used to monitor the entire network to know status of the networking devices in the network. Application may be deployed to power off obsolete virtual machines and network devices that are not used at a time to save cost on energy used in the data centre. SDN will help data centres to overcome the current network limitation in data centres by providing some of the solution listed below:

- SDN provides an easier scalable architecture in data centre by making use of already existing tunnelling technology and virtual networks which can span to millions of virtual switches in the data centres and making use of OpenFlow Protocol to make the scaling much easier than current network architecture.
- Adding, moving and deleting resources in data centre is promised to be easier with SDN which will make SDN-enabled data centres to be more agile in automating these actions dynamically.
- Failure recovery in today's data centre is complex due to the size and scale of data centres. The use of SDN in covering the complete view of the entire network makes it easier to get the best result in failure recovery.
- SDN will help data centres to customise routing and traffic engineering. A proper measurement and monitoring tools must be in place to monitor and calculate the best path

for the network traffic in the data centre and in interconnection of different data centres through the east-west traffic. This happens when different sections of data are being stored in different data centre locations and the data need to be pulled to make up a whole required data.

In summary, the use of SDN in data centres will help to provide better services, better control, better scalability, cost efficiency and fault tolerance which the current network cannot offer.

2.8.2 Benefits of SDN

Through logically-centralised controller, SDN gives network manager the flexibility to programme, configure, manage, secure and optimise network resources via dynamic automated SDN programs. Operations and management complexity have been reduced since the inception of OpenFlow-based SDN. SDN enables enterprises and carriers to address higher bandwidth requirements and ever dynamic change in today's services [3]. Some of the benefits of OpenFlow-based SDN include:

- Centralized control of multivendor environments: SDN controller can control any OpenFlow-enable network device from any vendor. SDN-based management tools and API can be used very fast to configure, deploy and update network devices across the entire network.
- Reduced complexity through automation: Improved automation and management using API to reduce Operational Expenses (OPEX) and to deliver a better service.
- Higher rate of innovation: SDN gives network to rapid innovation with the ability to deploy new policies and services across the network without the need to reconfigure each network device.
- Increased network reliability and security: Centralised and automated management of network devices and uniform policy enforcement have increased the reliability of the network. It also reduces configuration errors which occur during configuration of individual network devices in the network.
- More granular network control: OpenFlow-based implemented network can apply wide-range policies at the session, user, devices and application level to support multi-tenancy in the cloud environment.
- Better user experience: Centralised network control has helped infrastructure to adapt to users' needs [3].

In conclusion, literature review on SDN has been written in this chapter to provide information on SDN and all the necessary information that need to be known on the topic. The next chapter will go into more details on OpenFlow as a protocol to be used in this research.

3. OPENFLOW

OpenFlow is the first communication standard for SDN and it was standardised by the Open Networking Foundation (ONF) [48]. OpenFlow idea was first proposed in March 2008 initially for researchers to perform experiment on daily used network like campus and for network vendors to implement OpenFlow in their switch products. This started by some researchers with Martin Casado at the forefront of the research which is the continuation of Ethane research at Stanford University [36]. Network researchers always face problem of switches with proprietary operating system running on them. There are also additional protocols running on the network as well. Testing of experimental innovative ideas over the traditional networking environment has never been easy which prompted the concept of OpenFlow.

The concept of OpenFlow in SDN is to enable researchers to execute experiments and run it on a vendor-neutral hardware to write programmes that run on the hardware at line rate. This is done through the abstraction of the control plane from the data plane i.e. the separation of the control plane from the forwarding or data plane of a switch. This separation makes OpenFlow a standard communication interface between the control plane and the forwarding plane. It also empowers the controller to communicate with multiple network devices on the network.

The control plane which acts as the controller is moved to a centralised location to the networking device. The controller connects securely to the OpenFlow switches using OpenFlow protocol and can perform different functions such as adding, updating and deleting flow entries from the switches. OpenFlow uses flow tables which are like lookup tables in traditional switches and routers. The flow tables in the data plane can perform some layer 3 functions like the lookup table which include forwarding, metering, shaping, and firewall at line rate [36].

The flow table contains flow entries which are used in forwarding decisions. The decisions are based on match and action rules which can be created by the controller or modified by the network administrator via the logically centralised controller. Management of the entire network is also done via the logically centralised controller using the OpenFlow protocol.

3.1 OpenFlow Enabled Switch

OpenFlow enabled switches can be either hardware switches or software switches.

1. Hardware OpenFlow switches are OpenFlow enabled switch that have been designed and produced by networking device vendors. Some of the vendors that are already producing OpenFlow enabled switches include Arista, Cisco, Brocade, Dell, Extreme Networks, HP, IBM and Juniper [49].
2. OpenFlow-enabled software switches make use of software to implement switching functions and execute on a computer system, it can also be used in a virtual environment. Some of the available OpenFlow enabled software switches include the following:

a. OpenvSwitch – OpenvSwitch is a multilayer software switch designed to implement a production quality switch platform and to enable massive network automation through programmatic extension. It also supports standard management interfaces and protocols. OpenvSwitch is the most popular readily available OpenFlow software switch for SDN. It is mostly used as a virtual switch in virtual machine (VM) environment. It was designed to support distribution across multiple physical servers and across multiple Linux-based virtualization technologies such as Xen, KVM and VirtualBox. It also supports some commonly used protocol in networking including Netflow, sFlow, IPFIX, RSPAN, CLI, LACP and 802.1ag [50], [51].
b. Pica8 is an open switch software platform for hardware switching chips that include layer 2 and layer 3 and support for OpenFlow and other SDN features.
c. Indigo virtual switch is a lightweight, high-performance virtual switch with OpenFlow protocol support and designed to enable high-scale network virtualization applications [52].
d. Ofsoftswitch13 is an OpenFlow 1.3 compatible user-space software switch [53].

3.2 Understanding OpenFlow Enabled Switch

An OpenFlow enabled switch is connected to the controller via a secured channel using the OpenFlow protocol. The controller communicates with the switch via a secure channel that use SSL or TLS and they both exchange security certificates for their authentication. The switches also communicate with the hosts and the hosts can communicate with each other as well. OpenFlow message are exchanged between the controller and the switch to configure, monitor and manage the network. A connection diagram depicting their communication is shown in figure 3.1. An OpenFlow enabled switch consists of at least three parts namely:

1. A group table and one or more flow tables that make forwarding decision based on the match action associated with each flow entry in the flow table [54], [55].
2. One or more secure OpenFlow channel that connects the switch to the controller. The channel uses TLS or SSL to allow command and packets to be sent between controller and the switch using the OpenFlow protocol.
3. OpenFlow protocol provides an open and standard way of communication between the controller and the switch. With the use of OpenFlow protocol, the controller can perform many functions with the ability to add, update, and delete flow entries in flow tables both reactively and proactively.

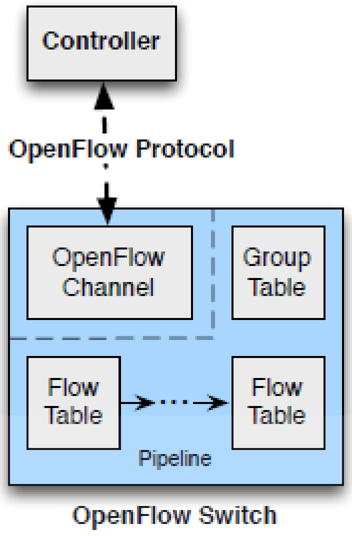


Figure 3.1: Main component of an OpenFlow Switch [55]

3.3 OpenFlow Table

Packet forwarding in an OpenFlow enabled switch that make use of one or more flow tables to store flow entries. On receiving an incoming packet in a flow table, a flow entry lookup is performed and instructions related to the flow entry are executed if there is a match in the flow table. If there is no match, the switch will take over the decision which depends on the configuration of the switch. Each flow entry has instructions related to the modification and forwarding of the packets in the flow table. According to OpenFlow specification, each OpenFlow entry has the following actions identified with it:

- To forward the flow' s packets to a given port
- To encapsulate and forward flow's packets to a controller
- To forward the flow's packets to the next flow table
- To drop the flow's packets [56].

Each flow entry consists of the following components namely: Match fields, Priority, Counters, Instructions, Timeouts, Cookie and Flags [55]. Some of these components are explained further in the following sections.

3.3.1 Match Field or Packet Header

The match field is used to match against incoming packet header after the table lookup have been performed first to decide on the forwarding. The match field contains some specific pipeline field use in the match, some of which are ingress port, metadata, Ethernet source, Ethernet destination, Ethernet frame type, VLAN id, VLAN priority, MPLS label, IPv4 source address, IPv4 destination address, IPv4 protocol, ARP code, TCP, UDP, SCTP source and destination port, ICMP type and ICMP code. Only the required fields are used for the Match fields while other fields are wild-carded out. Figure 3.2 shows the flow chart for the flow of packet through an OpenFlow switch. The incoming packet starts at table 0 and look for a match in the flow table, if there is a match then the action set, metadata and other required fields counters are updated after which the flow is sent out by executing the action set associated with the flow. If there is no match then, then flow will be sent to table-miss flow entry after which the packet may be dropped if there is no match in the table-miss flow entry, but if there is a match then it will update the counters and then sent out for action [55].

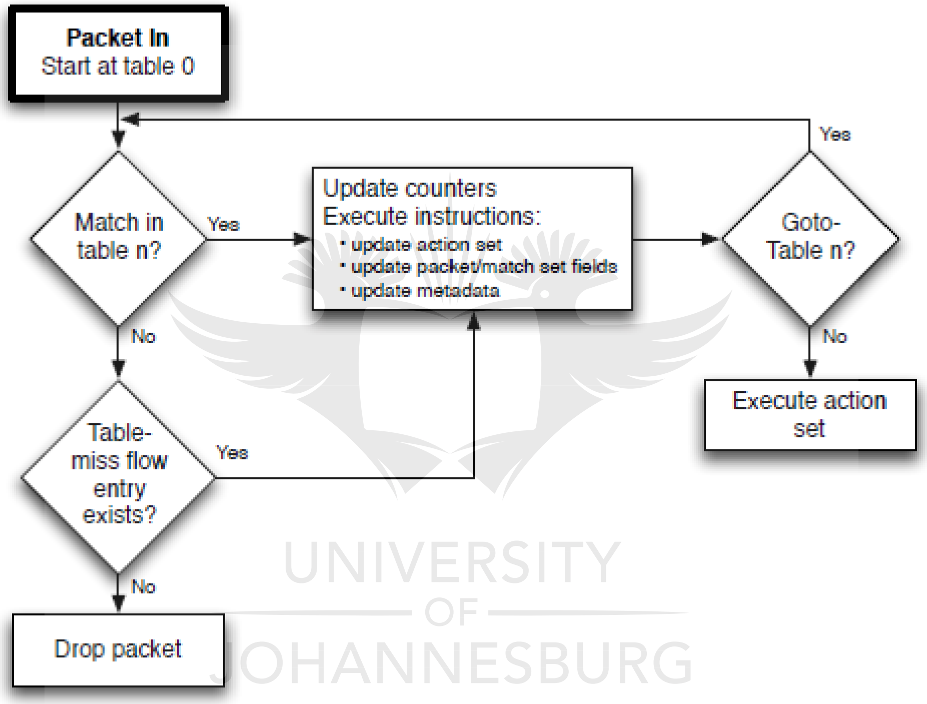


Figure 3.2: Flow chart showing packet flow through an OpenFlow switch [55]

3.3.2 Priority

The process of table lookup against the incoming packet makes use of priority component in the flow entry. In matching of the packet against the flow table, the selection is based on the matching of the packet with the flow entry with the highest priority. If there is a flow entry with multiple matching and possession of flow entry with same highest priority, the flow choice to be used in that case is not determined. It is the duty of the controller to specify desired priority and over-lapping flows by setting the OFPFF_CHECK_OVERLAP bit on flow mod messages to avoid multiple matching flow entries.

3.3.3 OpenFlow Table Pipeline Processing

The OpenFlow table pipeline processing determines the process and the interaction of packets with OpenFlow tables. Pipeline processing specifies the packets process and forwarding by the switch. An OpenFlow switch (hardware or software based) consist of one or more flow tables, with each flow table containing multiple flow entries. An OpenFlow switch with one flow table has a very simple pipeline processing and it is also valid [55].

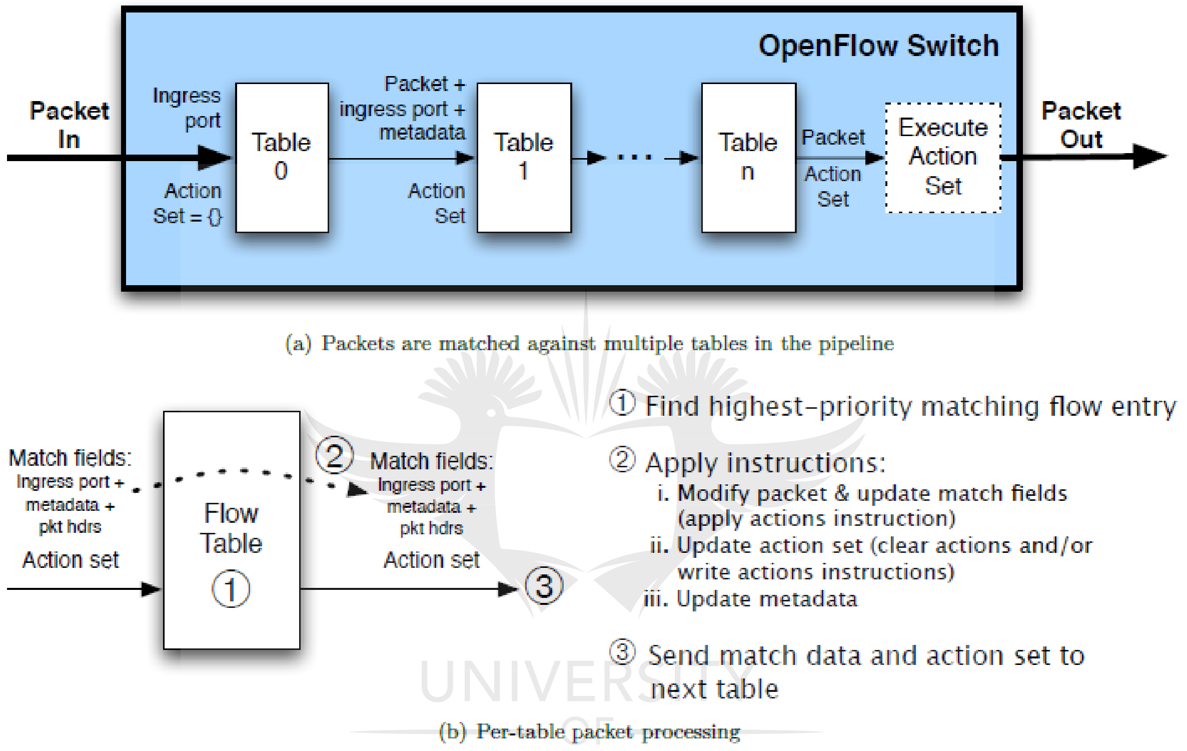

OpenFlow Switch
Packet In
Ingress port
Action Set = {}
Table 0
Packet + ingress port + metadata
Action Set
Table 1
Table n
Packet
Action Set
Execute Action Set
Packet Out
(a) Packets are matched against multiple tables in the pipeline
Match fields: Ingress port + metadata + pkt hdrs
Action set
Flow Table ①
② Match fields: Ingress port + metadata + pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table
(b) Per-table packet processing

Figure 3.3: Packet flow through the processing pipeline [55]

OpenFlow tables are sequentially numbered starting at 0 up to nth table. Figure 3.3 shows the details of the pipeline processing from packet-in to packet-out. The pipeline processing for packet that enters the switch start at flow table number 0, by matching the incoming packet again flow entries of flow table number 0. If a match is found, the instruction set associated with that flow entry is executed. The instructions may contain actions or modified the pipeline processing. An example of instruction that contains a forwarding action is shown in table 3.1

Header Field	**Actions**	**Priority**
if IP_address = = 192.168.1.10	re-write to 10.0.0.2, forward port 2	32768

Table 3.1: Flow table with forwarding action example

The priority is also taken into consideration in the forwarding, after a match has been found in the table number 0, the instructions associated with that flow entry where a match has been found will be executed and the instructions may also send the packet to another flow table using one of the required instruction type named Go-to Table Instruction.

Pipeline processing uses the Go-to instruction to specify the next flow table to direct the packet into which can only be a flow table with higher number than its own. This shows that pipeline processing can only be executed in forward direction. Instructions used to modify packet header and update the match fields are performed before the new match is executed.

If there is a match in the new table, another set of instruction will be applied to the action set in the flow table. This process is repeated throughout every subsequent table in the pipeline processing until a match is found which puts an end to pipeline processing at the table where there is a match, after which the action sets associated with the packet are executed and the packet is then forwarded [55], [56]. In the case where there is no match for the packet against all flow entries in the flow table, this is usually referred to as table-miss.

3.3.4 Table-Miss Flow Entry

Table-miss comes into action when there is no match for the packet against the flow entries in the flow table. It is ideal for every flow table to support a table-miss flow entry to process table misses. Table-miss flow entry has the entire match field wild-carded and has the lowest priority of 0 [55].

The controller may add the table-miss flow entry because it does not exist by default. Table-miss can also be removed by the controller and it expires over time. Table-miss flow entry is used in the processing of unmatched flow entries in the table. These processing include the following possible actions associated with table-miss flow entry which are:

1. The packet is sent to the controller using the controller reserve port
2. The packet is dropped
3. The packet is directed to another table with higher table ID than the current table.

3.3.5 Instructions

Instructions are used to perform operation on packet after a packet has been matched against flow entry. An OpenFlow switch must support the required instructions and optional instructions which may be a choice to set [55]. Some of the supported instructions types are:

- Write-Actions, actions – Required
- Go-to table, next-table-id – Required
- Meter, meter_id – Optional
- Apply-Actions, actions – Optional
- Clear-Actions – Optional
- Write-Metadata, metadata/mask – Optional

Instructions contain actions or modified pipeline processing that are associated with each flow entry. The actions are used to decide on the packet modification, packet forwarding, group table processing while pipeline processing decides on the forwarding of packets to another flow table with higher table_id than the current table for processing.

3.3.6 Counters

Counters are used to take statistics of all events on an OpenFlow switch. These include statistics across various parameter on an OpenFlow switch which includes; flow table, flow entry, port, queue, group, group bucket, meter and meter band. An example of using counters are, the packet received and packet transmitted on a port in an OpenFlow switch [55].

In summary, OpenFlow switch consists of a flow table which is used for packet lookup and forwarding. Each flow table also consists of flow entries which contain component fields like:

a. Packet header or Match field with information such as ingress port, metadata and others used for matching against incoming packets
b. Priority is used in the matching of the highest priority (highest number) flow entries against the packet to decide on forwarding the packet.
c. Counters are used to take statistics of event in an OpenFlow switch. Statistics such as number of received packets for a flow, number of bytes for a flow and duration of the flow.
d. Instructions that need to be applied to a matched packet against flow entries that decides on the processing of the packet which might be to forward the packet, pass it to the next table, pass it to the controller or drop the packet.

When an OpenFlow switch receives a packet in an OpenFlow network environment, the OpenFlow switch parses the packet header field and check for a match against the match field component of the flow table entries. The matching starts at the first flow table (number 0) and continues through subsequent flow table entries in the pipeline processing. If a match exists, the switch applies the specific instruction associated with the matched flow entry. In matching, priority is given to flow entry with the highest priority number. The counters then update the matched flow entry after each flow entry match against the packets. The flow table must also have the table-miss flow entry to further process unmatched flow entries in the table and then decide on what to do with the packet, which may be to forward the packet to the controller, drop the packet or direct the packet to another flow table for pipeline processing [2].

3.3.7 OpenFlow Secure Channel and Control Channel

The OpenFlow secure channel is the interface used for communication between OpenFlow controller and OpenFlow switches. OpenFlow controller uses this interface to configure the switch, manage the switch, receives events from the switch, and sends packets out of the switch. The control channel may support single or multiple OpenFlow channel to communicate and manage multiple switches on the network. The OpenFlow secure channel communication is encrypted using TLS which is more secured than TCP, although TCP may also be used.

3.3.8 OpenFlow Secure Channel Messages

The OpenFlow protocol supports three types of messages over the OpenFlow channel, namely: controller-to-switch, asynchronous and symmetric. The three are explained in detail below:

Controller-to-switch Messages

The controller-to-switch messages are established by the controller and sent to the switch over secured channel to inspect the state of the switch, configure the switch and manage the flow tables [55]. These messages may require response from the switch and sometimes may not necessarily require a response from the switch. The messages have the following attributes:

- Features – A features request message is sent to the switch requesting the identity and capabilities of the switch. The switch in return must respond with a feature reply message specifying the identity and capabilities of the switch.
- Configuration – This message allows the controller to configure the switch with needed parameters. Response is then received from the switch and the required information is sent to the controller
- Modify state – These messages are sent by the controller to the switches to manage the switches and it can add, delete and modify flow/group entries in the OpenFlow tables and set switch port properties.
- Read state – These messages are used by the controller to collect statistics, current configurations and capabilities from the switches.
- Packet out – The Packet-out messages are used by the controller to forward packet received via Packet-In messages out of specified port on the switch.
- Barrier – There is barrier request messages and reply messages that are used by the controller to ensure messages are delivered accordingly and to receive notification of completed operation.

Asynchronous Messages

Asynchronous messages are set-up by the switch and sent to the controller to notify the controller of packet arrival, switch state change or error. These messages are sent without the controller soliciting for the messages from the connected switches [55]. Asynchronous messages are of four main types namely:

- Packet-In: Packet-In event messages are sent to the controller for decision making; this happens whenever there is no match in the flow entry or the table-miss flow entry, or whenever there is a match with actions that orders the switch to send the packet to the controller such as TTL checking. Some switches have the capabilities to buffer packets, such switches are therefore configured to buffer the packets and send only some fraction of the packet header with default 128bytes. The controller uses buffer-id to identify each buffered packet when the switch forwards the packet.

- Flow-Removed: Flow removed messages are sent to the controller whenever flow entries are removed from the flow tables. Flow removal can either be because of idle timeout or hard timeout associated with each flow entry. Idle timeout can be set to a specific number of seconds in which the flow entry will expire and be removed if there is no match with the flow entry over the period given; while hard timeout is set to many seconds which will denote the lifetime of the flow and will be removed when the set time has been reached.
- Port-Status: Port status message is used to inform the controller of changes on the status of a port, for example, if the port was brought down.
- Error: This message is used by the switch to notify the controller of any problem that arises.

Symmetric Messages

Symmetric messages are sent in either direction by the controller or the switches [55]. These can be in the form of:

- Hello: Hello messages are exchanged between the switch and the controller when the connection starts-up
- Echo: These messages are in request/reply type. These messages can be sent by either the switch or the controller and must return an echo reply to verify the live-ness of a controller-to-switch connection, to measure bandwidth and latency.
- Experimenter: This is for future uses and provides a standard way for additional functionality to be added to OpenFlow messages in an OpenFlow switches.

In conclusion, OpenFlow protocol is a great protocol for networking with a lot of features that are easily configured and managed. More features are being added to enhance the use and the productivity of the protocol in SDN.

4. PROTOTYPE DESIGN AND IMPLEMENTATION

To achieve the objective of this research, a prototype LAN network was built. This was achieved by choosing design format that is readily available and adaptable. The following sub-headings listed the materials, Operating systems, applications and laboratory set-up used to achieve the aim of this thesis. Other Operating systems, applications and materials are listed in the subsequent subsections as they are used.

4.1 Laboratory Setup

A virtual network environment was set-up using a laptop with a quad core Intel Core i5 clocking at 2.6GHz and has 16GB of RAM. The laptop was running Windows 10 operating system as host. The prototype design requires a virtual network environment with the use of virtual machine (VM).

VirtualBox virtualization software from Oracle was installed on the Windows 10 host for settling a working virtual network environment. VirtualBox was chosen because it is free and readily available. Ubuntu 14.04 Linux Operating system was then installed on the VirtualBox which then run as a VM on the Windows host. OpenFlow specification 1.3.4 was installed at the time of starting this research. Newer versions have been released as well but this specification was chosen because it was supported by the virtual switch and controller being used at the time this research is being conducted. All other configuration was done by installing necessary applications and dependencies required to achieve the aim of the research. Some of the remaining applications needed are listed below.

4.1.1 Mininet

Mininet is the main application that will be used to build the virtual network environment. Mininet is a network emulator that creates a realistic virtual network and runs a collection of end-hosts, switches, routers and links on a single Linux kernel. A light weight virtualization is used to make a computer system e.g. laptop look like a complete network running the same kernel, system and user code. Hosts in Mininet acts just like a real machine that can run programs and can even send packets with the emulation of a real Ethernet interface. Some of the benefits of Mininet are speed, ability to create custom topologies; packet forwarding customisation, easy to use by running python scripts and it is open source [57], [58].

4.1.2 OpenvSwitch

OpenvSwitch is the most popular OpenFlow software switch for SDN. OpenvSwitch supports standard management interfaces and is used as a virtual switch in VM environments. It supports multiple Linux-based virtualizations such as Xen/Xen server, KVM, and VirtualBox. OpenvSwitch is made up of many components some of which are listed below [50], [51]:

- ovs-vswitchd: This is a daemon which implements the switch, along with a companion Linux kernel module for flow based switching.

- ovsdb-server: This is a lightweight database server that ovs-vswitch queries to obtain the stored configuration of the OpenvSwitch.
- ovs-vsctl: This is a tool used to query the switch to get the configuration and also to update the configuration of ovs-vswitchd.
- ovs-appctl: This utility is used to send command to a running OpenvSwitch daemon.
- ovs-dpctl: This is a tool used for configuring the switch kernel module. It is used in the administration of OpenvSwitch datapath.
- ovs-ofctl: This is the utility used for querying and controlling OpenFlow switches and controllers.

OpenvSwitch will be used in the virtual network environment in conjunction with Mininet to create a prototype network laboratory to be used for the research. OpenvSwitch version 2.3.1 will be used for this research.

4.1.3 Controller Selection

Controllers are the brain of the network in an SDN environment. The choice for the right, simple, available and open source SDN controller should be taken seriously to achieve a smooth research. The list of available SDN controllers have already being mentioned in Chapter 2.

After comparing the list of available SDN controllers, RYU controller was chosen. SDN controller comparison journal paper submitted was carefully examined and RYU controller fulfils most of the requirement needed for this research [59].

4.1.4 Ryu Controller

Ryu is a Japanese word for flow. Ryu is a component based SDN controller. It has well defined components that can be modified, extended and composed for creating network management applications and customised controller applications. Ryu supports fully OpenFlow specification 1.0, 1.2, 1.3, and 1.4. Ryu was chosen for the following reasons [43].

- It is an open source
- It is implemented entirely in Python
- It is readily available
- It has a Long-Term Support (LTS)
- It is well documented
- It has GUI and REST API support
- Ryu fully supports all the OpenFlow specifications
- It also supports Openstack Quantum support used for cloud computing.

One of the most important choices of selection is that it uses Python which is easier and user friendly than other programming languages.

4.1.5 RYU Application Creation Model and Operation Methodology

Ryu application uses three creation and operation methodologies namely: threads, events and event queues. Applications running on Ryu controllers are single-threaded entities and can be achieved by creating a subclass, a subclass of ryu.base.app-manager.RyuApp. Ryu makes use of FIFO for event queues; each Ryu application uses the FIFO for the preservation of the order of events which is later used for the processing of the events which is done by de-queuing the event and process the event with the help of event handler [60].

4.1.6 OpenFlow Event Classes

The event handler uses the ryu.controller.handle.set_ev_cls decorator to listen to specific events generated whenever an OpenFlow message is received. The event class name used is ryu.controller.ofp_event.EventOFP+<OpenFlow message name> e.g. For Packet-In message, the OpenFlow message name is PacketIn so the event class name will become EventOFPPacketIn. Ryu has a well-defined API and support REST. Its REST API can be used to update information on the switch. The REST API uses ryu.app.ofctl_rest to retrieve switch statistics and to update it as well [43] , [60].

4.1.7 Other Tools Used

List of other tools used in this research include the following:

- Iperf – Iperf is a command line tool used for measuring maximum bandwidth on IP network. It supports TCP and UDP and measures their performance which includes bandwidth and loss. For TCP, it can measure bandwidth and report MTU size while for UDP it can measure packet loss, delay jitter and modification of UDP stream with specific bandwidth usage [61].
- Netperf – It is a benchmark tool used to measure various networking performance by providing test for unidirectional throughput and end-to-end latency using TCP or UDP [62].
- cUrl – cUrl is a command line tool used to get files/documents from or send documents to a server using any of the supported protocols. The supported protocols are HTTP, HTTPS, FTP, GOPHER, and TELNET. It offers support for the use of proxy, user authentication, ftp upload, HTTP post, SSL connection and file transfer resume. It is a simple url in command form [63], [64].
- Wireshark – Wireshark is a network packet analyser that allows the user to see what is happening on the network. It works by capturing network packets, displays the packet data and then used to examine the details of communication on the network [65].
- Postman – Postman is GUI used to query network devices when developing API. It is used for testing and easy readability of the rules and status of the switches and routers on the network.

Different experiments were set-up to achieve the aim of this research. The first experiment shows the basic network set-up showing message communication on the network. Other experiments needed are also conducted during this research.

4.2 Experiment 1: Basic Network Setup

A simple basic network is setup comprising a controller, a switch and two hosts. The topology of the network is shown in figure 4.1

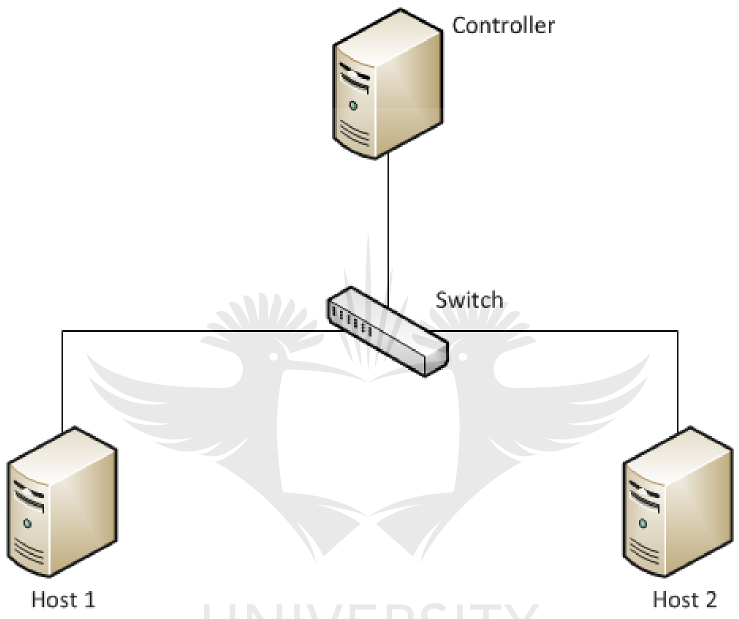


Figure 4.1: Basic network topology

To build the network, a terminal was used to execute the necessary commands. The first command at the terminal creates the network using Mininet with the following command:

```
sudo mn -topo single,2 -mac -switch ovsk -controller remote -x
```

The details of the command parameter are as follows:

1. In Linux "sudo" is used to get root access
2. "mn" signify Mininet
3. "topo" is used for topology to define number of switch and host e.g. single,2 means one switch and two hosts.
4. "mac" is used to automatically set the MAC address of the hosts
5. "switch" is used to signify the switch to be used e.g. either internal or remote
6. "x" is used to start xterm for each component created.

Another keyword to take note of is datapath which represents network devices, for example, a switch on the network, a single machine in the network may host many number of datapath. The details of the setup after running Mininet can be seen in figure 4.2 below.

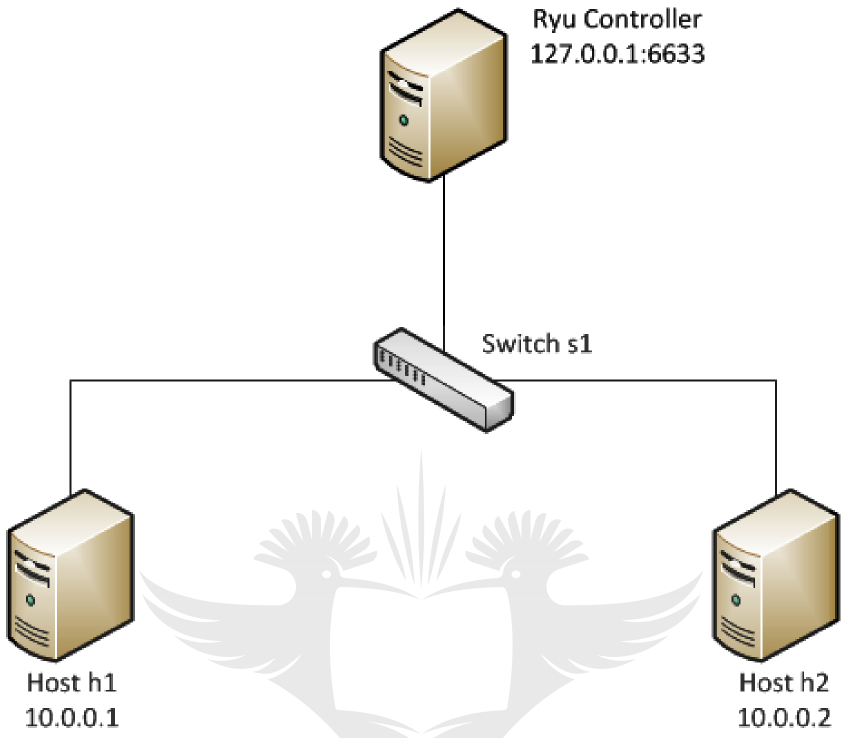
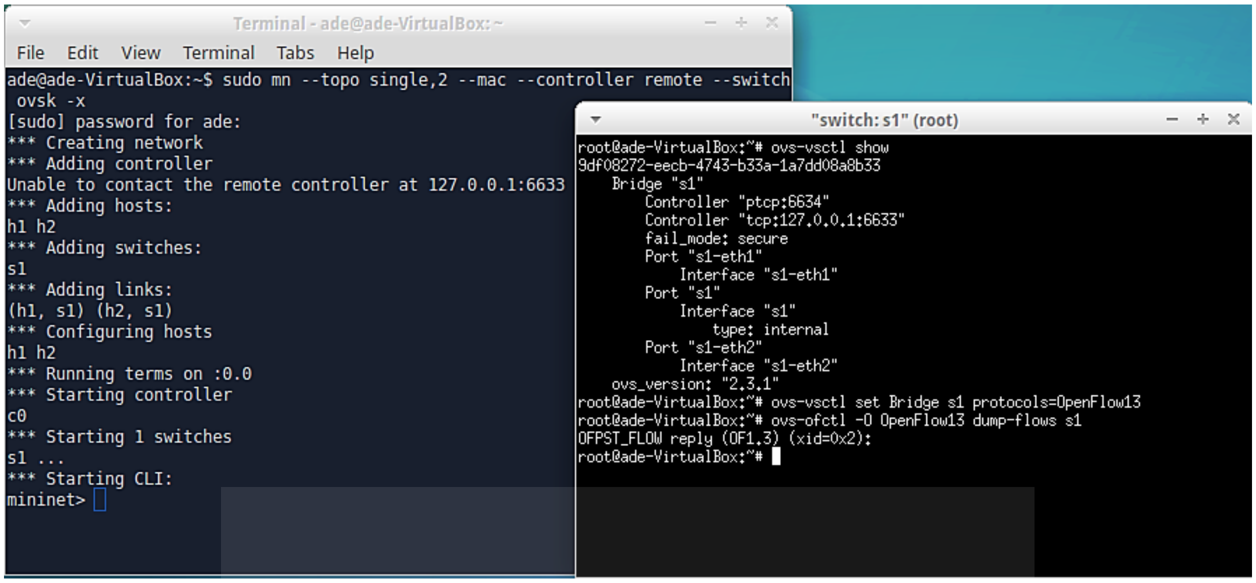


Figure 4.2: Basic network topology with details

Mininet created the network by adding the controller, switch and two hosts. It is shown that it is unable to contact the remote controller at 127.0.0.1:6633, this is because the Ryu controller has not yet been started. Terminals were created for the controller, switch, and the two hosts using xterm. Issuing different OpenvSwitch commands produced the following results shown in the attached figures. By issuing “ovs-vsctl show”, a brief overview of the database contents is shown on the screen showing port s1-eth1 and s2-eth2 that links to the two hosts and the OpenvSwitch version used which is 2.3.1.

Figure 4.3 shows the switch status and other commands which includes, the implementation of OpenFlow specification 1.3.4 protocol on the switch, the command “ovs-vsctl set Bridge s1 protocols=OpenFlow13” after which “ovs-ofctl” was issued to check the status of the flow table. This shows that the flow table is still empty and shows the version of the OpenFlow specification used as well.

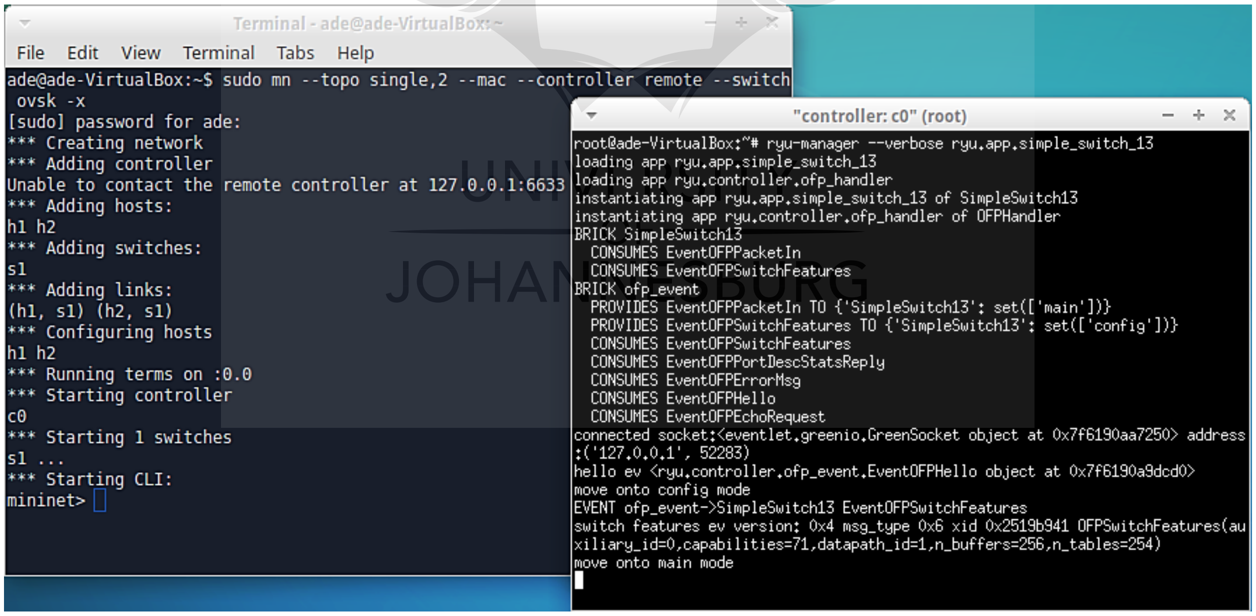

Terminal - ade@ade-VirtualBox: ~
File Edit View Terminal Tabs Help
"switch: s1" (root)

```
ade@ade-VirtualBox:~$ sudo mn --topo single,2 --mac --controller remote --switch
 ovsk -x
[sudo] password for ade:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Running terms on :0.0
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

```
root@ade-VirtualBox:~# ovs-vsctl show
9df08272-eecb-4743-b33a-1a7dd08a8b33
    Bridge "s1"
        Controller "ptcp:6634"
        Controller "tcp:127.0.0.1:6633"
        fail_mode: secure
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth2"
            Interface "s1-eth2"
    ovs_version: "2.3.1"
root@ade-VirtualBox:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
root@ade-VirtualBox:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
root@ade-VirtualBox:~#
```

Figure 4.3: switch status commands

Figure 4.4 shows the command to start the Ryu controller is then executed on the controller xterm “controller:c0”. The Ryu controller learning switch has been bundled with the controller and can be modified to perform more functions.

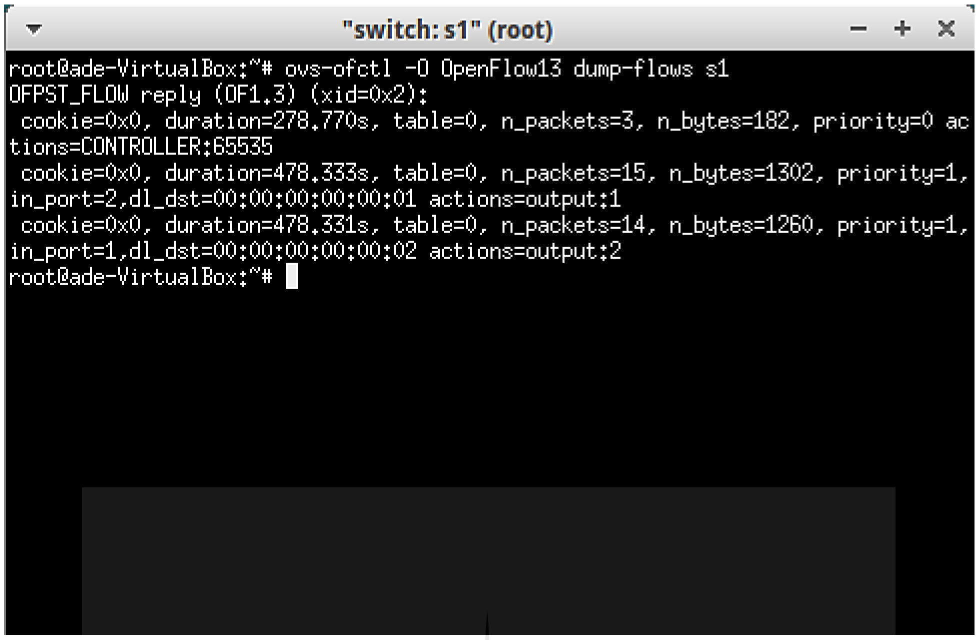

Terminal - ade@ade-VirtualBox: ~
File Edit View Terminal Tabs Help
"controller: c0" (root)

```
ade@ade-VirtualBox:~$ sudo mn --topo single,2 --mac --controller remote --switch
 ovsk -x
[sudo] password for ade:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Running terms on :0.0
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

```
root@ade-VirtualBox:~# ryu-manager --verbose ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
connected socket:<eventlet.greenio.GreenSocket object at 0x7f6190aa7250> address
:('127.0.0.1', 52283)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f6190a9dcd0>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x2519b941 OFPSwitchFeatures(au
xiliary_id=0,capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
```

Figure 4.4: Inside Ryu Controller

By issuing “ovs-ofctl –O OpenFlow13 dump-flows s1” again to check the status of the flow table produced the results show in figure 4.5



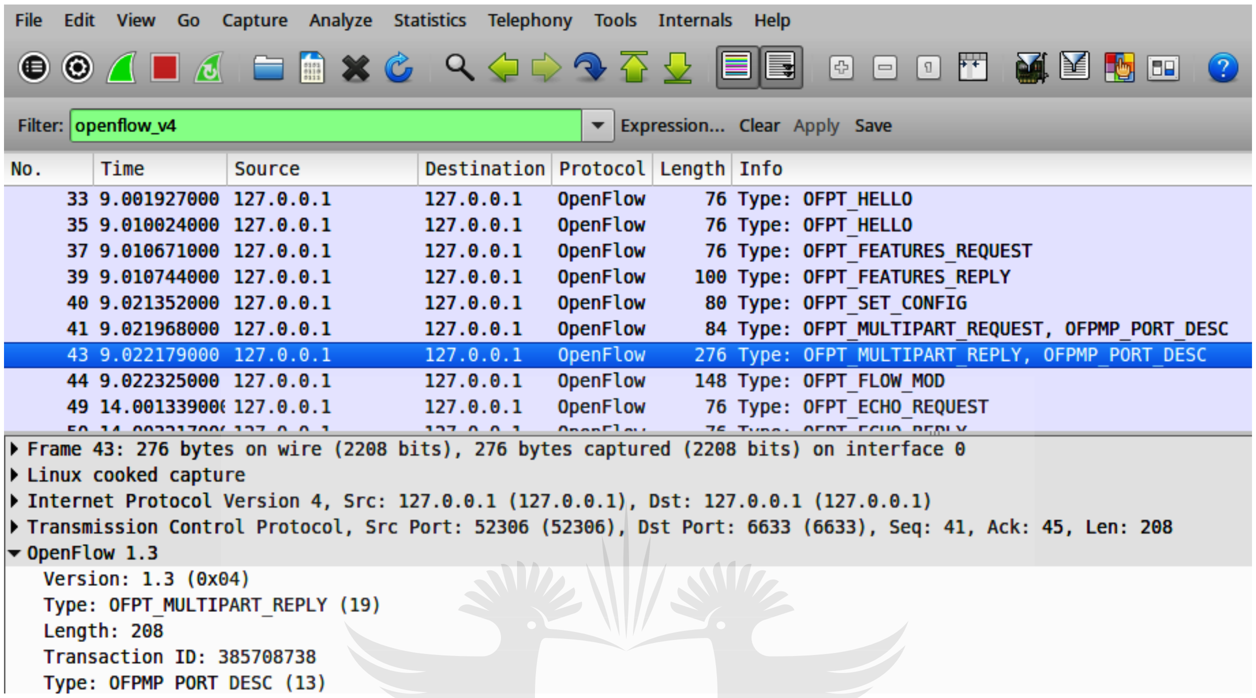
```
"switch: s1" (root)
root@ade-VirtualBox:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=278.770s, table=0, n_packets=3, n_bytes=182, priority=0 ac
tions=CONTROLLER:65535
 cookie=0x0, duration=478.333s, table=0, n_packets=15, n_bytes=1302, priority=1,
in_port=2,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=478.331s, table=0, n_packets=14, n_bytes=1260, priority=1,
in_port=1,dl_dst=00:00:00:00:00:02 actions=output:2
root@ade-VirtualBox:~#
```

Figure 4.5: OpenFlow table

It can be seen from switch terminal that the OpenFlow is using flow table number 0, and have 0 number of packet, priority of 0 and the actions is CONTROLLER. It also shows the updated flow table with two flow entries for the communication of the two hosts. The first flow entry shows that number of packets, the number of bytes, priority of 1, in_port which is port 2, destination MAC address which is 00:00:00:00:00:00:00:01 and actions which specify the output of port 1. The second flow entry also shows the same details as the first one but with different in_port which is port 1, MAC address which is 00:00:00:00:00:00:00:02 and actions which is output on port 2.

Message Communication between a Switch and a Controller in OpenFlow Network

Figure 4.6 shows details of exchange of messages between the switch and the controller. The switch establishes TCP handshake with the controller using loopback interface 127.0.0.1 and port 6633, while the switch is using port 52306. Both sides also exchange hello message by sending highest OpenFlow protocol version. The controller then sends feature request message to the switch and the switch respond with feature reply showing list of ports, port speed, available OpenFlow tables, OpenFlow version in use, datapath ID, and number of buffer, capabilities and actions. The controller then sends set-config message to the switch to set and query configuration parameters in the switch. The controller also sends OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC to the switch to obtain statistics or state information from the switch and the switch also respond with details containing the port description showing number of used ports, port number, hardware address of the ports, name of the port, configuration state (i.e. if the port is up or down), current speed, advertised speed, and supported speed.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: openflow_v4
Expression... Clear Apply Save

```
No.   Time         Source      Destination Protocol Length Info
   33 9.001927000 127.0.0.1    127.0.0.1   OpenFlow    76 Type: OFPT_HELLO
   35 9.010024000 127.0.0.1    127.0.0.1   OpenFlow    76 Type: OFPT_HELLO
   37 9.010671000 127.0.0.1    127.0.0.1   OpenFlow    76 Type: OFPT_FEATURES_REQUEST
   39 9.010744000 127.0.0.1    127.0.0.1   OpenFlow   100 Type: OFPT_FEATURES_REPLY
   40 9.021352000 127.0.0.1    127.0.0.1   OpenFlow    80 Type: OFPT_SET_CONFIG
   41 9.021968000 127.0.0.1    127.0.0.1   OpenFlow    84 Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
   43 9.022179000 127.0.0.1    127.0.0.1   OpenFlow   276 Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
   44 9.022325000 127.0.0.1    127.0.0.1   OpenFlow   148 Type: OFPT_FLOW_MOD
   49 14.00133900( 127.0.0.1   127.0.0.1   OpenFlow    76 Type: OFPT_ECHO_REQUEST

▸ Frame 43: 276 bytes on wire (2208 bits), 276 bytes captured (2208 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▸ Transmission Control Protocol, Src Port: 52306 (52306), Dst Port: 6633 (6633), Seq: 41, Ack: 45, Len: 208
▾ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_MULTIPART_REPLY (19)
    Length: 208
    Transaction ID: 385708738
    Type: OFPMP_PORT_DESC (13)
```

Figure 4.6: Wireshark Flow analysis

The controller finally sends OFPT_FLOW_MOD to the switch to modify the flow table which also contains the table ID, idle timeout, hard timeout, priority, buffer ID, out port, match, instruction and action. If there is an error, an error message will also be sent to the controller**.** Request messages are then sent frequently from the switch to the controller, while the controller respond with echo reply messages to exchange more information about their connection status, to keep the communication between them alive, to know the latency and the bandwidth between them.

Message Communication between Hosts on an OpenFlow Network

To analyse the exchange of messages between hosts on an OpenFlow network, some tools will be required to perform the task. Some of the tools required include Wireshark, iperf, cUrl, tcpdump and inbuilt OpenvSwitch commands. Some of the targeted functionalities to be found out include flooding, MAC address learning though the use of Address Resolution Protocol (ARP), forwarding and OpenFlow table. The topology to be used is shown in figure 4.7 by using Mininet to generate the network laboratory. The command “sudo –s” was first issued to get root access after which the following command was then issued, “mn –topo single,3 –mac –controller remote –switch ovsk –x” to create a controller, a switch and three hosts.

The switch was enforced to only use OpenFlow specification 1.3 because OpenvSwitch started by Mininet uses OpenFlow 1.0 by default. “ovs-vsctl show” was issued to query the status of the switch and its database, “ovs-ofctl dump-flows s1” was also used to query the OpenFlow tables. The Ryu controller was then started by running “ryu-manager –verbose ryu.app. simple_switch_13” on the controller xterm window.

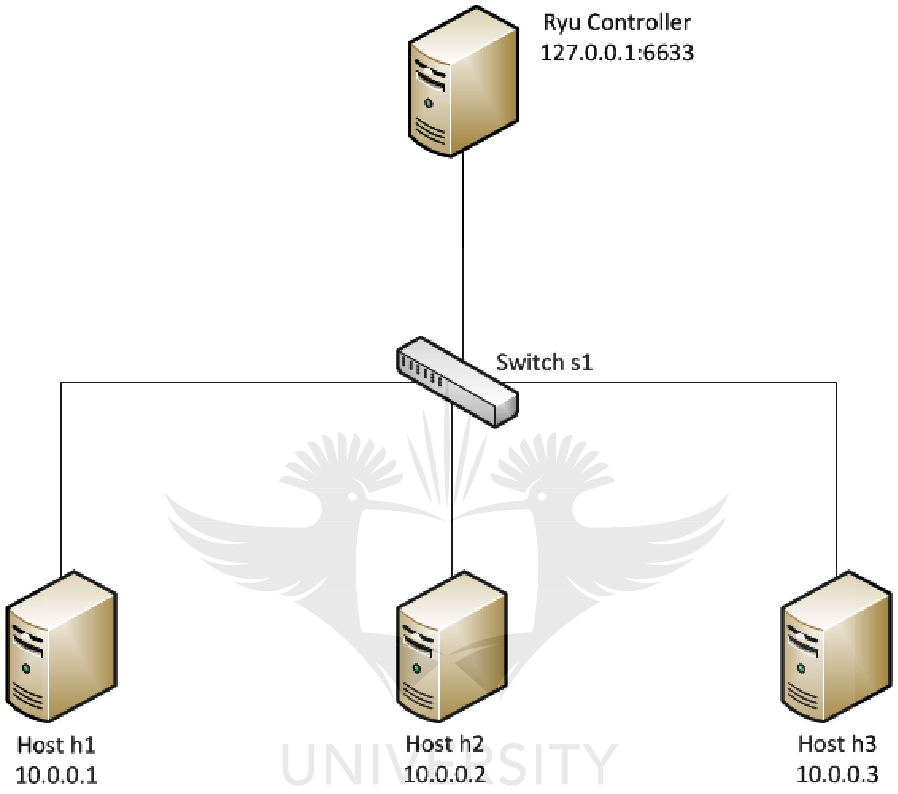


Figure 4.7: Message communication between Hosts

With the use of Wireshark, all network protocols on an OpenFlow network are encapsulated within the OpenFlow protocol. The data section of the OpenFlow1.3 version used showed the data-link layer which has the Ethernet with MAC address, others include IPv4, ICMP, ARP and others depending on the specific protocol being looked for which can be easily accessed by filtering the Wireshark display.

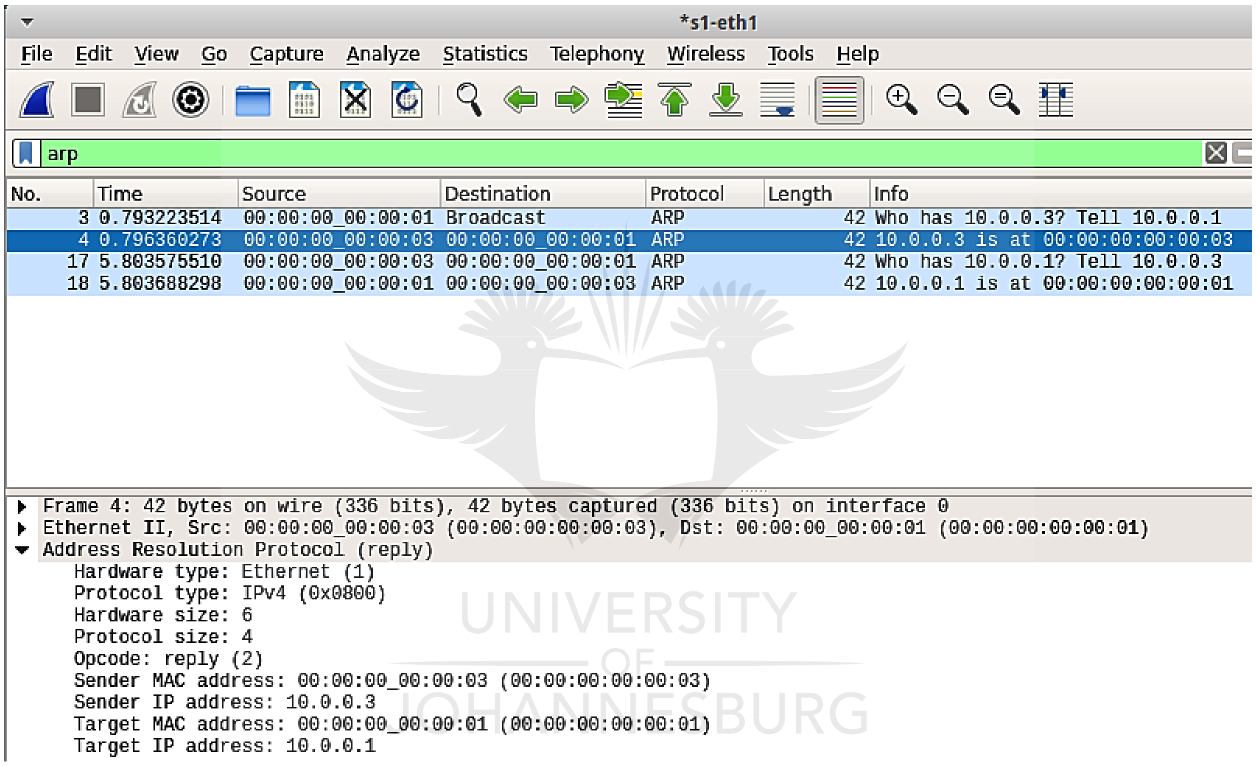
4.2.1 Result and Discussion

MAC Address Learning

After starting the network, the packets in the network were analysed to show details of information on the switch and in the network. The features reply by the switch in response to the feature request sent form the controller to the switch shows the switch datapath ID is 000000000000001 which is shown as dpid after query the switch s1 with “ovs-ofctl -O OpenFlow13 show s1, it also shows the capabilities of the switch such as flow stats, table stats, port stats and queue stats. This was followed by port description and flow modification as explained in experiment 1. To know the

MAC address of the hosts connected to the switch, Address Resolution Protocol (ARP) was used. ARP is normally used for the mapping of the network address such as IPv4 to MAC address.

A ping command tool "h1 ping –c8 h3" was issued on the Mininet terminal. The ping tool send ICMP packets from h1 to h2 and vice-versa, this was noted on the Wireshark as shown in figure 4.8. This also updates the flow table on the switch as well. MAC addresses learning starts when host h1 send a broadcast out on the network asking the entire connected host that "who has 10.0.0.3 tell 10.0.0.1" as shown in figure 4.8 below. Pinging and ICMP messages were used to check the connectivity and live-ness between two points.

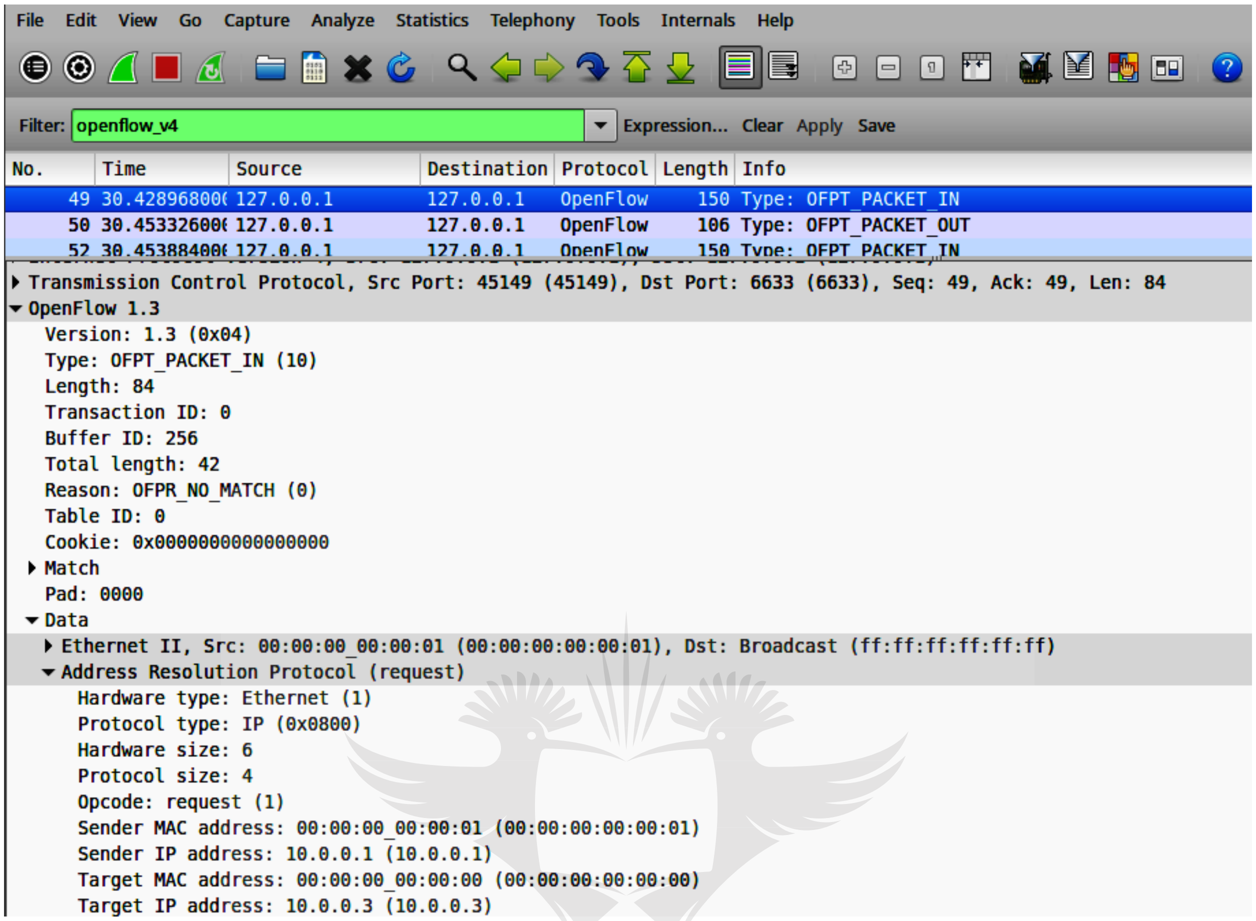

*s1-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp

```
No.  Time         Source             Destination        Protocol  Length Info
   3 0.793223514  00:00:00_00:00:01  Broadcast          ARP           42 Who has 10.0.0.3? Tell 10.0.0.1
   4 0.796360273  00:00:00_00:00:03  00:00:00_00:00:01  ARP           42 10.0.0.3 is at 00:00:00:00:00:03
  17 5.803575510  00:00:00_00:00:03  00:00:00_00:00:01  ARP           42 Who has 10.0.0.1? Tell 10.0.0.3
  18 5.803688298  00:00:00_00:00:01  00:00:00_00:00:03  ARP           42 10.0.0.1 is at 00:00:00:00:00:01

▶ Frame 4: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
▼ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 00:00:00_00:00:03 (00:00:00:00:00:03)
    Sender IP address: 10.0.0.3
    Target MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
    Target IP address: 10.0.0.1
```

Figure 4.8: Address Resolution Protocol

To know the MAC address of 10.0.0.3 with the use of ARP, the switch sends a "Packet_In" (which is an ARP request) to the controller after receiving the broadcast to know what to do with the packet. Host h3 using its MAC address responded with the message saying "10.0.0.3 is at 00:00:00:00:00:03". It also sends another message to host h1 that "Who has 10.0.0.1? Tell 10.0.0.3". Host h1 then responded by saying "10.0.0.1 is at 00:00:00:00:00:01". Figure 4.9 shows the Packet_In as the broadcast with the MAC address of the source and IP address of the destination.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: openflow_v4
Expression... Clear Apply Save

```
No.   Time             Source       Destination  Protocol  Length  Info
  49  30.42896800( 127.0.0.1    127.0.0.1    OpenFlow     150  Type: OFPT_PACKET_IN
  50  30.45332600( 127.0.0.1    127.0.0.1    OpenFlow     106  Type: OFPT_PACKET_OUT
  52  30.45388400( 127.0.0.1    127.0.0.1    OpenFlow     150  Type: OFPT_PACKET_IN
```

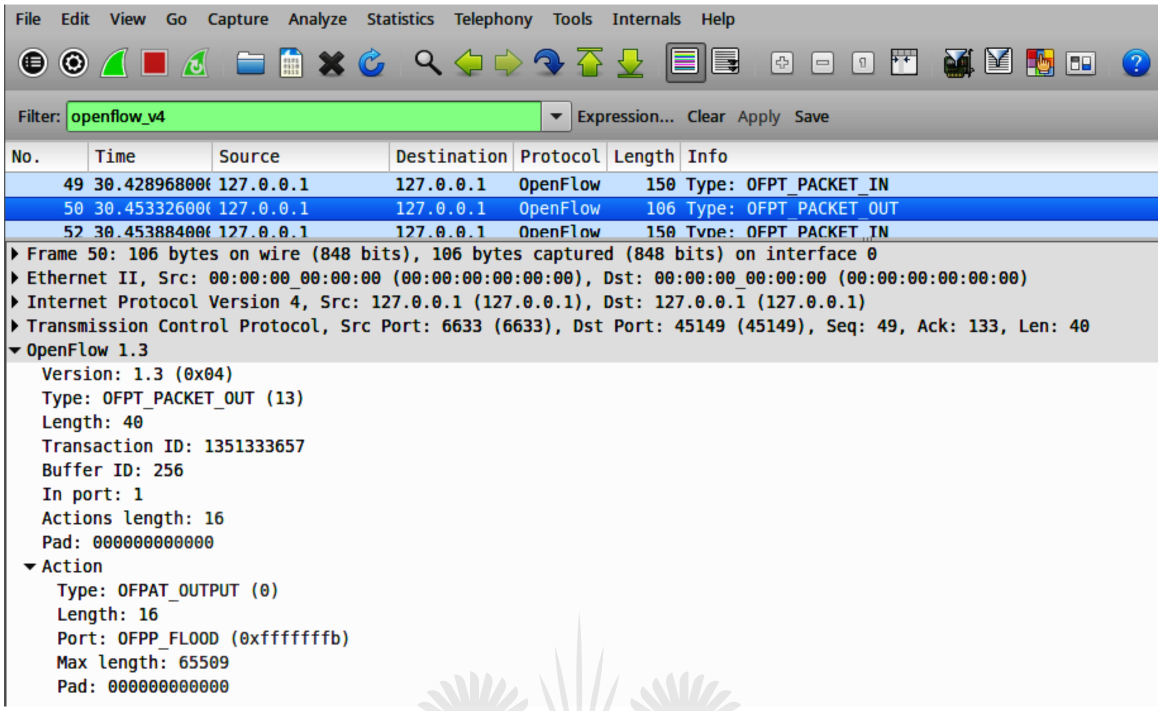
```
▸ Transmission Control Protocol, Src Port: 45149 (45149), Dst Port: 6633 (6633), Seq: 49, Ack: 49, Len: 84
▾ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 84
    Transaction ID: 0
    Buffer ID: 256
    Total length: 42
    Reason: OFPR_NO_MATCH (0)
    Table ID: 0
    Cookie: 0x0000000000000000
  ▸ Match
    Pad: 0000
  ▾ Data
    ▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    ▾ Address Resolution Protocol (request)
        Hardware type: Ethernet (1)
        Protocol type: IP (0x0800)
        Hardware size: 6
        Protocol size: 4
        Opcode: request (1)
        Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
        Sender IP address: 10.0.0.1 (10.0.0.1)
        Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
        Target IP address: 10.0.0.3 (10.0.0.3)
```

Figure 4.9: OpenFlow Packet_In Message

Figure 4.10 shows how the controller answers the switch back with a “Packet_Out” message instructing the switch to send the packet out to all ports excluding the source port which is port1 in this experiment, the switch will then wait for response.

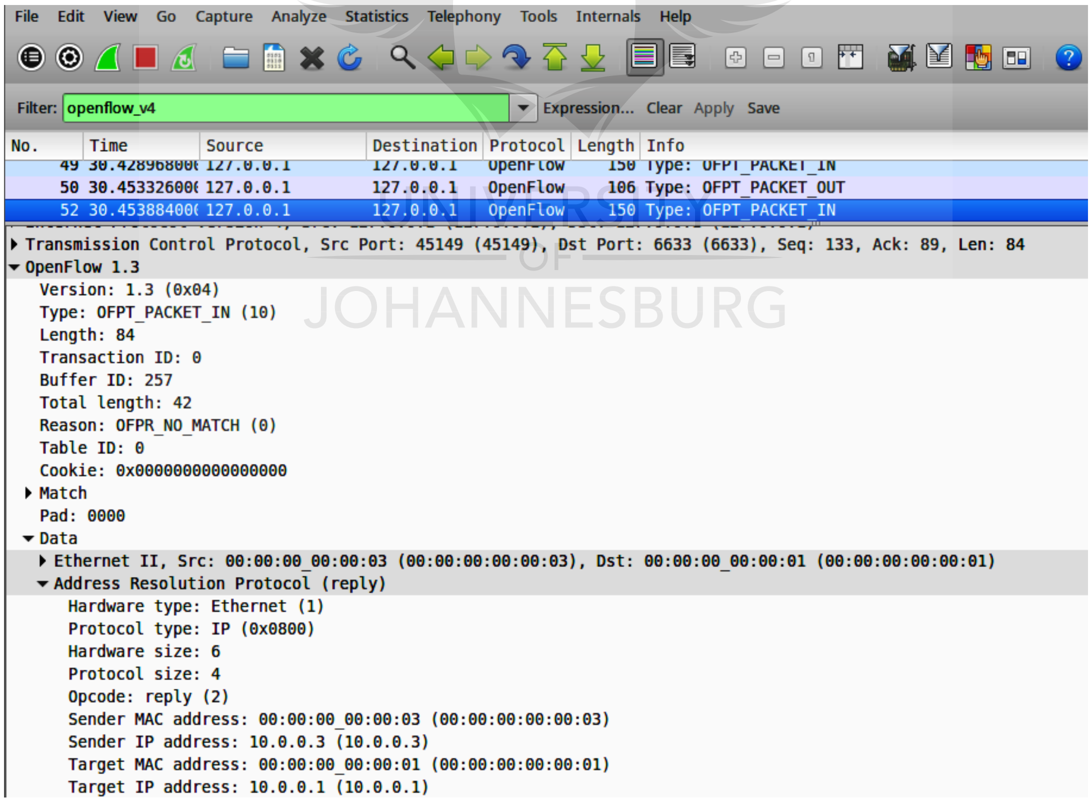
Host h3 respond with another “Packet _In” message which is an ARP reply to the switch which the switch also sends to the controller for forward decision making. Upon reception of the ARP reply, the controller sends “FLOW_MOD” message to the switch to install new flow entry in the flow table for present and future use. The figures below show the broadcast, Packet_In and Packet_Out.

Host h1 then sends ICMP message to host h3 as shown in figure 4.12. The status of the switch flow table was later checked with “ovs-ofctl – O OpenFlow13 dump-flows s1” as shown in figure 4.13. The flow table shows the priority of 0 for the message being sent to the controller while the communication between hosts has the priority of 1. The flow table also shows the in_port which is the receiving port on the switch and the destination port which is a MAC address of port1(host1) and the actions specifying the forwarding of the packet out of port1. This process was repeated from host1 to host2 with the ICMP messages. Additional information


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: openflow_v4
Expression... Clear Apply Save

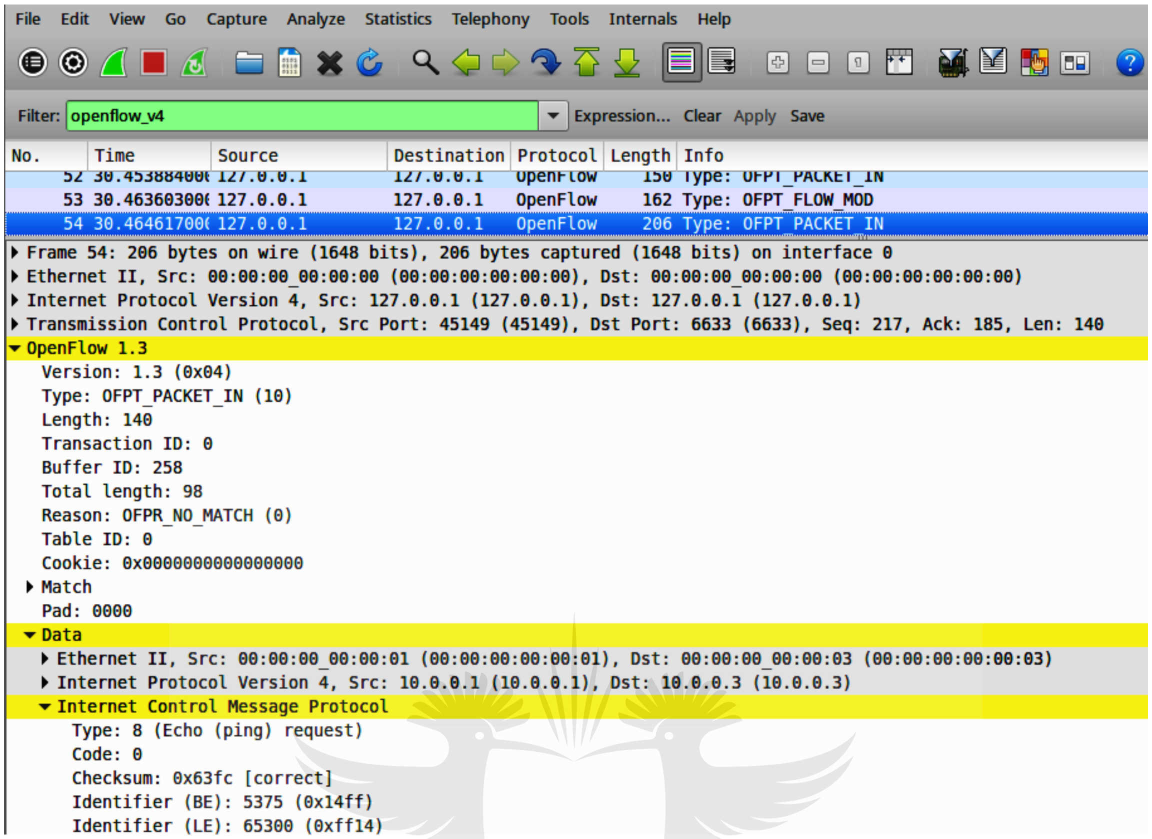
```
No.   Time            Source      Destination Protocol Length Info
   49 30.42896800( 127.0.0.1   127.0.0.1   OpenFlow    150 Type: OFPT_PACKET_IN
   50 30.45332600( 127.0.0.1   127.0.0.1   OpenFlow    106 Type: OFPT_PACKET_OUT
   52 30.45388400( 127.0.0.1   127.0.0.1   OpenFlow    150 Type: OFPT_PACKET_IN
▸ Frame 50: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▸ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 45149 (45149), Seq: 49, Ack: 133, Len: 40
▾ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_OUT (13)
    Length: 40
    Transaction ID: 1351333657
    Buffer ID: 256
    In port: 1
    Actions length: 16
    Pad: 000000000000
  ▾ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: OFPP_FLOOD (0xfffffffb)
      Max length: 65509
      Pad: 000000000000
```

Figure 4.10: OpenFlow Packet_Out Message


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: openflow_v4
Expression... Clear Apply Save

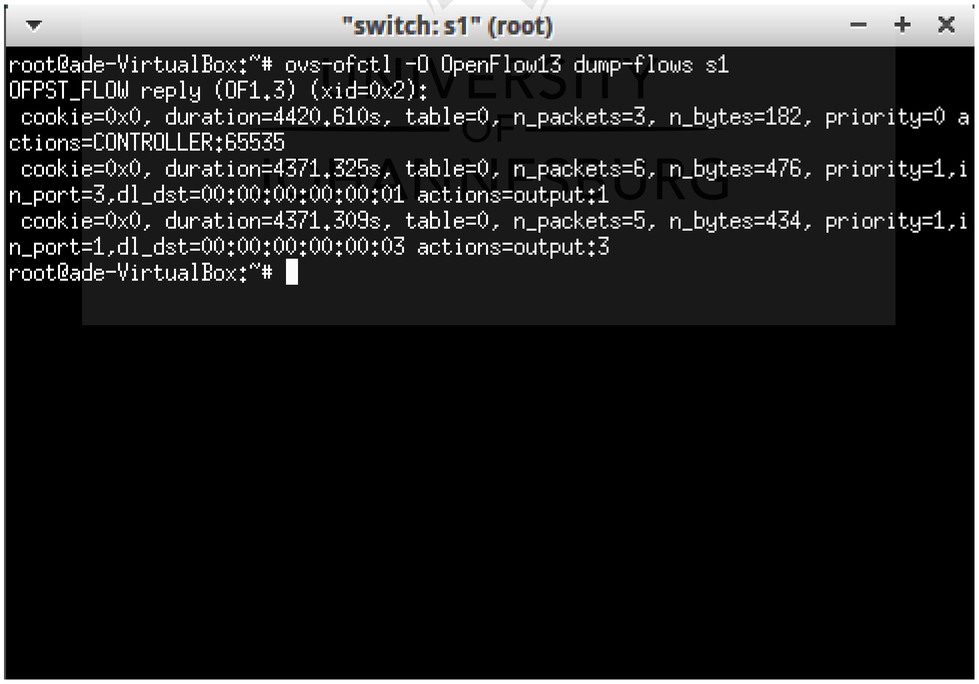
```
No.   Time            Source      Destination Protocol Length Info
   49 30.42896800( 127.0.0.1   127.0.0.1   OpenFlow    150 Type: OFPT_PACKET_IN
   50 30.45332600( 127.0.0.1   127.0.0.1   OpenFlow    106 Type: OFPT_PACKET_OUT
   52 30.45388400( 127.0.0.1   127.0.0.1   OpenFlow    150 Type: OFPT_PACKET_IN
▸ Transmission Control Protocol, Src Port: 45149 (45149), Dst Port: 6633 (6633), Seq: 133, Ack: 89, Len: 84
▾ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 84
    Transaction ID: 0
    Buffer ID: 257
    Total length: 42
    Reason: OFPR_NO_MATCH (0)
    Table ID: 0
    Cookie: 0x0000000000000000
  ▸ Match
    Pad: 0000
  ▾ Data
    ▸ Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
    ▾ Address Resolution Protocol (reply)
        Hardware type: Ethernet (1)
        Protocol type: IP (0x0800)
        Hardware size: 6
        Protocol size: 4
        Opcode: reply (2)
        Sender MAC address: 00:00:00_00:00:03 (00:00:00:00:00:03)
        Sender IP address: 10.0.0.3 (10.0.0.3)
        Target MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
        Target IP address: 10.0.0.1 (10.0.0.1)
```

Figure 4.11: OpenFlow Packet_In Message with destination IP and MAC address


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: openflow_v4
Expression... Clear Apply Save

```
No.   Time          Source        Destination  Protocol  Length  Info
   52 30.45388400( 127.0.0.1     127.0.0.1    OpenFlow    150 Type: OFPT_PACKET_IN
   53 30.46360300( 127.0.0.1     127.0.0.1    OpenFlow    162 Type: OFPT_FLOW_MOD
   54 30.46461700( 127.0.0.1     127.0.0.1    OpenFlow    206 Type: OFPT_PACKET_IN
▶ Frame 54: 206 bytes on wire (1648 bits), 206 bytes captured (1648 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶ Transmission Control Protocol, Src Port: 45149 (45149), Dst Port: 6633 (6633), Seq: 217, Ack: 185, Len: 140
▼ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 140
    Transaction ID: 0
    Buffer ID: 258
    Total length: 98
    Reason: OFPR_NO_MATCH (0)
    Table ID: 0
    Cookie: 0x0000000000000000
  ▶ Match
    Pad: 0000
  ▼ Data
    ▶ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:03 (00:00:00:00:00:03)
    ▶ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.3 (10.0.0.3)
    ▼ Internet Control Message Protocol
        Type: 8 (Echo (ping) request)
        Code: 0
        Checksum: 0x63fc [correct]
        Identifier (BE): 5375 (0x14ff)
        Identifier (LE): 65300 (0xff14)
```

Figure 4.12: Host h1 ICMP message



```
root@ade-VirtualBox:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=4420.610s, table=0, n_packets=3, n_bytes=182, priority=0 a
ctions=CONTROLLER:65535
 cookie=0x0, duration=4371.325s, table=0, n_packets=6, n_bytes=476, priority=1,i
n_port=3,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=4371.309s, table=0, n_packets=5, n_bytes=434, priority=1,i
n_port=1,dl_dst=00:00:00:00:00:03 actions=output:3
root@ade-VirtualBox:~#
```

Figure 4.13: Switch Flow table status

Tcpdump command tool was also issued on the three hosts to crosscheck the result achieved using Wireshark. Tcpdump displays the broadcast message received by the hosts excluding host h1, it also displays the ARP and ICMP echo request and reply.

The flow table showing the flow entries and the matching conditions and actions is shown in table 4.1 below.

FLOW	MATCHING CONDITION	ACTION

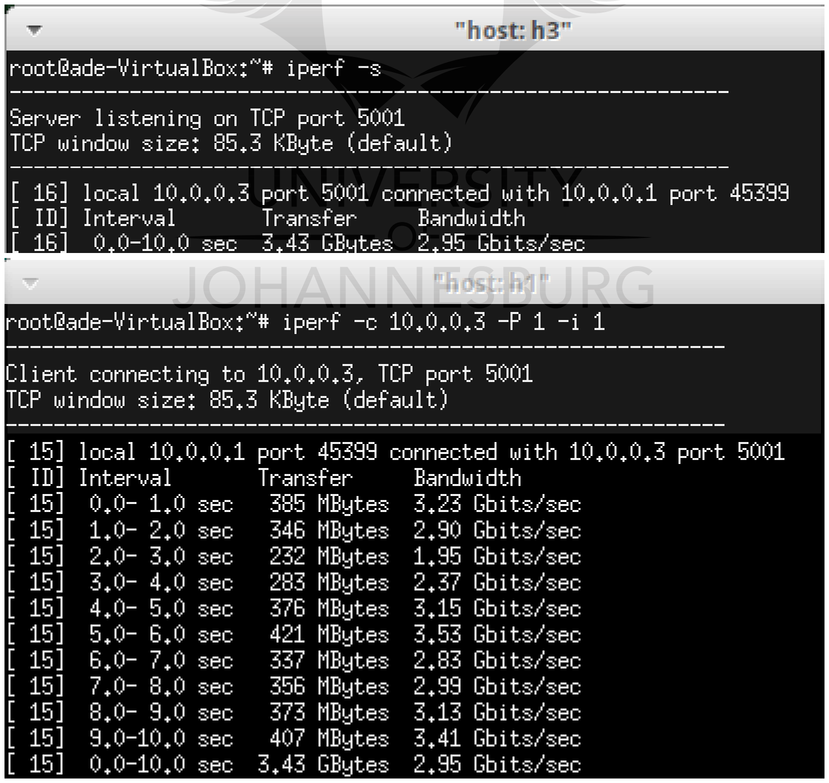
1	in_port = 3
	dl_dst = 00:00:00:00:00:01
		output = 1

2	in_port = 1
	dl_dst = 00:00:00:00:00:01
		output = 3

Table 14.2: Flow table

4.2.2 TCP Throughput

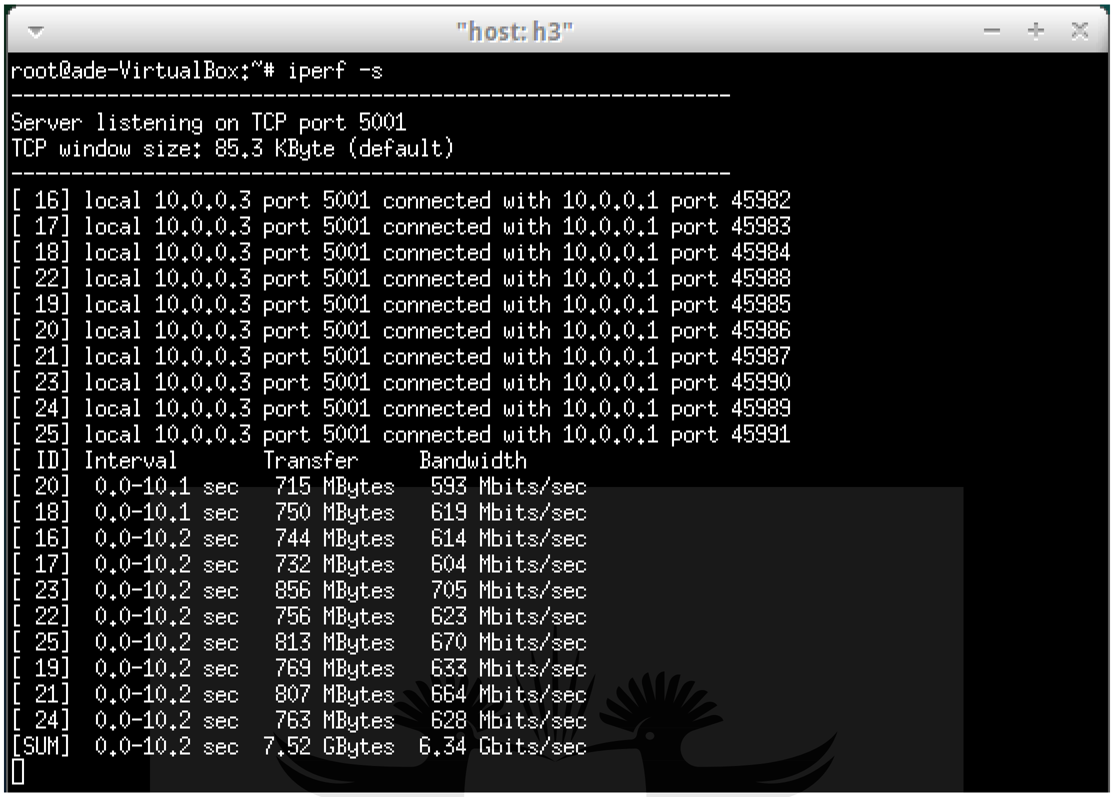
Iperf is a throughput measurement tool used to test the maximum achievable bandwidth on the network, quality of network link and latency between the hosts. The achieved results are shown in figure 4.14 and figure 4.15 below with host h3 as the server and host h1 as the client.



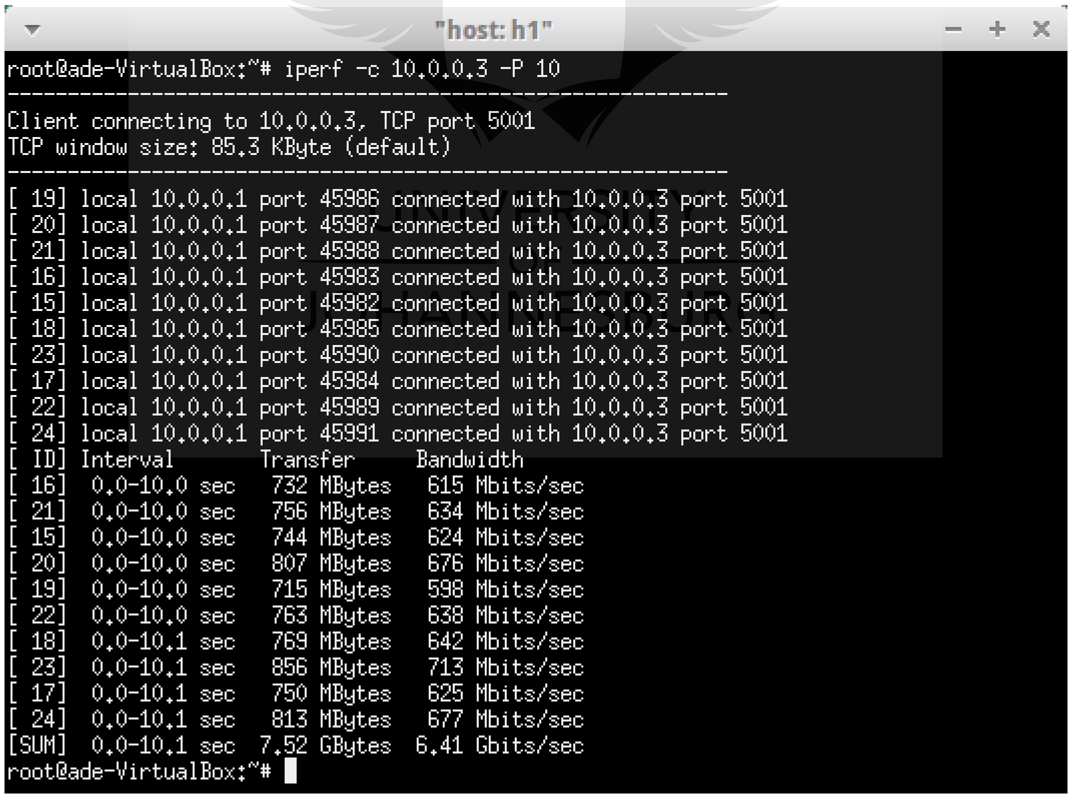
```
root@ade-VirtualBox:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45399
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-10.0 sec  3.43 GBytes  2.95 Gbits/sec
```

```
root@ade-VirtualBox:~# iperf -c 10.0.0.3 -P 1 -i 1
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.1 port 45399 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0- 1.0 sec   385 MBytes  3.23 Gbits/sec
[ 15]  1.0- 2.0 sec   346 MBytes  2.90 Gbits/sec
[ 15]  2.0- 3.0 sec   232 MBytes  1.95 Gbits/sec
[ 15]  3.0- 4.0 sec   283 MBytes  2.37 Gbits/sec
[ 15]  4.0- 5.0 sec   376 MBytes  3.15 Gbits/sec
[ 15]  5.0- 6.0 sec   421 MBytes  3.53 Gbits/sec
[ 15]  6.0- 7.0 sec   337 MBytes  2.83 Gbits/sec
[ 15]  7.0- 8.0 sec   356 MBytes  2.99 Gbits/sec
[ 15]  8.0- 9.0 sec   373 MBytes  3.13 Gbits/sec
[ 15]  9.0-10.0 sec   407 MBytes  3.41 Gbits/sec
[ 15]  0.0-10.0 sec  3.43 GBytes  2.95 Gbits/sec
```

Figure 4.14: TCP throughput with single connection



```
root@ade-VirtualBox:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45982
[ 17] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45983
[ 18] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45984
[ 22] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45988
[ 19] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45985
[ 20] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45986
[ 21] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45987
[ 23] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45990
[ 24] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45989
[ 25] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45991
[ ID] Interval       Transfer     Bandwidth
[ 20]  0.0-10.1 sec   715 MBytes   593 Mbits/sec
[ 18]  0.0-10.1 sec   750 MBytes   619 Mbits/sec
[ 16]  0.0-10.2 sec   744 MBytes   614 Mbits/sec
[ 17]  0.0-10.2 sec   732 MBytes   604 Mbits/sec
[ 23]  0.0-10.2 sec   856 MBytes   705 Mbits/sec
[ 22]  0.0-10.2 sec   756 MBytes   623 Mbits/sec
[ 25]  0.0-10.2 sec   813 MBytes   670 Mbits/sec
[ 19]  0.0-10.2 sec   769 MBytes   633 Mbits/sec
[ 21]  0.0-10.2 sec   807 MBytes   664 Mbits/sec
[ 24]  0.0-10.2 sec   763 MBytes   628 Mbits/sec
[SUM]  0.0-10.2 sec  7.52 GBytes  6.34 Gbits/sec
```



```
root@ade-VirtualBox:~# iperf -c 10.0.0.3 -P 10
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 19] local 10.0.0.1 port 45986 connected with 10.0.0.3 port 5001
[ 20] local 10.0.0.1 port 45987 connected with 10.0.0.3 port 5001
[ 21] local 10.0.0.1 port 45988 connected with 10.0.0.3 port 5001
[ 16] local 10.0.0.1 port 45983 connected with 10.0.0.3 port 5001
[ 15] local 10.0.0.1 port 45982 connected with 10.0.0.3 port 5001
[ 18] local 10.0.0.1 port 45985 connected with 10.0.0.3 port 5001
[ 23] local 10.0.0.1 port 45990 connected with 10.0.0.3 port 5001
[ 17] local 10.0.0.1 port 45984 connected with 10.0.0.3 port 5001
[ 22] local 10.0.0.1 port 45989 connected with 10.0.0.3 port 5001
[ 24] local 10.0.0.1 port 45991 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-10.0 sec   732 MBytes   615 Mbits/sec
[ 21]  0.0-10.0 sec   756 MBytes   634 Mbits/sec
[ 15]  0.0-10.0 sec   744 MBytes   624 Mbits/sec
[ 20]  0.0-10.0 sec   807 MBytes   676 Mbits/sec
[ 19]  0.0-10.0 sec   715 MBytes   598 Mbits/sec
[ 22]  0.0-10.0 sec   763 MBytes   638 Mbits/sec
[ 18]  0.0-10.1 sec   769 MBytes   642 Mbits/sec
[ 23]  0.0-10.1 sec   856 MBytes   713 Mbits/sec
[ 17]  0.0-10.1 sec   750 MBytes   625 Mbits/sec
[ 24]  0.0-10.1 sec   813 MBytes   677 Mbits/sec
[SUM]  0.0-10.1 sec  7.52 GBytes  6.41 Gbits/sec
root@ade-VirtualBox:~#
```

Figure 4.15: TCP throughput with multiple connections

Results and Discussion of the Throughput

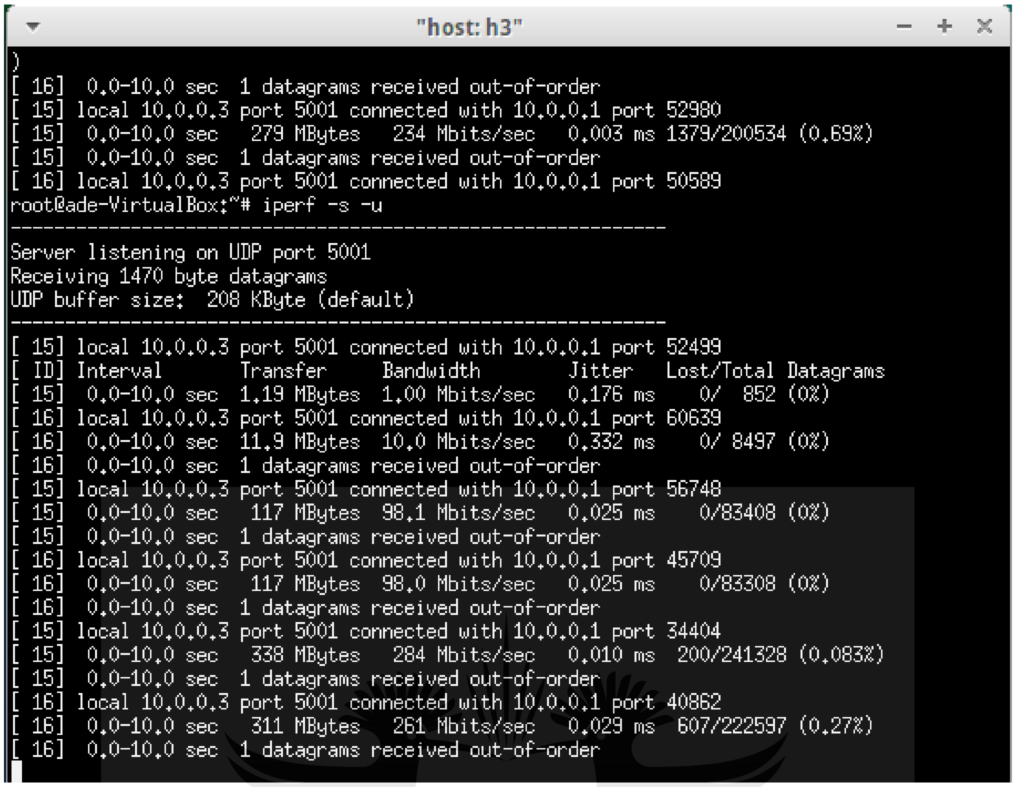
To test the throughput, host h3 was made the server by invoking iperf –s command on its terminal while host h1 was made the client with iperf –c 10.0.0.3 –P 1 –i 1 with –P 1 as option for single connection to the server and –i 1 is used to display the transfer rate at an interval of 1 second for the whole 10 seconds' transfer.

Using figure 4.14, the result obtained by transferring 3.43GBytes within 10 seconds using all the available bandwidth of 10GB was 2.95Gbits/sec. the bandwidth usage of 2.95Gbits/sec was achieved out of the 10GB bandwidth available. Figure 4.15 shows the use of iperf in creating multiple TCP connections to the server. The use of –P 10 tells host h1 to create 10 parallel TCP connection to server host h3. The 10 parallel connections were created with different port number with all connecting to port 5001 of the server. All the 10-parallel connections were shown with different data transfer and bandwidth used. The sum of the total data transfer was 7.52 GBytes at the rate of 6.41 Gbits/sec.

It can be deduced that parallel connection to the server has a higher data transfer and use higher bandwidth, which is better than using a single connection. This shows that SDN with OpenFlow can still perform the same functions the traditional applications (e.g. internet download manager) are capable of by speeding up download rate and reduce downloading time.

4.2.3 UDP Throughput

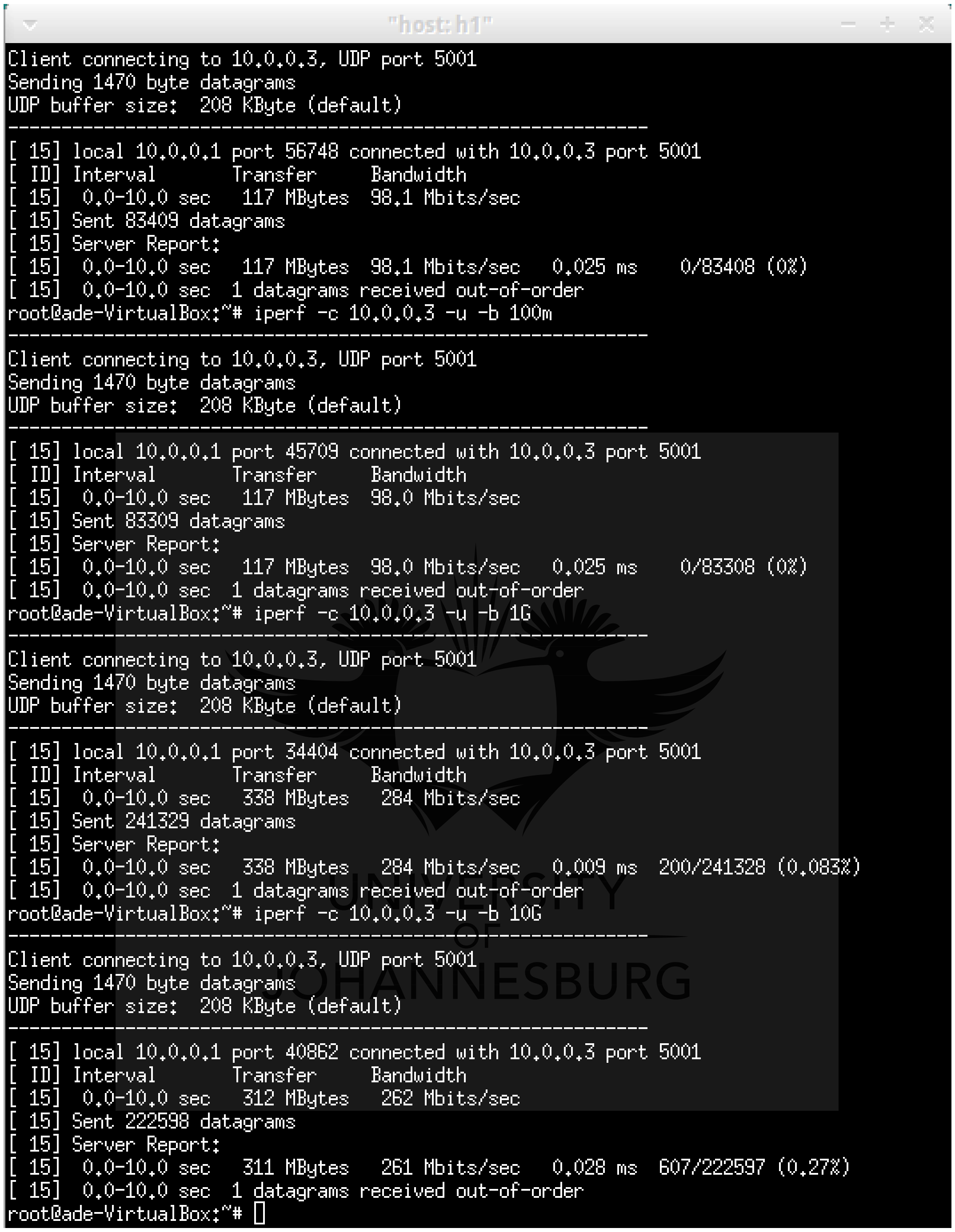
A UDP throughput was also performed using the same network used in the TCP throughput. Host h3 was also made the server while host h1 was made the client. UDP test results shown in figures display the interval of 10 seconds, transferred data, bandwidth usage, and network jitter and packet loss out of total number of packets. Using the results shown in figure 4.16 and figure 4.17, the use of 1M, 10M and 100M perform very well with no loss and very little network jitter while 1G and 10G has some packet loss with 1G having just 0.083% and 10G having a higher 0.27%.



```
"host: h3"
)
[ 16]  0.0-10.0 sec  1 datagrams received out-of-order
[ 15] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 52980
[ 15]  0.0-10.0 sec   279 MBytes   234 Mbits/sec   0.003 ms 1379/200534 (0.69%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 50589
root@ade-VirtualBox:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 52499
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 15]  0.0-10.0 sec  1.19 MBytes  1.00 Mbits/sec   0.176 ms    0/  852 (0%)
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 60639
[ 16]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.332 ms    0/ 8497 (0%)
[ 16]  0.0-10.0 sec  1 datagrams received out-of-order
[ 15] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 56748
[ 15]  0.0-10.0 sec   117 MBytes  98.1 Mbits/sec   0.025 ms    0/83408 (0%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 45709
[ 16]  0.0-10.0 sec   117 MBytes  98.0 Mbits/sec   0.025 ms    0/83308 (0%)
[ 16]  0.0-10.0 sec  1 datagrams received out-of-order
[ 15] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 34404
[ 15]  0.0-10.0 sec   338 MBytes   284 Mbits/sec   0.010 ms  200/241328 (0.083%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 40862
[ 16]  0.0-10.0 sec   311 MBytes   261 Mbits/sec   0.029 ms  607/222597 (0.27%)
[ 16]  0.0-10.0 sec  1 datagrams received out-of-order
```

Figure 4.16: UDP throughput server result

The experiment has shown that SDN with the use of OpenFlow protocol can function more like the traditional network and can even be easier to tweak for higher performance. It also shows that networking tools that are used by network and system administrator can also perform well within SDN environment.



```
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.1 port 56748 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec   117 MBytes  98.1 Mbits/sec
[ 15] Sent 83409 datagrams
[ 15] Server Report:
[ 15]  0.0-10.0 sec   117 MBytes  98.1 Mbits/sec   0.025 ms    0/83408 (0%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
root@ade-VirtualBox:~# iperf -c 10.0.0.3 -u -b 100m
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.1 port 45709 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec   117 MBytes  98.0 Mbits/sec
[ 15] Sent 83309 datagrams
[ 15] Server Report:
[ 15]  0.0-10.0 sec   117 MBytes  98.0 Mbits/sec   0.025 ms    0/83308 (0%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
root@ade-VirtualBox:~# iperf -c 10.0.0.3 -u -b 1G
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.1 port 34404 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec   338 MBytes   284 Mbits/sec
[ 15] Sent 241329 datagrams
[ 15] Server Report:
[ 15]  0.0-10.0 sec   338 MBytes   284 Mbits/sec   0.009 ms  200/241328 (0.083%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
root@ade-VirtualBox:~# iperf -c 10.0.0.3 -u -b 10G
------------------------------------------------------------
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.1 port 40862 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec   312 MBytes   262 Mbits/sec
[ 15] Sent 222598 datagrams
[ 15] Server Report:
[ 15]  0.0-10.0 sec   311 MBytes   261 Mbits/sec   0.028 ms  607/222597 (0.27%)
[ 15]  0.0-10.0 sec  1 datagrams received out-of-order
root@ade-VirtualBox:~# 
```

Figure 4.17: UDP throughput client result

4.3 Experiment 2: VLAN in SDN

VLAN is used to bread to break broadcast domain in the network by creating a smaller broadcast domain within the network. It is also used to group hosts with similar functions. As explained in section 2.3.1, VLAN can be used to break domains in the building to separate different departments in this same building. For instance, finance department and administration department staffs in the same building can be separated with the use of VLAN. VLAN uses 802.1Q for the tagging of the packets in the VLAN network. This tagging or identifier is used in recognizing each packet in the network and is also used in forwarding packet to desired destination. This experiment demonstrates a testbed with 1 controller, 5 switches and 16 hosts. The testbed is created using a tree topology as shown in figure 4.18.

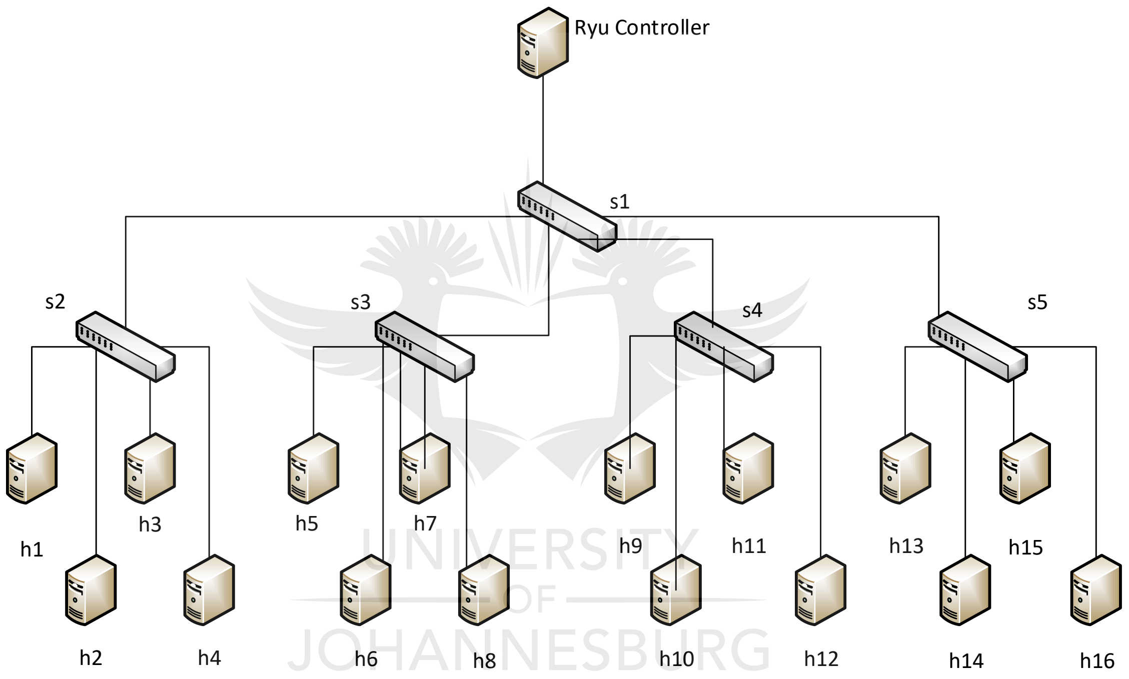
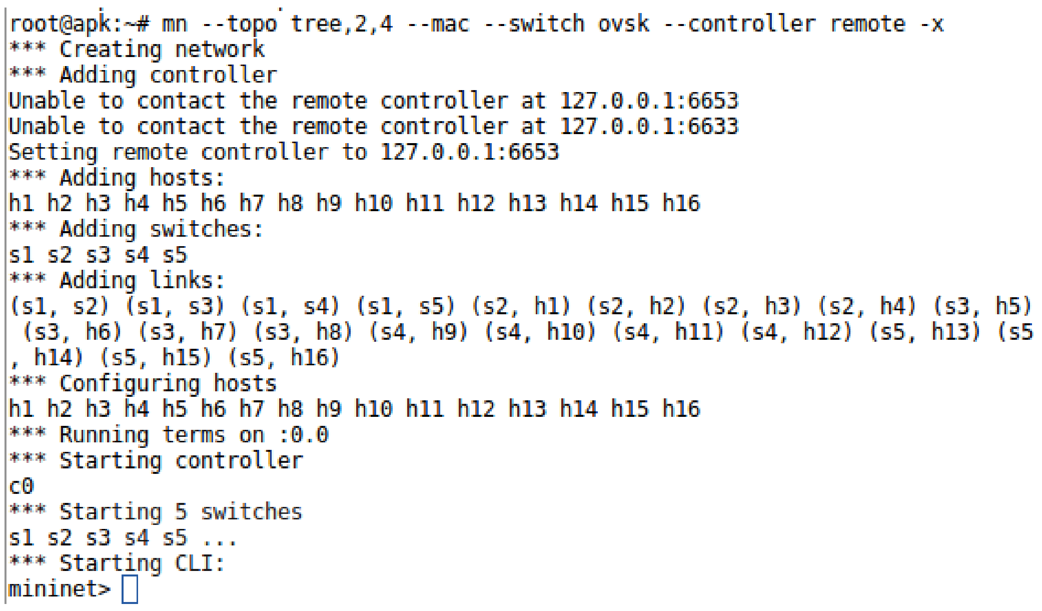


Figure 4.18: VLAN Topology

Figure 4.19 shows the Command Line Interface (CLI) of the creation of the testbed. The testbed uses tree topology and consists of five switches. Switch s1 is the central switch that connect all the remaining four switches together, namely s2, s3, s4 and s5 together. Unlike traditional tree topology that it is the sole responsibility of the core switch to forward packet to respective destination, switch s1 is also responsible for packet forwarding from one switch to the other but the remaining four switches are also connected to the controller. Though the connection to controller does not determine the packet forwarding but helps in decision making. The controller remains the brain of the network which is responsible for decision regarding packet forwarding, other functions include forwarding out packet to respective output port, send to controller for decision, or dropping the packet.



```
root@apk:~# mn --topo tree,2,4 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s3, h5)
 (s3, h6) (s3, h7) (s3, h8) (s4, h9) (s4, h10) (s4, h11) (s4, h12) (s5, h13) (s5
, h14) (s5, h15) (s5, h16)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Running terms on :0.0
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> 
```

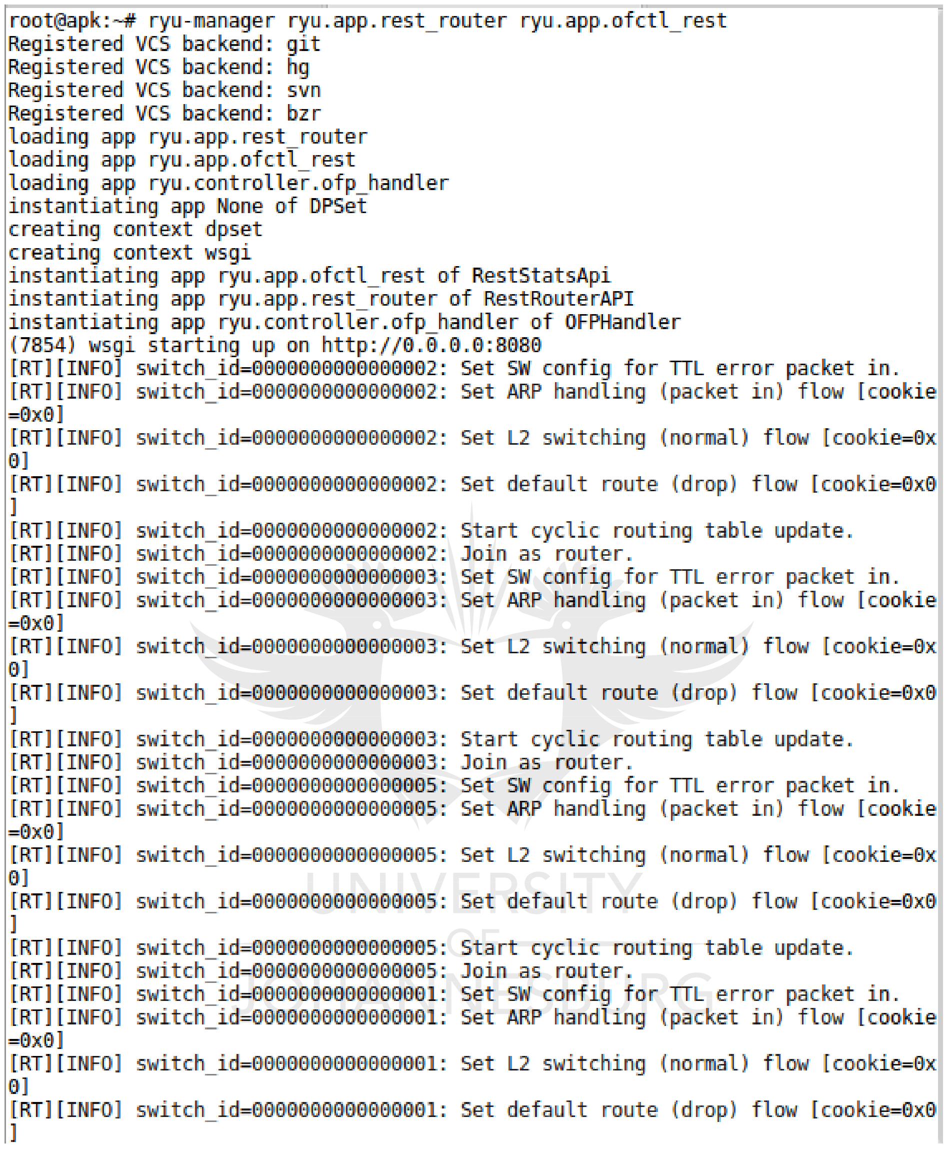
Figure 4.19: Tree topology testbed creation

After the Mininet start-up, the Ryu router application is started with ofctl_rest which makes REST and API integration possible [66]. This is shown in figure 4.20. The ofctl_rest is started with the router application for easier querying with browser or Postman. Postman is used to display the query of the JSON in a better readable format [67]. The IP addresses of the routers and hosts and other information used are shown in the table 14.2.

Host	IP Address	VLANs	Default Gateway	Switch	Switch IP address	Switch Port
				S1	172.16.10.1	
H1	192.168.100.10	10	192.168.100.1	S2	172.16.10.2	S2-eth1
H2	192.168.100.11	10	192.168.100.1	S2	172.16.10.2	S2-eth2
H3	192.168.100.12	100	192.168.100.1	S2	172.16.10.2	S2-eth3
H4	192.168.100.13	200	192.168.100.1	S2	172.16.10.2	S2-eth4
H5	192.168.200.10	10	192.168.200.1	S3	172.16.10.3	S3-eth1
H6	192.168.200.11	100	192.168.200.1	S3	172.16.10.3	S3-eth2
H7	192.168.200.12	100	192.168.200.1	S3	172.16.10.3	S3-eth3
H8	192.168.200.13	200	192.168.200.1	S3	172.16.10.3	S3-eth4
H9	192.168.10.10	10	192.168.10.1	S4	172.16.10.4	S4-eth1
H10	192.168.10.11	100	192.168.10.1	S4	172.16.10.4	S4-eth2
H11	192.168.10.12	200	192.168.10.1	S4	172.16.10.4	S4-eth3
H12	192.168.10.13	200	192.168.10.1	S4	172.16.10.4	S4-eth4
H13	192.168.20.10	10	192.168.20.1	S5	172.16.10.5	S5-eth1
H14	192.168.20.11	100	192.168.20.1	S5	172.16.10.5	S5-eth2
H15	192.168.20.12	200	192.168.20.1	S5	172.16.10.5	S5-eth3
H16	192.168.20.13	10	192.168.20.1	S5	172.16.10.5	S5-eth4

Table 14.2: VLAN IP address allocation

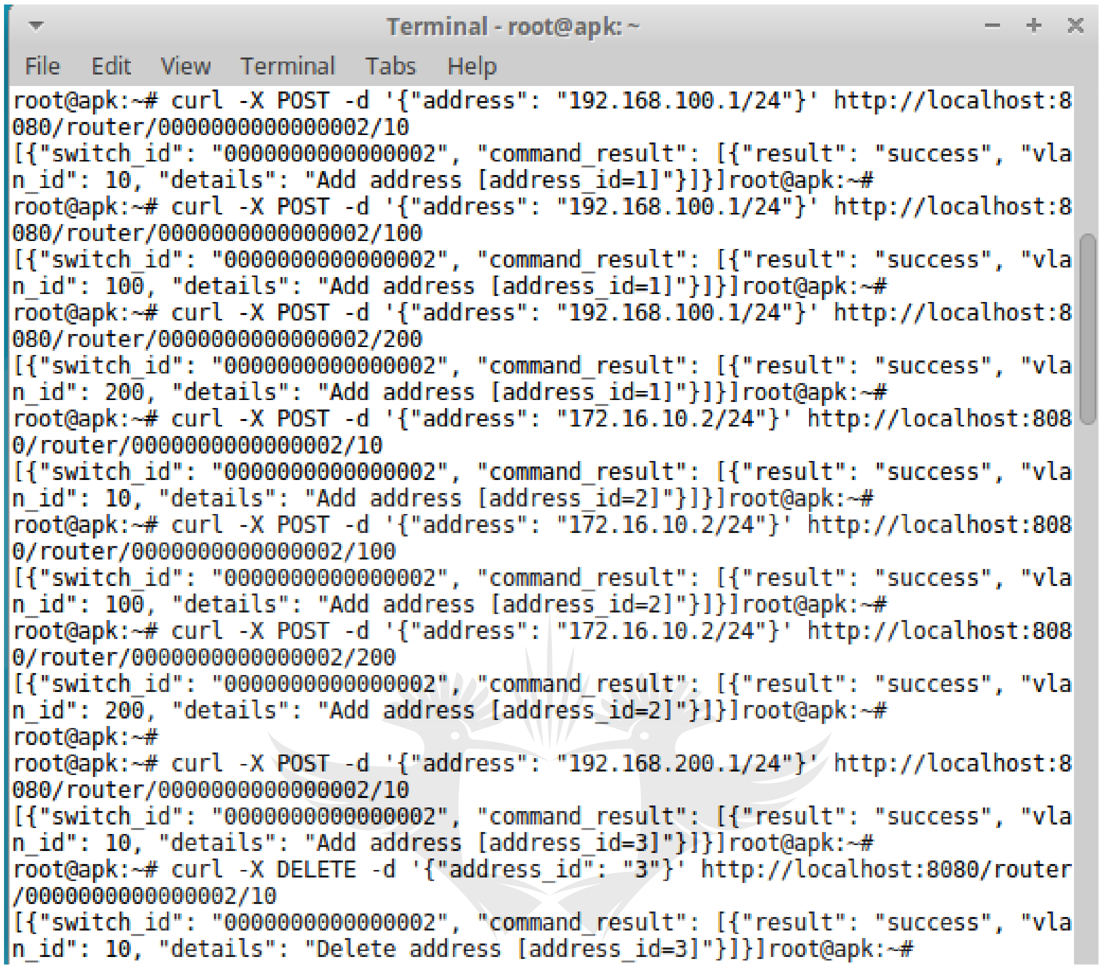
This table also shows the VLANs allocation of to each host on the network and the corresponding gateway and default route used by networking devices and hosts on the network.



```
root@apk:~# ryu-manager ryu.app.rest_router ryu.app.ofctl_rest
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app ryu.app.rest_router
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.app.rest_router of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(7854) wsgi starting up on http://0.0.0.0:8080
[RT][INFO] switch_id=0000000000000002: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000002: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000002: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000002: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000002: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000002: Join as router.
[RT][INFO] switch_id=0000000000000003: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000003: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000003: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000003: Join as router.
[RT][INFO] switch_id=0000000000000005: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000005: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000005: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000005: Set default route (drop) flow [cookie=0x0
]
[RT][INFO] switch_id=0000000000000005: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000005: Join as router.
[RT][INFO] switch_id=0000000000000001: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000001: Set ARP handling (packet in) flow [cookie
=0x0]
[RT][INFO] switch_id=0000000000000001: Set L2 switching (normal) flow [cookie=0x
0]
[RT][INFO] switch_id=0000000000000001: Set default route (drop) flow [cookie=0x0
]
```

Figure 4.20: Ryu router application start-up

Figure 4.21 shows some of the configuration of the switches on the network. cUrl is used in the command line with POST to input the information required on the switches. The result is displayed on the CLI immediately after the execution of the POST command using cUrl. The switches configuration figure shows the switch_id, the vlan_id, and the address_id of the switch being configured in the network.



```
root@apk:~# curl -X POST -d '{"address": "192.168.100.1/24"}' http://localhost:8
080/router/0000000000000002/10
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 10, "details": "Add address [address_id=1]"}]}]root@apk:~#
root@apk:~# curl -X POST -d '{"address": "192.168.100.1/24"}' http://localhost:8
080/router/0000000000000002/100
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 100, "details": "Add address [address_id=1]"}]}]root@apk:~#
root@apk:~# curl -X POST -d '{"address": "192.168.100.1/24"}' http://localhost:8
080/router/0000000000000002/200
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 200, "details": "Add address [address_id=1]"}]}]root@apk:~#
root@apk:~# curl -X POST -d '{"address": "172.16.10.2/24"}' http://localhost:808
0/router/0000000000000002/10
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 10, "details": "Add address [address_id=2]"}]}]root@apk:~#
root@apk:~# curl -X POST -d '{"address": "172.16.10.2/24"}' http://localhost:808
0/router/0000000000000002/100
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 100, "details": "Add address [address_id=2]"}]}]root@apk:~#
root@apk:~# curl -X POST -d '{"address": "172.16.10.2/24"}' http://localhost:808
0/router/0000000000000002/200
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 200, "details": "Add address [address_id=2]"}]}]root@apk:~#
root@apk:~#
root@apk:~# curl -X POST -d '{"address": "192.168.200.1/24"}' http://localhost:8
080/router/0000000000000002/10
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 10, "details": "Add address [address_id=3]"}]}]root@apk:~#
root@apk:~# curl -X DELETE -d '{"address_id": "3"}' http://localhost:8080/router
/0000000000000002/10
[{"switch_id": "0000000000000002", "command_result": [{"result": "success", "vla
n_id": 10, "details": "Delete address [address_id=3]"}]}]root@apk:~#
```

Figure 4.21: Switches address configuration

Each host on the network is also configured with the default route, this is a must for the hosts on the switches can be queried with the result viewed in JSON format using POSTMAN [67]. POSTMAN makes the visualizing of the configuration data including flow entry to be more user friendly and easily readable. Using GET command on the POSTMAN to query the status of a networking device or all device on the network. Issuing the GET command to view the status of all devices after the configuration of IP addresses to all the devices using this command: http://localhost:8080/router/all/all gives the following;

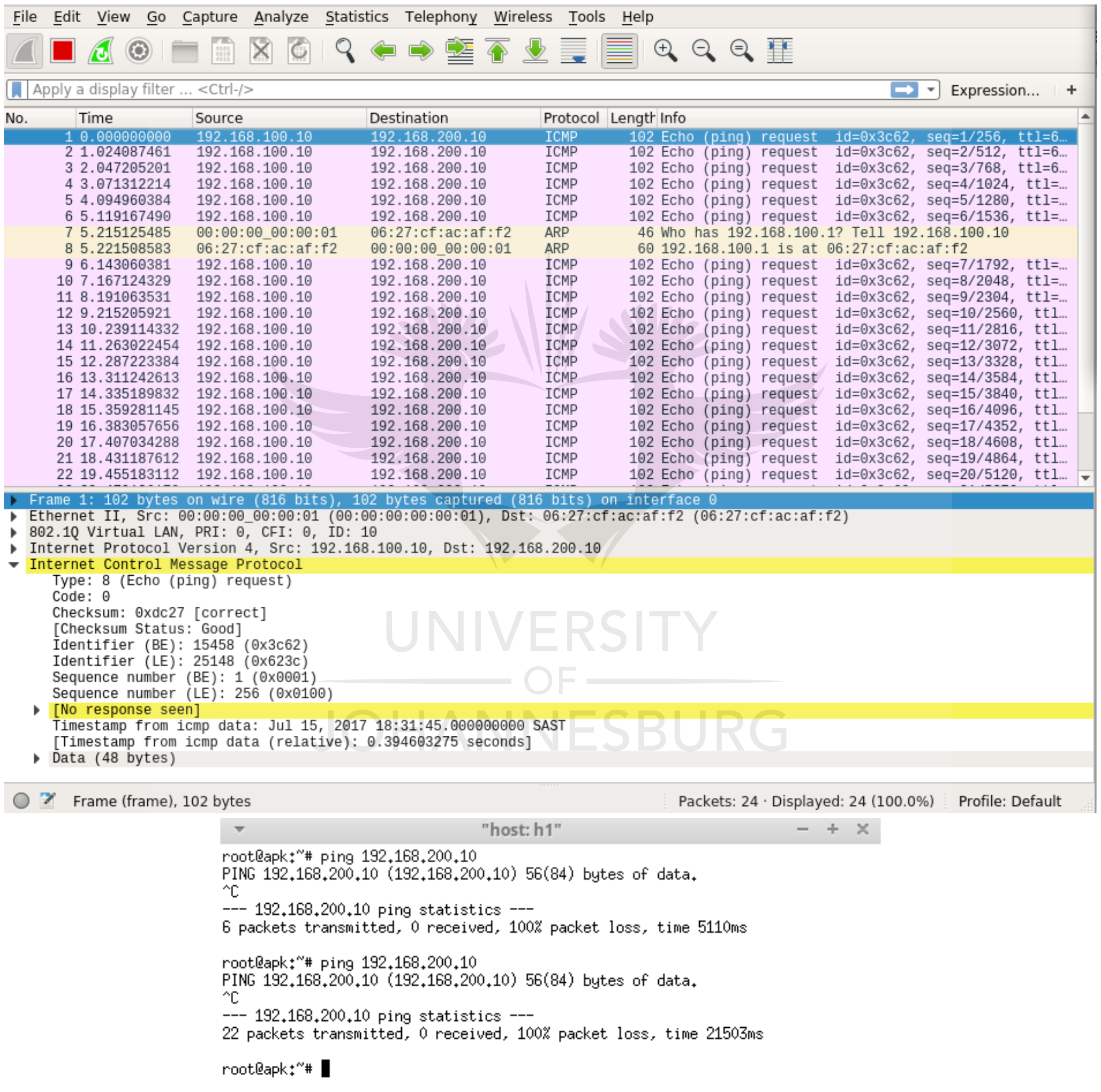
```
[
    {
        "internal_network": [
            {},
            {
                "vlan_id": 200,
                "address": [
                    {
                        "address_id": 1,
                        "address": "172.16.10.1/24"
                    }
                ]
        },
        {
            "vlan_id": 10,
            "address": [
                {
                    "address_id": 1,
                    "address": "172.16.10.1/24"
                }
            ]
        },
        {
            "vlan_id": 100,
            "address": [
                {
                    "address_id": 1,
                    "address": "172.16.10.1/24"
                }
            ]
        }
    ],
    "switch_id": "0000000000000001"
},
{
    "internal_network": [
        {},
        {
            "vlan_id": 200,
            "address": [
                {
                    "address_id": 1,
                    "address": "192.168.100.1/24"
                },
                {
                    "address_id": 2,
                    "address": "172.16.10.2/24"
                }
            ]
        },
        {
            "vlan_id": 10,
            "address": [
                {
                    "address_id": 1,
                    "address": "192.168.100.1/24"
                },
                {
                    "address_id": 2,
                    "address": "172.16.10.2/24"
                }
            ]
        },
        {
            "vlan_id": 100,
            "address": [
                {
                    "address_id": 1,
                    "address": "192.168.100.1/24"
                },
                {
                    "address_id": 2,
                    "address": "172.16.10.2/24"
                }
            ]
```

```
                    }
                ],
                "switch_id": "0000000000000002"
            },
            {
                "internal_network": [
                    {},
                    {
                        "vlan_id": 200,
                        "address": [
                            {
                                "address_id": 1,
                                "address": "192.168.200.1/24"
                            },
                            {
                                "address_id": 2,
                                "address": "172.16.10.3/24"
                            }
                        ]
                    },
                    {
                        "vlan_id": 10,
                        "address": [
                            {
                                "address_id": 1,
                                "address": "192.168.200.1/24"
                            },
                            {
                                "address_id": 2,
                                "address": "172.16.10.3/24"
                            }
                        ]
                    },
                    {
                        "vlan_id": 100,
                        "address": [
                            {
                                "address_id": 1,
                                "address": "192.168.200.1/24"
                            },
                            {
                                "address_id": 2,
                                "address": "172.16.10.3/24"
                            }
                        ]
                    }
                ],
                "switch_id": "0000000000000003"
            },
            {
                "internal_network": [
                    {},
                    {
                        "vlan_id": 200,
                        "address": [
                            {
                                "address_id": 1,
                                "address": "192.168.10.1/24"
                            },
                            {
                                "address_id": 2,
                                "address": "172.16.10.4/24"
                            }
                ]
            },
            {
                "vlan_id": 10,
                "address": [
                    {
                        "address_id": 1,
                        "address": "192.168.10.1/24"
                    },
                    {
                        "address_id": 2,
                        "address": "172.16.10.4/24"
                    }
                ]
            },
            {
                "vlan_id": 100,
                "address": [
                    {
                        "address_id": 1,
                        "address": "192.168.10.1/24"
                    },
                    {
                        "address_id": 2,
                        "address": "172.16.10.4/24"
                    }
                ]
            }
        ],
        "switch_id": "0000000000000004"
    },
    {
        "internal_network": [
            {},
            {
                "vlan_id": 200,
                "address": [
                    {
                        "address_id": 1,
                        "address": "192.168.20.1/24"
                    },
                    {
                        "address_id": 2,
                        "address": "172.16.10.5/24"
                    }
                ]
            },
            {
                "vlan_id": 10,
                "address": [
                    {
                        "address_id": 1,
                        "address": "192.168.20.1/24"
                    },
                    {
                        "address_id": 2,
                        "address": "172.16.10.5/24"
                    }
                ]
            },
            {
                "vlan_id": 100,
                "address": [
                    {
                        "address_id": 1,
                        "address": "192.168.20.1/24"
                    },
                    {
                        "address_id": 2,
                        "address": "172.16.10.5/24"
                    }
                ]
            }
        ],
        "switch_id": "0000000000000005"
    }
]
```

The three VLANs created are VLAN 10, VLAN 100, and VLAN 200. From table 4.2, each host on the switches are allocated into specific VLAN. The status shows the addresses on all the devices with respective VLAN allocations. To get status of each device, the switch_id or dpid of the device must be specified and the VLAN_id can also be specified as well to get data of interest. The data from switch s1 with VLAN 10 is shown below by using the command: http://localhost:8080/router/0000000000000001/10 .

```
[
    {
        "internal_network": [
            {},
            {
                "vlan_id": 200,
                "address": [
                    {
                        "address_id": 1,
                        "address": "172.16.10.1/24"
                    }
                ]
            },
            {
                "vlan_id": 10,
                "address": [
                    {
                        "address_id": 1,
                        "address": "172.16.10.1/24"
                    }
                ]
            },
            {
                "vlan_id": 100,
                "address": [
                    {
                        "address_id": 1,
                        "address": "172.16.10.1/24"
                    }
                ]
            }
        ],
        "switch_id": "0000000000000001"
    },
]
```

Pinging between hosts on the same VLAN on the same switch is successful while pinging between hosts on different VLAN on the same switch is not possible. Before the configuration of default route on the hosts, pinging between hosts on the same VLAN located on different switches is not successful. This can be seen in figure 4.22. Inter-switch routing is possible after the configuration of default route on the host and configuration of IP addresses with default route and static route with respective destination network addresses on the switches. Figure 4.23 shows successful communication between host h1 on VLAN 10 located on switch s1 and host h5 on VLAN 10 located on switch s2.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
"host: h1"

```
No.  Time          Source             Destination        Protocol Length Info
   1 0.000000000   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=1/256, ttl=6…
   2 1.024087461   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=2/512, ttl=6…
   3 2.047205201   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=3/768, ttl=6…
   4 3.071312214   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=4/1024, ttl=…
   5 4.094960384   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=5/1280, ttl=…
   6 5.119167490   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=6/1536, ttl=…
   7 5.215125485   00:00:00_00:00:01  06:27:cf:ac:af:f2  ARP       46 Who has 192.168.100.1? Tell 192.168.100.10
   8 5.221508583   06:27:cf:ac:af:f2  00:00:00_00:00:01  ARP       60 192.168.100.1 is at 06:27:cf:ac:af:f2
   9 6.143060381   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=7/1792, ttl=…
  10 7.167124329   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=8/2048, ttl=…
  11 8.191063531   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=9/2304, ttl=…
  12 9.215205921   192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=10/2560, ttl…
  13 10.239114332  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=11/2816, ttl…
  14 11.263022454  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=12/3072, ttl…
  15 12.287223384  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=13/3328, ttl…
  16 13.311242613  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=14/3584, ttl…
  17 14.335189832  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=15/3840, ttl…
  18 15.359281145  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=16/4096, ttl…
  19 16.383057656  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=17/4352, ttl…
  20 17.407034288  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=18/4608, ttl…
  21 18.431187612  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=19/4864, ttl…
  22 19.455183112  192.168.100.10     192.168.200.10     ICMP     102 Echo (ping) request  id=0x3c62, seq=20/5120, ttl…

▸ Frame 1: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 06:27:cf:ac:af:f2 (06:27:cf:ac:af:f2)
▸ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
▸ Internet Protocol Version 4, Src: 192.168.100.10, Dst: 192.168.200.10
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xdc27 [correct]
    [Checksum Status: Good]
    Identifier (BE): 15458 (0x3c62)
    Identifier (LE): 25148 (0x623c)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
  ▸ [No response seen]
    Timestamp from icmp data: Jul 15, 2017 18:31:45.000000000 SAST
    [Timestamp from icmp data (relative): 0.394603275 seconds]
  ▸ Data (48 bytes)

Frame (frame), 102 bytes        Packets: 24 · Displayed: 24 (100.0%)    Profile: Default
```

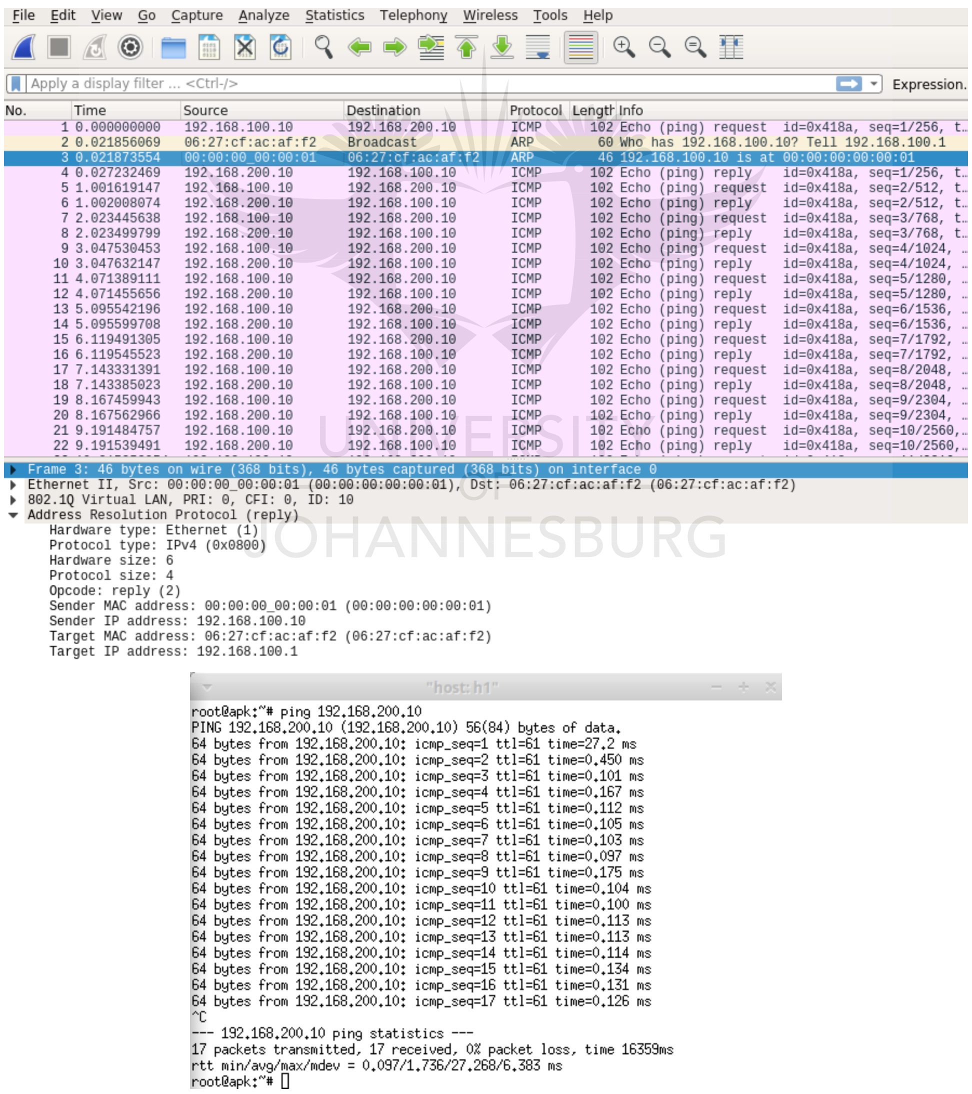
```
root@apk:~# ping 192.168.200.10
PING 192.168.200.10 (192.168.200.10) 56(84) bytes of data.
^C
--- 192.168.200.10 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5110ms

root@apk:~# ping 192.168.200.10
PING 192.168.200.10 (192.168.200.10) 56(84) bytes of data.
^C
--- 192.168.200.10 ping statistics ---
22 packets transmitted, 0 received, 100% packet loss, time 21503ms

root@apk:~# █
```

Figure 4.22: Result of failed inter-switch connection

Deleting the default route on switch s1 will make the communication between the hosts on switch s1 and switch s2 to be unsuccessful. By examining the three inter-switch results in figure 4.23, 4.24, 4.25 and 4.26 the results show that the broadcast sent is asking for the default gateway and not the destination network address. This is because the default gateway knows the address of the inter-connected address and knows how to forward the packet in line with feedback from the controller. Before any packet can leave the switch the need to know the route and switch s1 is the core switch responsible for forwarding of all the inter-switch packet in the network. Even though all the switches are configured with addresses, an ARP message is still sent to locate the default gateway address of the switch from which the packet is being sent. Once this is done, the core switch s1 will then perform the remaining operation by forwarding the received packet from one switch to the other.

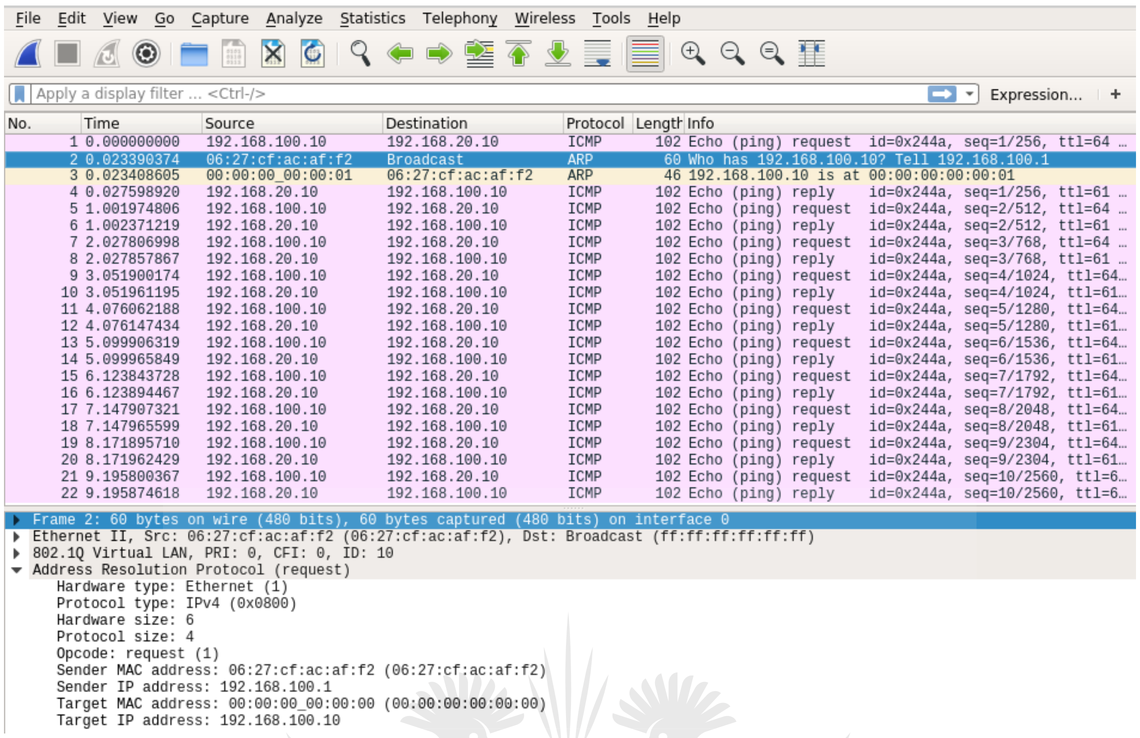


```
No.  Time         Source              Destination        Protocol Length Info
   1 0.000000000  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=1/256, t…
   2 0.021856069  06:27:cf:ac:af:f2   Broadcast          ARP       60 Who has 192.168.100.10? Tell 192.168.100.1
   3 0.021873554  00:00:00_00:00:01   06:27:cf:ac:af:f2  ARP       46 192.168.100.10 is at 00:00:00:00:00:01
   4 0.027232469  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=1/256, t…
   5 1.001619147  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=2/512, t…
   6 1.002008074  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=2/512, t…
   7 2.023445638  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=3/768, t…
   8 2.023499799  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=3/768, t…
   9 3.047530453  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=4/1024, …
  10 3.047632147  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=4/1024, …
  11 4.071389111  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=5/1280, …
  12 4.071455656  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=5/1280, …
  13 5.095542196  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=6/1536, …
  14 5.095599708  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=6/1536, …
  15 6.119491305  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=7/1792, …
  16 6.119545523  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=7/1792, …
  17 7.143331391  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=8/2048, …
  18 7.143385023  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=8/2048, …
  19 8.167459943  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=9/2304, …
  20 8.167562966  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=9/2304, …
  21 9.191484757  192.168.100.10      192.168.200.10     ICMP     102 Echo (ping) request  id=0x418a, seq=10/2560,…
  22 9.191539491  192.168.200.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x418a, seq=10/2560,…

▸ Frame 3: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 06:27:cf:ac:af:f2 (06:27:cf:ac:af:f2)
▸ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
    Sender IP address: 192.168.100.10
    Target MAC address: 06:27:cf:ac:af:f2 (06:27:cf:ac:af:f2)
    Target IP address: 192.168.100.1
```

```
root@apk:~# ping 192.168.200.10
PING 192.168.200.10 (192.168.200.10) 56(84) bytes of data.
64 bytes from 192.168.200.10: icmp_seq=1 ttl=61 time=27.2 ms
64 bytes from 192.168.200.10: icmp_seq=2 ttl=61 time=0.450 ms
64 bytes from 192.168.200.10: icmp_seq=3 ttl=61 time=0.101 ms
64 bytes from 192.168.200.10: icmp_seq=4 ttl=61 time=0.167 ms
64 bytes from 192.168.200.10: icmp_seq=5 ttl=61 time=0.112 ms
64 bytes from 192.168.200.10: icmp_seq=6 ttl=61 time=0.105 ms
64 bytes from 192.168.200.10: icmp_seq=7 ttl=61 time=0.103 ms
64 bytes from 192.168.200.10: icmp_seq=8 ttl=61 time=0.097 ms
64 bytes from 192.168.200.10: icmp_seq=9 ttl=61 time=0.175 ms
64 bytes from 192.168.200.10: icmp_seq=10 ttl=61 time=0.104 ms
64 bytes from 192.168.200.10: icmp_seq=11 ttl=61 time=0.100 ms
64 bytes from 192.168.200.10: icmp_seq=12 ttl=61 time=0.113 ms
64 bytes from 192.168.200.10: icmp_seq=13 ttl=61 time=0.113 ms
64 bytes from 192.168.200.10: icmp_seq=14 ttl=61 time=0.114 ms
64 bytes from 192.168.200.10: icmp_seq=15 ttl=61 time=0.134 ms
64 bytes from 192.168.200.10: icmp_seq=16 ttl=61 time=0.131 ms
64 bytes from 192.168.200.10: icmp_seq=17 ttl=61 time=0.126 ms
^C
--- 192.168.200.10 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16359ms
rtt min/avg/max/mdev = 0.097/1.736/27.268/6.383 ms
root@apk:~# 
```

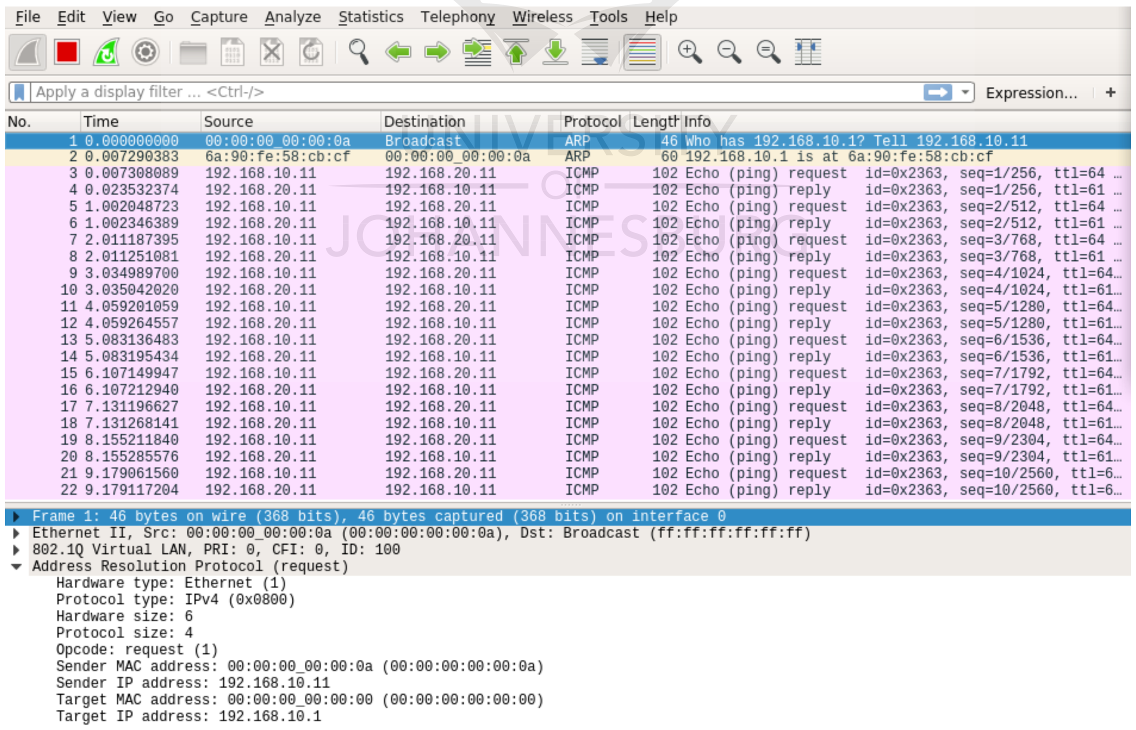
Figure 4.23: Successful VLAN inters-switch communication


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=1/256, ttl=64 …
   2 0.023390374  06:27:cf:ac:af:f2  Broadcast          ARP       60 Who has 192.168.100.10? Tell 192.168.100.1
   3 0.023408605  00:00:00_00:00:01  06:27:cf:ac:af:f2  ARP       46 192.168.100.10 is at 00:00:00:00:00:01
   4 0.027598920  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=1/256, ttl=61 …
   5 1.001974806  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=2/512, ttl=64 …
   6 1.002371219  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=2/512, ttl=61 …
   7 2.027806998  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=3/768, ttl=64 …
   8 2.027857867  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=3/768, ttl=61 …
   9 3.051900174  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=4/1024, ttl=64…
  10 3.051961195  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=4/1024, ttl=61…
  11 4.076062188  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=5/1280, ttl=64…
  12 4.076147434  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=5/1280, ttl=61…
  13 5.099906319  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=6/1536, ttl=64…
  14 5.099965849  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=6/1536, ttl=61…
  15 6.123843728  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=7/1792, ttl=64…
  16 6.123894467  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=7/1792, ttl=61…
  17 7.147907321  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=8/2048, ttl=64…
  18 7.147965599  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=8/2048, ttl=61…
  19 8.171895710  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=9/2304, ttl=64…
  20 8.171962429  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=9/2304, ttl=61…
  21 9.195800367  192.168.100.10     192.168.20.10      ICMP     102 Echo (ping) request  id=0x244a, seq=10/2560, ttl=6…
  22 9.195874618  192.168.20.10      192.168.100.10     ICMP     102 Echo (ping) reply    id=0x244a, seq=10/2560, ttl=6…
```

```
▸ Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: 06:27:cf:ac:af:f2 (06:27:cf:ac:af:f2), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 06:27:cf:ac:af:f2 (06:27:cf:ac:af:f2)
    Sender IP address: 192.168.100.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.100.10
```

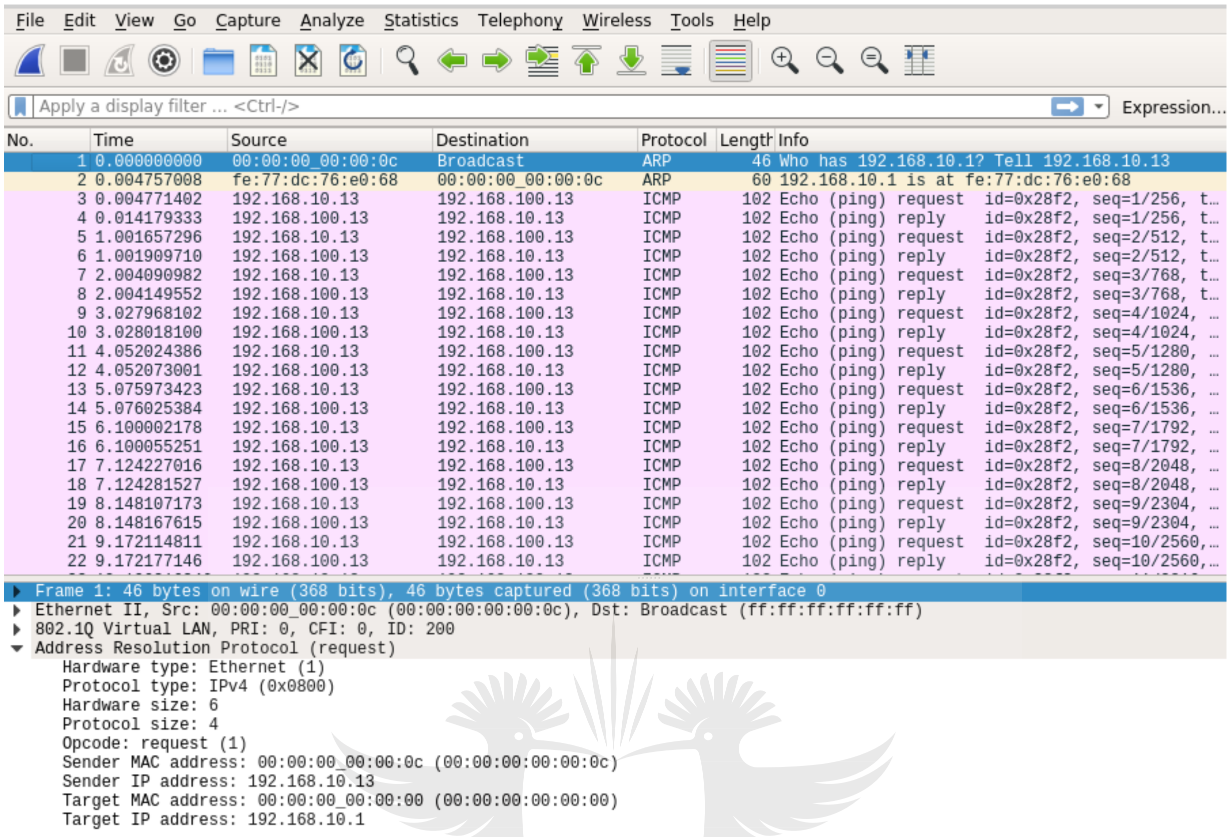
Figure 4.24: Successful communication between host h1(on s1) and h13 (on s5) with VLAN 10


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  00:00:00_00:00:0a  Broadcast          ARP       46 Who has 192.168.10.1? Tell 192.168.10.11
   2 0.007290383  6a:90:fe:58:cb:cf  00:00:00_00:00:0a  ARP       60 192.168.10.1 is at 6a:90:fe:58:cb:cf
   3 0.007308089  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=1/256, ttl=64 …
   4 0.023532374  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=1/256, ttl=61 …
   5 1.002048723  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=2/512, ttl=64 …
   6 1.002346389  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=2/512, ttl=61 …
   7 2.011187395  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=3/768, ttl=64 …
   8 2.011251081  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=3/768, ttl=61 …
   9 3.034989700  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=4/1024, ttl=64…
  10 3.035042020  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=4/1024, ttl=61…
  11 4.059201059  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=5/1280, ttl=64…
  12 4.059264557  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=5/1280, ttl=61…
  13 5.083136483  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=6/1536, ttl=64…
  14 5.083195434  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=6/1536, ttl=61…
  15 6.107149947  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=7/1792, ttl=64…
  16 6.107212940  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=7/1792, ttl=61…
  17 7.131196627  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=8/2048, ttl=64…
  18 7.131268141  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=8/2048, ttl=61…
  19 8.155211840  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=9/2304, ttl=64…
  20 8.155285576  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=9/2304, ttl=61…
  21 9.179061560  192.168.10.11      192.168.20.11      ICMP     102 Echo (ping) request  id=0x2363, seq=10/2560, ttl=6…
  22 9.179117204  192.168.20.11      192.168.10.11      ICMP     102 Echo (ping) reply    id=0x2363, seq=10/2560, ttl=6…
```

```
▸ Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:0a (00:00:00:00:00:0a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 100
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:00:00_00:00:0a (00:00:00:00:00:0a)
    Sender IP address: 192.168.10.11
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.10.1
```

Figure 4.25: Successful communication between host h10 (on s4) and h14 (on s5) with VLAN100


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time          Source              Destination        Protocol Length Info
    1 0.000000000   00:00:00_00:00:0c   Broadcast          ARP       46 Who has 192.168.10.1? Tell 192.168.10.13
    2 0.004757008   fe:77:dc:76:e0:68   00:00:00_00:00:0c  ARP       60 192.168.10.1 is at fe:77:dc:76:e0:68
    3 0.004771402   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=1/256, t…
    4 0.014179333   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=1/256, t…
    5 1.001657296   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=2/512, t…
    6 1.001909710   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=2/512, t…
    7 2.004090982   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=3/768, t…
    8 2.004149552   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=3/768, t…
    9 3.027968102   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=4/1024, …
   10 3.028018100   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=4/1024, …
   11 4.052024386   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=5/1280, …
   12 4.052073001   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=5/1280, …
   13 5.075973423   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=6/1536, …
   14 5.076025384   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=6/1536, …
   15 6.100002178   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=7/1792, …
   16 6.100055251   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=7/1792, …
   17 7.124227016   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=8/2048, …
   18 7.124281527   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=8/2048, …
   19 8.148107173   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=9/2304, …
   20 8.148167615   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=9/2304, …
   21 9.172114811   192.168.10.13       192.168.100.13     ICMP     102 Echo (ping) request  id=0x28f2, seq=10/2560,…
   22 9.172177146   192.168.100.13      192.168.10.13      ICMP     102 Echo (ping) reply    id=0x28f2, seq=10/2560,…

▶ Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:0c (00:00:00:00:00:0c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▶ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 200
▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:00:00_00:00:0c (00:00:00:00:00:0c)
    Sender IP address: 192.168.10.13
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.10.1
```

Figure 4.26: Successful communication between host h12 (on s4) and h4 (on s1) with VLAN 200

4.4 Conclusion

The experiment shows the use and functionality of VLAN in SDN by testing its use in a tree topology testbed. The results show VLAN communication within a certain VLAN for example communication on VLAN 10 on switch s1 is possible while it also shows that inter-switch VLAN communication is also possible within all the four switches s2, s3, s4, and s5. We also found out that broadcast ARP message sent out for inter-switch VLAN communication is directed to the default gateway of the packet origin. This is used by the source host address to know the route to send the packet out of. The default gateway replied the message with its MAC address and the packets are then forwarded out. Subsequent communications between the hosts on the network residing in the same VLAN are then transmitted without any delay.

5. FIREWALL AND QoS IN SDN

5.1 Firewall

Firewall is a network security system used to monitor and control access to a network environment [68]. A firewall may either be a dedicated hardware or software, and it is primarily used in an organisation to secure and to prevent unauthorised access to the network based on predetermined security rules [68], [69], [70]. It controls traffic flowing in and out of a network by granting or denying permission to packets flowing in the network [71]. The rules or policies set up for a firewall may include parameters such as IP address, MAC address, port or protocol. Permission within a network or between two networks is only given to a packet that matches information in the Access Control Lists (ACL). The firewall serves as a security barrier by protecting the network resources from unwanted access or illegal visitors.

Firewall is mostly used on the network to secure traffic flowing in and out of the network. To implement firewall in SDN, OpenFlow protocol is used and enabled on both the Ryu controller and the associated network devices in the network. In this experiment, the Ryu firewall application is used and uses the OpenFlow protocol to manipulate the forwarding table of the networking devices which is also referred to as datapath by remotely inserting, modifying or deleting flow entries on the flow table residing in each device [72], [48]**.** Flows can be added to the flow table either proactively or reactively. In this experiment, the flows were added to the flow table proactively. This means the flow entries are added to the flow table before the arrival of the packets. The flow entries added contain information to be used by the switch for forwarding. The forwarding decision is made based on the result of the matching. The flow entries consist of fields that are matched with incoming packets and the associated actions with each match are executed. This process is done by analysing the packet header of the incoming packet against the flow entries which are installed in the flow table. Packets are forwarded when a match is otherwise it may be dropped or forwarded to the controller for further processing.

The experiments includes the ability to add, modify and delete rules to be utilised in the firewall and uses one OpenvSwitch [51] and eight hosts in conjunction with the Ryu controller [43], [66]. In addition, we also discussed the implementation of a firewall at layer 2, layer 3 and layer 4 of the OSI model. In achieving the objective of this paper, some of the match fields utilised in our experiments and the network layer at which they are applicable are listed in table 5.1.

Match Fields	**Layer**
TCP, UDP, ICMP	Transport layer 4
IPv4, ARP	Network layer 3
MAC address	Data Link layer 2

Table 5.1: Match Fields

Different scenarios were tested and documented. The flow entries entered contain firewall rules in the form of Access Control Lists (ACLs). The experiment also explores the implementation of firewall in SDN at layer 2, layer 3 and layer 4. The scenario investigated will include the following functionalities of firewall in SDN:

- Allowing or blocking ICMP traffic
- Allowing ICMP traffic while blocking HTTP
- Blocking certain hosts from accessing a web server
- Blocking or allowing hosts to communicate based on source or destination IP address.

The network topology used is shown in figure 5 and consists of 1 controller, 1 OpenFlow switch (OpenvSwitch) and 8 hosts and is created with Mininet using the following command:

```
sudo mn -topo single,8 -mac -switch ovsk - controller remote -x
```

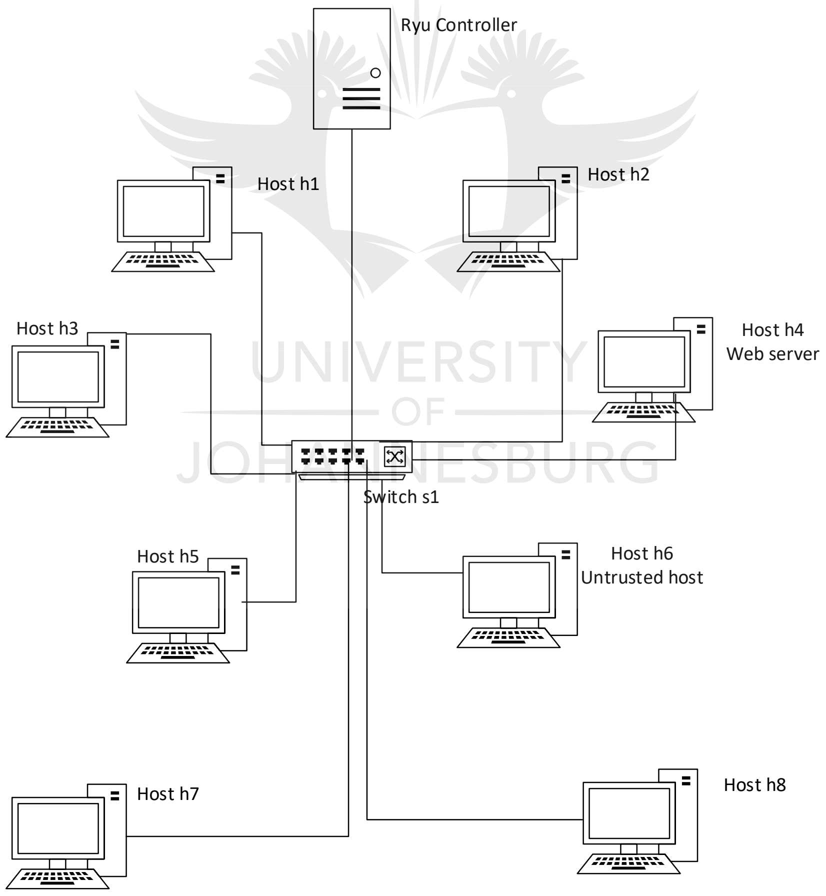


Figure 5: Topology setup

Switch s1 is set to use OpenFlow 1.3 specification using this command: `ovs-vsctl set Bridge s1 protocols=OpenFlow13`. The following command is also issued to start the firewall application: `ryu-manager ryu.app.rest_firewall ryu.app.ofctl_rest`. The `ofctl_rest` application started with the firewall allows easy management and viewing of the network and device configurations in JSON format using POSTMAN. POSTMAN is used for easy readability of the rules and status of the switches on the network. The configured firewall is enabled on the switch using: `curl -X PUT` http://localhost:8080/firewall/ module/enable/0000000000000001. The state of the switch is obtained through Postman, using: `http://localhost:8080/firewall/module/status`

Four scenarios are used to test the functionality of SDN. The first three scenarios make use of the topology shown in figure 5. with single switch while the fourth scenario uses 4 switches with 4 VLANs to demonstrate real scenario like a University campus with building that has different departmental functions (e.g Administration, finance, security)

5.1.1 Scenario 1

The first aspect of the experiments investigates the operation of the firewall at layer 2 using the MAC address of hosts. The rules setup first allows the hosts to ping each other. This is achieved by specifying a flow entry to allow communication from host h1 to h2 and vice versa by specifying their MAC addresses in the *sourceMAC* and *destinationMAC* using the following command format.

```
curl -X POST -d '{"dl_src": "sourceMAC", "dl_dst": "destinationMAC", "nw_proto": "ICMP", "actions": "ALLOW", "priority": "200"}' http://localhost:8080/firewall/rules/0000000000000001
```

This rules specifies that communication from the specified source MAC address to the specified destination MAC address should be allowed, with a given priority value when the ICMP protocol is used. The rules were then updated to disallow communication between the two hosts. This is also achieved by specifying a new flow entry to disallow communication from h1 to h2 and vice versa using the command formats:

- `curl -X POST -d '{"dl_src": "sourceMAC", "dl_dst": "destinationMAC", "nw_proto": "ICMP", "actions": "DENY", "priority": "201"}' http://localhost:8080/firewall/rules/0000000000000001`

To ensure that the new rules take precedence over the previous rule, a higher priority value is given to the new flow entries to disallow communication between the two hosts. This is confirmed by pinging both hosts, which shows that they were unreachable from each other.

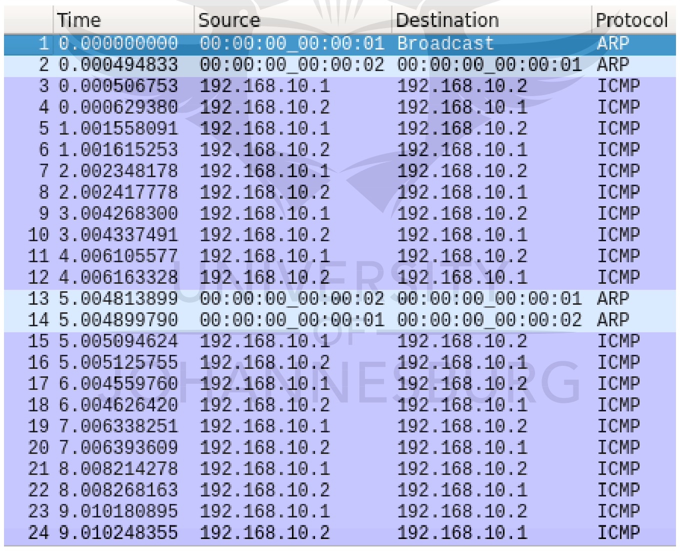
5.1.2 Scenario 2

Firewall rules used in this experiment is based on layer 3 IP address and layer 4 protocols. Firewall rules were added to allowing pinging using the ICMP messages between host h1 and host h2. The network source address and the destination address is used while permission is given to allow

ICMP network protocol communication between the two hosts. The format for the command used is as follows:

```
curl -X POST -d '{"nw_src": "sourceIP", "nw_dst": "destinationIP", "nw_proto": "ICMP",
"actions":              "ALLOW",              "priority":              "205"}'
http://localhost:8080/firewall/rules/0000000000000001
```

Two commands are entered with source and destination IP address and then interchanged the source with destination address on the second command. Priority number is also increased to overwrite the existing rule used in the earlier experiment. Figure 5.1 shows the ARP and the ICMP requests and response as the packets flow through the network after executing ten ping requests. Even though the flow entry has already been proactively added to the flow table earlier before, a broadcast is still sent using the Address Resolution Protocol (ARP) followed by response from the concerned host. The use of proactive flow entry makes the ARP response to be faster than reactive flow entry as seen on the pinging results. It is also seen that new ARP message is sent after the fifth reply message is sent after the fifth reply message, but after analysing the packet, we discover that these ARP message does not consume bandwidth. This message serves as a keep live message sent by host h2 to host h1to confirm if host h1 is still communicating.



```
   Time          Source              Destination         Protocol
 1 0.000000000   00:00:00_00:00:01   Broadcast           ARP
 2 0.000494833   00:00:00_00:00:02   00:00:00_00:00:01   ARP
 3 0.000506753   192.168.10.1        192.168.10.2        ICMP
 4 0.000629380   192.168.10.2        192.168.10.1        ICMP
 5 1.001558091   192.168.10.1        192.168.10.2        ICMP
 6 1.001615253   192.168.10.2        192.168.10.1        ICMP
 7 2.002348178   192.168.10.1        192.168.10.2        ICMP
 8 2.002417778   192.168.10.2        192.168.10.1        ICMP
 9 3.004268300   192.168.10.1        192.168.10.2        ICMP
10 3.004337491   192.168.10.2        192.168.10.1        ICMP
11 4.006105577   192.168.10.1        192.168.10.2        ICMP
12 4.006163328   192.168.10.2        192.168.10.1        ICMP
13 5.004813899   00:00:00_00:00:02   00:00:00_00:00:01   ARP
14 5.004899790   00:00:00_00:00:01   00:00:00_00:00:02   ARP
15 5.005094624   192.168.10.1        192.168.10.2        ICMP
16 5.005125755   192.168.10.2        192.168.10.1        ICMP
17 6.004559760   192.168.10.1        192.168.10.2        ICMP
18 6.004626420   192.168.10.2        192.168.10.1        ICMP
19 7.006338251   192.168.10.1        192.168.10.2        ICMP
20 7.006393609   192.168.10.2        192.168.10.1        ICMP
21 8.008214278   192.168.10.1        192.168.10.2        ICMP
22 8.008268163   192.168.10.2        192.168.10.1        ICMP
23 9.010180895   192.168.10.1        192.168.10.2        ICMP
24 9.010248355   192.168.10.2        192.168.10.1        ICMP
```

Figure 5.1: ARP and ICMP messages

5.1.3 Scenario 3

In this experiment, host h4 was made a web server using command "python -m SimpleHTTPServer 80 &". All hosts in the network are granted permission to access the web server except for host h6. Layer 3 IP address and layer 4 protocols were used with a higher priority number in the firewall rules.

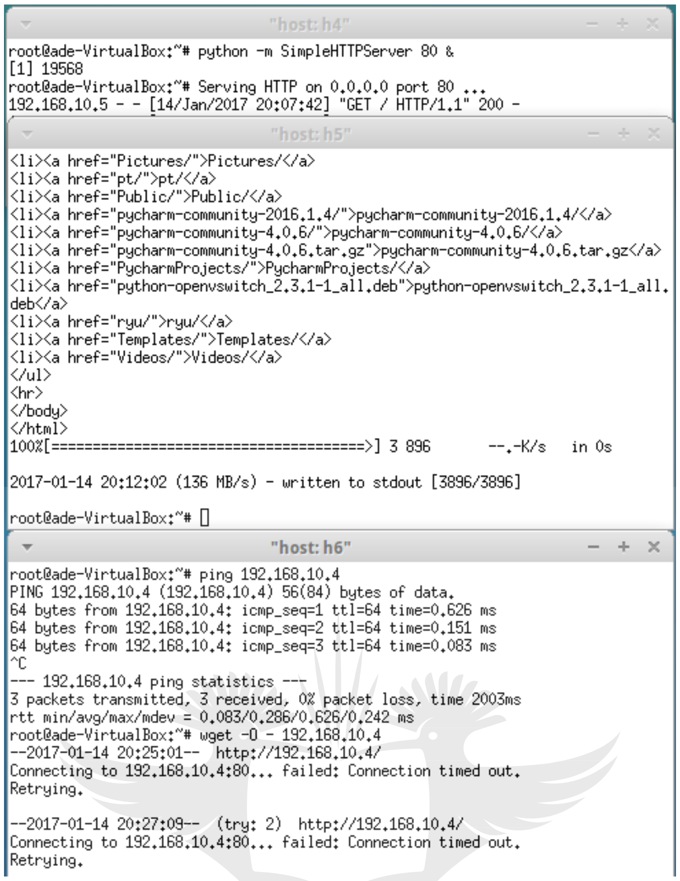
The first command below shows that access was granted to all host on the network after which the second command is then issued to deny host h6 from accessing the webserver. This is done by specifying the source IP address, destination IP address, port number, network protocol and associated action to be taken.

```
a) curl -X POST -d '{"nw_src": "192.168.10.0/24", "nw_dst": "192.168.10.0/24",
   "nw_proto": "ICMP", "actions": "ALLOW", "priority": "205"}'
   http://localhost:8080/firewall/rules/0000000000000001
b) curl -X POST -d '{"nw_src": "192.168.10.6/32", "nw_dst": "192.168.10.4/32",
   "nw_proto": "TCP", "tp_dst": "80", "actions": "DENY", "priority": "206"}'
   http://localhost:8080/firewall/rules/0000000000000001
```

Host h5 and h6 were picked to access the web server. Host h5 sucessfully connect and retrieved information from the web server residing on host h4. Host h6, on the other hand, could not secure a connection to the web server. This can be seen in figure 5.2 as the host is only displaying connecting for a long time and retrying again after an unsucessful connection. It can also be seen that firewall is functioning very well in SDN, this is confirmed on the controller terminal. The controller terminal shows that the host and packets are blocked.

The second command grant pinging access is to host h5 and host h6. This enable both hosts to be able to ping the web server. Pinging is not possible between the two hosts to the web server before, this is because there is no rule to allow the pinging. Pinging between host h5 and host h6 is then blocked while that of host h6 remains, this is done by either deleting the "rule_id" or by adding another rule with a higher priority number. This still allows host h5 to access the web server but not being able to ping the web server while host h6 can still ping but does not have access to any information on the web server.

From all the experiments in the three scenarios, it is seen that firewall in SDN can function more like firewall used in traditional network. It can be used to secure the network by blocking access to the web server from specific IP address or group of IP address. It can also be used to secure intranet where permission is only granted to specific host in the network. Even though SDN is still in early stage, firewall can still be implemented and can funciton effectively in an SDN environment. The use of firewall application in SDN can save some amount on Capital Expenditure (CapEx) incurred on the purchasing of firewall equipment.



```
root@ade-VirtualBox:~# python -m SimpleHTTPServer 80 &
[1] 19568
root@ade-VirtualBox:~# Serving HTTP on 0.0.0.0 port 80 ...
192.168.10.5 - - [14/Jan/2017 20:07:42] "GET / HTTP/1.1" 200 -
```

```
<li><a href="Pictures/">Pictures/</a>
<li><a href="pt/">pt/</a>
<li><a href="Public/">Public/</a>
<li><a href="pycharm-community-2016.1.4/">pycharm-community-2016.1.4/</a>
<li><a href="pycharm-community-4.0.6/">pycharm-community-4.0.6/</a>
<li><a href="pycharm-community-4.0.6.tar.gz">pycharm-community-4.0.6.tar.gz</a>
<li><a href="PycharmProjects/">PycharmProjects/</a>
<li><a href="python-openvswitch_2.3.1-1_all.deb">python-openvswitch_2.3.1-1_all.
deb</a>
<li><a href="ryu/">ryu/</a>
<li><a href="Templates/">Templates/</a>
<li><a href="Videos/">Videos/</a>
</ul>
<hr>
</body>
</html>
100%[===================================>] 3 896       --.-K/s   in 0s

2017-01-14 20:12:02 (136 MB/s) - written to stdout [3896/3896]

root@ade-VirtualBox:~# []
```

```
root@ade-VirtualBox:~# ping 192.168.10.4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_seq=1 ttl=64 time=0.626 ms
64 bytes from 192.168.10.4: icmp_seq=2 ttl=64 time=0.151 ms
64 bytes from 192.168.10.4: icmp_seq=3 ttl=64 time=0.083 ms
^C
--- 192.168.10.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.083/0.286/0.626/0.242 ms
root@ade-VirtualBox:~# wget -O - 192.168.10.4
--2017-01-14 20:25:01--  http://192.168.10.4/
Connecting to 192.168.10.4:80... failed: Connection timed out.
Retrying.

--2017-01-14 20:27:09--  (try: 2)  http://192.168.10.4/
Connecting to 192.168.10.4:80... failed: Connection timed out.
Retrying.
```

Figure 5.2: Firewall confirmation

All host have access to the web server; figure 5.2 shows that host h5 still have access to the web server while host h6 did not have access but can ping the server. Another rule was then added to block the pinging from host h6 to the web server.

Pinging access from host h6 to the web server was later denied by deleting the rule_id from the firewall rule using the following command:

`curl -X DELETE -d '{"rule_id": "5"}' http://localhost:8080/firewall/rules/0000000000000001`. Deleting the rule_id then gives host h6 pinging access back, but it did not have access to the web server.

5.1.4 LATENCY AND THROUGHPUT

Latency in networking can be seen as the time taken for a packet to travel from a source to a destination and back [73]. It is mostly measured using the ping tool. We measured latency without the introduction of firewall and with the firewall runnig on the network. the result achieved is then plotted into a graph and shown in figure 5.3. From the result, it is seen that firewall introduces minimal overhead compared to a network without firewall. From the analysis the network using wireshark, it is seen that a unicast ARP message is sent from host h2 to host h1 asking for the owner of 192.168.10.1 and host h1 also reply back. This unicast message does not consume any bandwidth and therefore does not create additional overheads on the network but have a lower latency at the sixth sent packet. Even though both latency have their flow entries installed proactively, the firewall still has a little more latency than the network without firewall. The latency without firewall has an average of 0.138ms while the average latency with firewall is 0.155ms. This little difference can be ignored in a network environment. The latency results shows that firewall implementation in SDN is functioning very well without affecting the quality of the network.

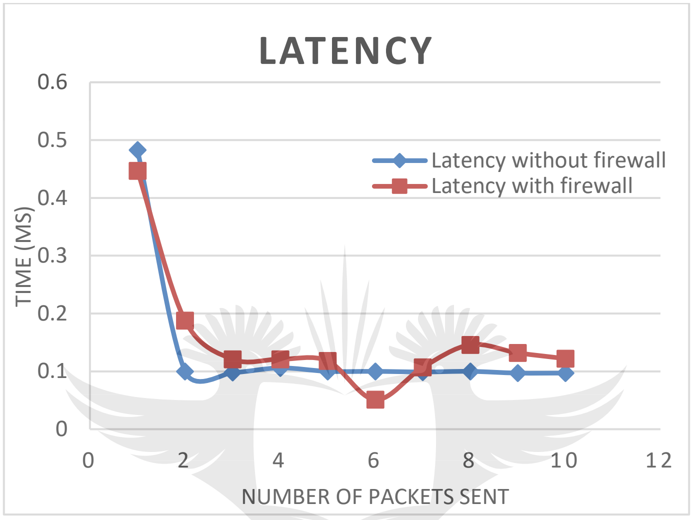


Figure 5.3: Network Latency

Throughput is measurement of data transfer rate from one location to the other over a given amount of time [74]. Network throughput is measurement of maximum achievable bandwidth on a network. This can be measure as maximum achievable data transfer rate from one host to the other over a specified period of time.

In other to test for the throughput, one host is made a server while others a client. We run the test for different number of times. The throuput with firewall and without firewall are plotted on the graph as shown in figure 5.4. The result shows that throuput without firewall is slightyly higher than without firewall. The resulted average of throughput without firewall is 65.1Mbps out of the 100 Mbps while the one with firewall is 59.9Mbps which is still acceptable.

5.1.5 Conclusion

In testing the functionality of firewall in SDN environement, this paper shows that the objectives were achieved. This includes the use of firewall in securing the network by blocking untrusted host or allowing trusted ones using ICMP, TCP, HTTP protocols. We were able to measure the latency and the throughput in the network with firewall in place and without firewall. The firewall should be put in place to block any attack within the network or from outside the network. We were able to perform different experiments using Mininet, Openvswitch and Ryu controller to setup the network toplogy to achieve the aim of this paper. The firewall used in this paper functions by focusing on the packet header of flows in the network rather than the data. The firewall operates from layer 2 to layer 4 on the OSI model for matching of packet header against the firewall rules.

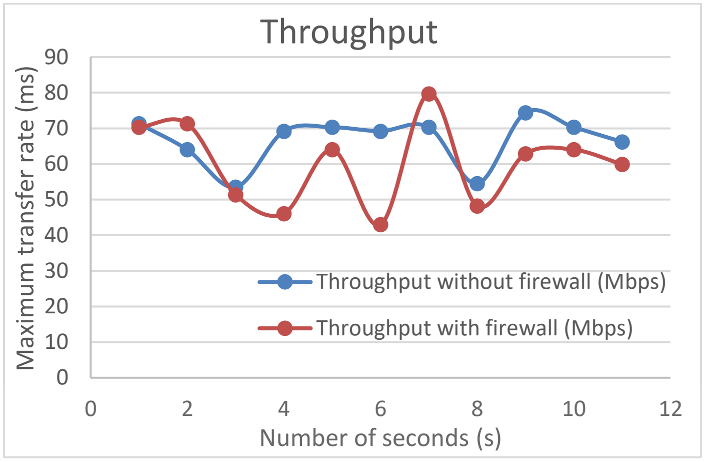


Figure 5.4: Network Throughput

Packet filtering technology firewall was used and rules were added proactively to achieve the goal of this paper. The results acquired shows that sources and destination not listed in the policy are blocked while those listed and matched with the packet header are granted permission based on the associated actions attached. Some of our key findings include the ability to use SDN firewall to secure the network by granting permission or deny access to network services. Considering latency and throughput, we conclude that SDN firewall implementation only has minimal overheads compared to SDN network without firewall. The use of firewall in an SDN environment will reduce the capital expenditure incurred in purchasing firewall hardware in organisations. The limitation of this experiment is that it is only tested using one controller, the use of multiple controllers may increase overheads on the network and there is also security concerns regarding the connection of multiple controllers to switches on the network. This security concern can be recommended for future work to check security issues that may affect secured connection between Ryu controllers and the switches. Another future work is to test the functions of the firewall using application layer 7 on proxy server by using proxy technology.

5.2 QoS in SDN

Quality of Service (QoS) in networking is a way of managing the network resources effectively. QoS has been widely used in traditional network and can also be implemented in Software Defined Network (SDN). QoS in SDN can be used to control the network bandwidth, latency and throughput. QoS works by giving higher priority to some packets over others. Taking real-time video streaming as an example, this is a delay sensitive data and its packets are usually treated with higher priority.

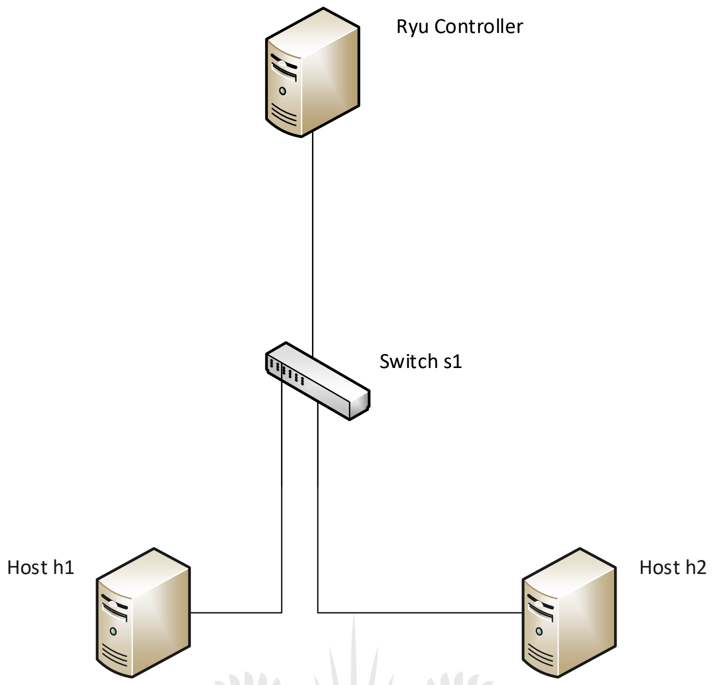


Figure 5.5: QoS topology

5.2.1 QoS Implementation in SDN

As in traditional network, QoS can also be implemented in SDN. SDN makes use of OpenFlow protocol and OpenvSwitch (OVS) for the implementation. OVS support Linux traffic control (tc) which it uses for rate limiting. It also uses Linux queueing disciplines (qdisc) for bandwidth management and Hierarchical Token Bucket (HTB). HTB helps in controlling outbound bandwidth on a given link. When QoS is implemented, HTB ensures each queue in the QoS settings has a number of resources allocated to them [75].

The two major ways of using QoS with OVS in SDN is by Policing and Shaping. OVS support traffic shaping for traffic that egress from a switch. The traffic can be in form of DSCP values allocated to each queue on the QoS. OVS also supports policing for traffic that ingresses into the switch. In summary, SDN makes use of OVS to implement QoS by policing which is based on per-interface ingress rate-limiting using qdisc or by shaping which is based on queues and priority on the network link using HTB [76].

5.2.2 QoS Toolset

The QoS toolset consists of classification tools, marking tools, policing tools and shaping tools. A brief explanation of each toolset category is given below [77]:

1. Classification is the terminology used for flow analysis to determine the traffic class that the flow belongs to and makes decision based on the result of the analysis. The decision to be made is called marking.
2. Marking is done after flows have been analysed, the packets are then marked at the ingress edge router in the network.
3. Policing is the action of dropping packet whenever the allocated network resources have been exceeded.
4. Shaping involves slowing down traffic on the network to maximise the use of the allocated network resources. Unlike policing, shaping will slow down the packet transmission or reception instead of dropping the packet. The slowing down of traffic is done mostly by queueing up packets and then forward them as at when due.

5.2.3 Experimental Setup and Methodology

To test the functionality of QoS in SDN, we set up a testbed using Mininet [58] installed on Virtual Machine, Ryu controller [43], OpenvSwitch [50] and two Virtual Machine installed on VirtualBox. The QoS performance is tested using Policing and Shaping techniques. Four scenarios are tested to explore different functions of Qos and its performance in SDN. The QoS application used is based on Ryu controller while the Mininet source code was modified to accommodate performance monitoring with sflow. This modification makes it easier to monitor the flow of traffic using sflow [78].

The first scenario makes use of Mininet to create a testbed with two hosts [57], [58]. QoS is implemented using traffic policing on the switch connected to the hosts. The second scenario also makes use of Mininet and the QoS is implemented on the network with per-flow traffic shaping. This limits the rate of traffic flow on the interfaces configured on the switch in respect to each flow on the port. Mininet is also used for the third scenario while the QoS is implemented using traffic shaping with queues created on the QoS list. The last scenario uses two VMs to implement traffic shaping using video streaming. The performance of the QoS is checked with the use of queues in the traffic shaping techniques. For all the scenarios, flow entries were entered into the flow table residing on the switch. These flow entries contains the QoS policies such as source and destination IP address, source and destination post numbers, DSCP values and priority. We shall investigate the following in this paper:

- Packet dropping
- Per-interface bandwidth management
- Traffic shaping and scheduling
- Traffic monitoring with sflow
- Measurement of throughput in the network

The experiments in scenario 1-3 were carried out on a virtual machine (VM) running Ubuntu 16.04 LTS and installed on Oracle Virtual Box with an Intel Core i5 quad core and 16GB of RAM host. Applications including Mininet, Ryu controller, OpenvSwitch, Video Lan Client (VLC), iperf, Wireshark, Postman and sflow were installed on the Ubuntu VM. The fourth scenario includes the use of two VMs running inside VirtualBox and has 3GB RAM each.

Mininet is a Linux application that uses the real Linux Kernel to create a realistic virtual network of hosts, links and switches on a single machine which may be VM or cloud host. Ryu controller is written in Python programming language and provides a well defined API that makes customisation, control and management easy. OpenvSwitch is multilayer virtual switch that has been designed to support standard management interfaces and protocols such as sflow, Netflow and CLI. It can also be used to enable network automation through programmatic extension and it can be used in multiple physical servers, multiple Linux-based virtualization technologies such as Xen, KVM and VirtualBox. Iperf is a command line tool used for measuring the maximum achievable bandwidth on a network.

Wireshark is a network protocol and packet analyser used to examine details of communication on the network [65]. sflow is a short form for sampled flow and it is a network technology standard used for monitoring, to enhance network performance and to visualize the network. Postman is also installed to get a coordinated response from the switch instead of getting the results through curl. sFlow is a multi-vendor monitoring technology embedded within switches and routers used to monitor traffic flows at wire speed on network devices interfaces simultaneous [78]. VLC is a free and open source cross-platform multimedia player and streamer. It has support for different protocols such as RTP, HTTP, RSTP and UDP [79].

The network topology used in all the scenarios consist of Ryu controller [43], [66], OpenvSwitch and 2 hosts. The OpenvSwitch is set to use OpenFlow specification 1.3 [55] using the command: `ovs-vsctl set Bridge s1 protocols = OpenFlow13`. We also set the OpenvSwitch to listen on port 6632 for OVSDB access by using the following command: `ovs-vsctl set-manager ptcp:6632`

5.2.4 Scenario 1

The first experiment comprises two hosts created with Mininet as shown in figure 5.2. QoS functionality is investigated on the network with the use of traffic policing on the ports on the switch. The Ryu QoS app is also started as well. Qos rules are setup to limit the rate of the transmission on the interfaces by setting the ingress policing rate and ingress policing burst using the following command: `ovs-vsctl set interface s1-eth0 ingress_policing_rate=10000`

```
Ovs-vsctl set interface s1-eth0 ingress_policing_burst=1000
```

The interface is set to 10Mbps and it is expected to drop excess traffic. Iperf is then used to check the throughput and to confirm the QoS on the interface. We make host h1 the server and host h2 the client. It is found out that excess traffic above 10Mbps are dropped when traffic passes from host h2 to host h1**.** Even though per-interface traffic is good and can be used on the network to limit amount of resources that users are using on the network. It is not considered effective in a situation where services and applications are needed to be segregated. Per-interface is best used with other QoS techniques to manage resources on the network and give better performance.

5.2.5 Scenario 2

Unlike per-interface policing that use QoS rules on each interface, this experiment uses per-flow QoS Policies on the interface. The advantage of per-flow QoS is the fact that the interface will not only be limited to a specific bandwidth but instead, known applications or services can be assigned a bandwidth using IntServ classification. In this scenario, three queues are created on the QoS and configured on the interface. The flow entry is then installed on the switch by matching the network destination address and network protocol to the specific port number followed with an action of assigning the queue. The format for the command used is as follows:

```
curl -X POST -d '{"nw_src": "sourceIP", "nw_dst": "destinationIP", "nw_proto": "ICMP",
"actions":          "ALLOW",          "priority":          "priority          number"}'
http://localhost:8080/firewall/rules/0000000000000001
```

From the attached diagram in figure 5.6, it is seen that the QoS functions very well, as seen by the two queues created. The queue q0 is the default which is only able to use 500kbps bandwidth while q1 is configured to use 1Mbps.

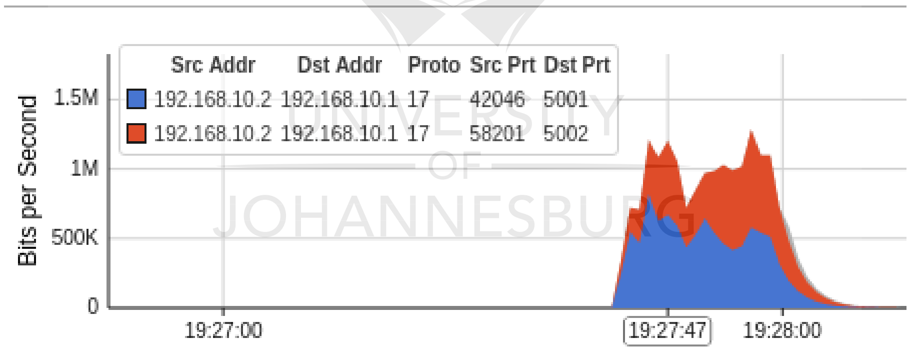
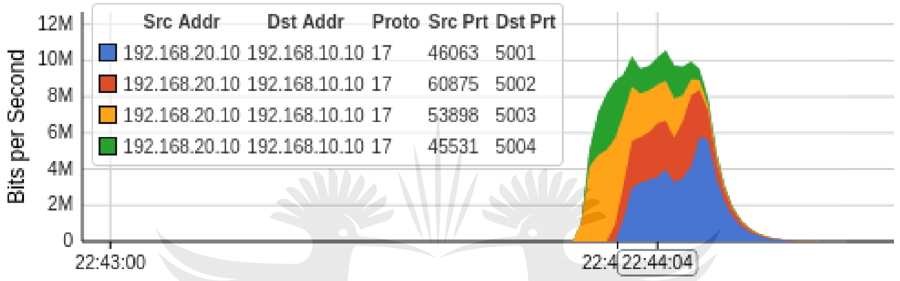


Figure 5.6: Per-interface policing

By analysing with the throughput result it is also seen that excessive packets are queued when the buffer becomes full and are later transmitted when it is not full again. However, shaping is not meant to drop packets, there are situations where some packets are dropped due to too much excess packets that make the buffer to be full without enough bandwidth to pass the first packet in the queue. The attached figure 5.6 used to display the throughput graph is sflow which is used to monitor and manage the network. The sflow also confirms that QoS is effectively maintained and monitored and displays the interface being used and the associated port number.

5.2.6 Scenario 3

In this experiment, the QoS is created using DiffServ. DiffServ uses per-aggregate instead of per-flow in Scenario 2. Per-flow QoS used in scenario 2 is effective but will result in having list of flow entries to be entered onto the switch in a situation where there are many devices on the network and where the bandwidth also increases and this leads to complexity. DiffServ is scalable and can make network QoS configuration easier. It makes use of the edge router to be used as the base router to have the QoS configured, the edge router then feed other connected router with the QoS settings needed. The two routers are configured with IP addresses and gateway after which default routes for the hosts are also configured.



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	192.168.20.10	192.168.10.10	17	46063	5001
🟥	192.168.20.10	192.168.10.10	17	60875	5002
🟧	192.168.20.10	192.168.10.10	17	53898	5003
🟩	192.168.20.10	192.168.10.10	17	45531	5004

Figure 5.7: DiffServ QoS

The installed QoS in this experiment uses DSCP for marking the queues, that is each DSCP value is matched with a queue and entered as a flow entry on router s1. Four queues are entered using DSCP chart as shown in table 1 which is provided by Cisco mostly used by organizations. Table 1 shows the maximum and the minimum bandwidth rate, the DSCP values, QoS ID and Queue ID. The settings for the edge router s2 is also shown in Table 2 and it includes the network destination address, network destination port number, network protocol, DSCP value and queue ID.

Table 2: QoS List with DSCP values

QoS ID	Queue ID	Maximum bandwidth	Minimum bandwidth	DSCP value	Codes
0	0	10Mbps	-		
1	1	10Mbps	1.5Mbps	26	AF43
2	2	10Mbps	4Mbps	40	CS5
3	3	10Mbps	3.5Mbps	46	EF

Table 3: Queues settings with DSCP values

Ntw. Dst Address	Ntw. Dst Port No	Ntw Protocol	DSCP value	Queue ID
192.168.10.10	5001	UDP	0	0
192.168.10.10	5002	UDP	26	1
192.168.10.10	5003	UDP	40	2
192.168.10.10	5004	UDP	46	3

The use of DiffServ classification makes the QoS on the network to be scalable. The router s1 matches the DSCP values with the associated queues while the marking of the DSCP values to the network destination address and destination port number is done on edge s2. Figure 5.7 shows the monitoring of the ports using sflow to show how the packets are transferred, the port number used and the confirmation of the QoS queues with the use of DiffServ. The command to configure and match the DSCP values are written below:

```
a) curl -X POST -d '{"match": {"ip_dscp": "DSCP value"}, "actions":{"queue": "queue
   ID"}}' http://localhost:8080/qos/rules/0000000000000001
b) curl -X POST -d '{"match": {"nw_dst": "IP address", "nw_proto": "UDP", "tp_dst":
   "dst port number"}, "actions":{"mark": "DSCP value"}}'
   http://localhost:8080/qos/rules/0000000000000002
```

The queues created are in line with the standard used by CISCO as seen from the chart and used in this experiment to replicate a real organizational scenario. The bandwidth usage is shown in figure 5.7 which confirms that the allocated ports are not used more than the assigned bandwidth marked with DSCP values. The figure also shows the bandwidth usage of each allocated ports that are assigned to the specific queue and DSCP values. It is also seen that the first queue q0 which is known as best effort has a limited bandwidth usage while other queues are in use. This shows the functionality of the QoS by giving priority to queues with DSCP in line with the assigned bandwidth to manage the network resources effectively.

5.2.7 Scenario 4

In this scenario, two virtual machines (VMs) were used to implement the traffic shaping or scheduling on the network. Video streaming performance is checked by making use of the QoS application in Ryu controller. This experiment use the per-flow method used in scenario 2. On the host computer, we created a bridge named "engr" after which it was turned on and the host eth0 was shutdown. The following command shows the detailed steps taken:

- `ovs-vsctl add-br engr`
- `ifconfig engr up`
- `ifconfig eth0 0`
- `dhclient -v engr`
- `ip tuntap add mode tap vport1`
- `ip tuntap add mode tap vport2`
- `ifconfig vport1 up`
- `ifconfig vport2 up`

- `ovs-vsctl add-port engr vport1`
- `ovs-vsctl add-port engr vport2`
- `ovs-vsctl set Bridge engr protocols=OpenFlow13`

The two virtual ports vport1 and vport2 created are then attached on the VirtualBox network interface to the two VMs before starting them. To get a better performance, we use the traffic policing used in scenario 1 with per-flow traffic shaping. The traffic policing was used to limit the traffic to 10Mbps on vport1 and vport2. This means the two port will only allow 10Mbps traffic on the ports and excess traffic will be dropped. Before starting the Ryu controller, we first set the "engr" switch to be accesible by the Ryu controller using the following command: `ovs-vsctl set-controller engr tcp:192.168.2.1:6653` The OpenFlow is also set to listen on port 6632 in order to access the OVSDB.

The following command is entered to start the QoS on Ryu controller. After Ryu connect to the switch, it displayed the message showing the "dpid" of the switch that it join QoS switch showing: `[QoS][INFO] dpid= 00000021ccd02a48: Join qos switch.` We also configure "ovsdb_addr" to have access to the OVSDB by using this command: `curl -X PUT -d '"tcp:127.0.0.1:6632'" http://localhost:8080/v1.0/conf/switches/00000021ccd02a48/ovsdb_addr`

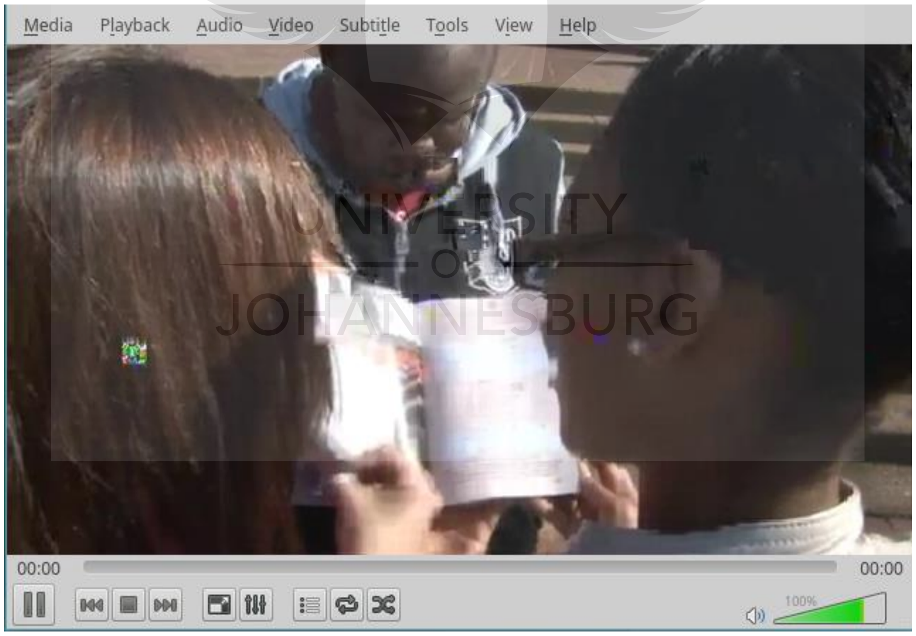


Figure 5.8a: Video streaming without QoS

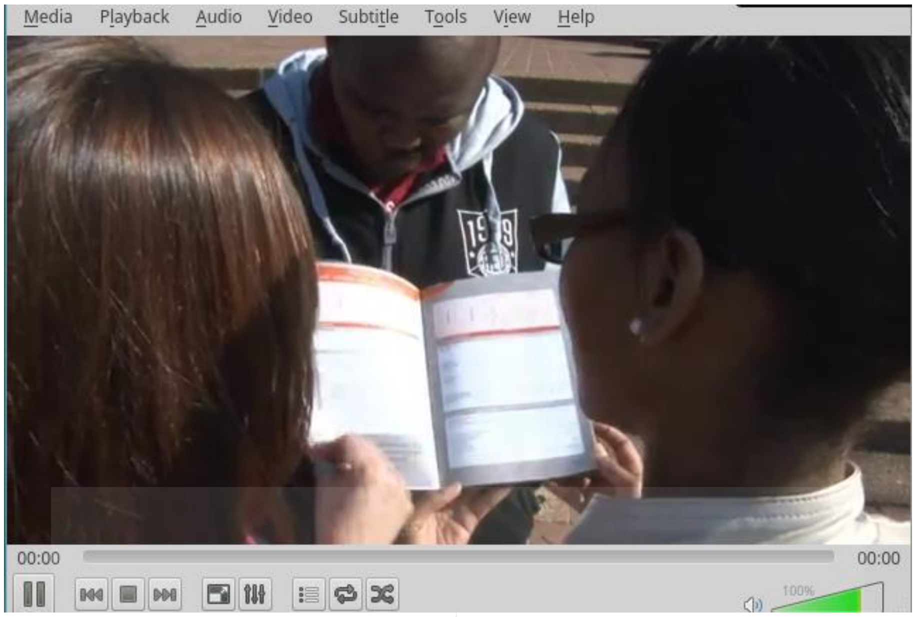


Figure 5.8b: Video streaming with QoS

We also create two queues to be used using the following bandwidth allocation. The following is used to set the queues on the switch: `curl -X POST -d '{"port_name": "vport2", "type": "linux-htb", "max_rate": "10000000", "queues": [{"max_rate": "800000"}, {"min_rate": "9000000"}]}' http://localhost:8080/qos/queue/00000021ccd02a48`

The flow entries to be used which include the match and subsequent action is `curl -X POST -d '{"match": {"nw_dst": 192.168.2.8", "nw_proto": "UDP", "tp_dst": "5004"}, "actions":{"queue": "1"}}' http://localhost:8080/qos/rules/00000021ccd02a48`

Two tests are taken and snapshots are also taken. The first test is taken without the presence of Qos configured on the switch. This is done by setting up a VLC server on host 1 and VLC on host 2. This is done to stream video from one host to the other to replicate a real life scenario where video streaming services such as skype video has more priority than ordinary internet browsing or data downloading. Iperf server is also set up on host 1 while iperf client is set up on host 2 (almost simultaneoulsy with around 2s difference between the start times). From the attached image in figure 5.8a, it is seen that the video scrambled a lot and not really viewable. This is due to the sharing of bandwidth between the two services running while they both shared the bandwidth, which led to the effect on the quality of the video.

The second test involves the use of QoS and queues are created and installed on the switch. After the installation of queues and flow entries on the switch, we also start VLC server and iperf server on host 1 and their respective client services on host 2 as we did in the first test. The two queues were created, the first default queue q0 has 800kbps maximum out of the available 10Mbps while the second has 9Mbps minimum bandwidth. It is seen from attached figure 5.8b that the video did not scramble like it did in the first test without QoS. The use of queues make it easier to manage trafffic. This experiments also shows that QoS function well in SDN with the use of OpenFlow protocol in the netwok.

5.2.8 Conclusion

In testing the functionality of QoS in SDN, this paper shows that the goals were achieved. The testbed QoS functionality include per-interface traffic policing, per-flow QoS, DiffServ QoS and the traffic monitoring with sflow. The first three scenarios testebed was created with Mininet while Ryu controller, OpenvSwitch, iperf and sflow were used to achieve the objectives of this paper. The last scenario, however, makes use of two virtual machines to confirm the functionality of QoS in SDN with the use of VLC for video streaming. The QoS is set in the network by creating queues and installed flow entries on the switch after which specific actions are matched to the queues to function. The destination IP address and the destination port number was used in matching the created queues with associated actions.

We, therefore, recommend the use of per-interface QoS with per-flow QoS or with DiffServ QoS to get effective result and to manage the network resources. We also found out that QoS function efficiently in SDN like as it is in traditional network. We were able to to use per-flow QoS which can be used in smaller network with little number of QoS while we also use DiffServ technique which is used at the edge router of bigger network and uses DSCP values for traffic classification which also makes QoS planning easier. VLC video streaming was also used to confirm the performance in real scenario where services such as skype or live youtube video require higher priority. The limitation of this experiment is that only virtual network environment is used, though this does not really have any significant diffence on the QoS performance. Future work on this paper include the use of sflow to build a QoS and traffic monitoring and management interface.

6. SDN MONITORING AND VISUALISATION

6.1 Introduction

Network monitoring being part of network management involves the use of hardware or software to monitor the network traffic. Network monitoring uses measurement techniques that may be passive or active. Passive method measures the network traffic by observing the network without invoking additional probing packet into the traffic. Lack of additional probing packets is major advantage because there are no additional overheads in the network. Another advantage is that network performance is not affected while the major disadvantage is that if not configured properly it may not be suitable for a large network. Active method on the other hand work be invoking packets in the network to monitor network traffic. Addition of probing packets in the network create some additional overheads on the network which may affect network performance if not configured properly.

Most of the time polling technique is used to request the traffic details and status of the networking devices on the network. Traditional network usually uses Simple Network Management Protocol (SNMP), NetFlow and sFlow to monitor the network. SNMP works by polling information request per-interface on all ports to get full data from the switching device. NetFlow and sFlow works by sampling 1in n packets to represent information requested. This reduces overhead in the network as well.

With an increase in the number of connected devices on the network, managing the network traffic has been an issue for network administrators. Network traffic monitoring and visualization is of great importance for Internet Service Providers (ISPs) and network administrators. Different monitoring techniques that have been used earlier include monitoring the packets and ports on networking devices. Monitoring enables the network to have better performance and gives the network administrator ability to manage, plan and use the network resources efficiently.

Managing and configuring network with multiple devices from different vendors has been cumbersome for network administrators. There is need to learn specific vendor configuration and monitoring packages. To eliminate these vendor dependencies, a new network architecture was developed called Software Defined Network (SDN). SDN separates the control plane from the data plane in the network which are combined together in a traditional network device In other words, SDN moves the network intelligence control plane to a logically centralised location and it is responsible for configuration used in packet forwarding in the data plane that resides in the forwarding hardware [80], [81].

Originally, the control plane is responsible for decision making regarding packet forwarding used by the data plane which consists of switches and routers. The SDN controller represents the control plane which connects to the data plane and placed in physically centralised or logically centralised location. Unlike the traditional network that rely on protocols such as OSPF, EIGRP, RIP etc, SDN makes use of OpenFlow protocol which is set as a standard and used for secured communication between the controllers and the switches in the network. OpenFlow uses secured TLS or SSL connection for communication among devices

In this chapter, we develop SDN-MON which operates on the control and management plane and it is used for monitoring and visualizing network traffic. We design SDN-MON on top of Ryu topology viewer and SDNHub viewer [82]. SDN-MON which is webGUI which is an improvement of SDNHub viewer. This improvement includes redesign of the interface and integration of sFlow into the webGUI. The web GUI is built with the modification of earlier one built by SDNHub [82].

6.2 Monitoring, Visualizing and Measurement.

Traffic monitoring and visualizing is a key management activity that should be taken seriously to provide better services to end users and to maintain a good network resource management. Network traffic changes dynamically and continual monitoring is required by network administrators to keep the network in good condition. The parameters to monitor and measure include packets and flows passing through the network devices [83]. In SDN, network traffic can be of two categories namely: control traffic and data traffic. Control traffic represent the traffic flow between SDN controller and switches on the network while data traffic is the traffic or data communication between switches on the network [83]. Some of the parameter used include throughput, latency, packet loss and bandwidth utilisation. All these parameters are used and visualized on the CLI or GUI using RESTAPI with web interface.

SDN network monitoring uses CLI polling and sampling like traditional network but does not utilise SNMP like traditional network. The mechanism behind traffic monitoring is of two parts namely: real-time monitoring and statistical data monitoring [84]. Both can be visualized and measured. Real-time monitoring works by using OpenFlow SDN controller and OpenFlow protocol to monitor the network traffic. This is done by making use of API's to configure, monitor, and manage traffic entering or leaving the network devices. Statistical data monitoring makes use of sFlow or NetFlow to sample data at a set interval on devices on the network and monitor them to analyse the network for performance or error detection.

6.3 Design and Implementation

This section contains details on the design and the implementation. The design involves the use of Ryu [43] and OpenFlow REST API, JSON and JavaScript which are used with sFlow. The design comes in two-fold by using the inbuilt OpenFlow monitoring and sFlow. The inbuilt OpenFlow monitoring makes use of RYU REST API to monitor the flow table, the flows and the ports. sFlow on the other hand monitors and measures the traffic in the network by sampling the switches for a specified time to get the status of the traffic on all the switches and all connected hosts.

The two features are built and can be accessed on the web GUI. Both features are embedded in a web GUI called SDN-MON and built with HTML, CSS and JavaScript. REST API and JSON are used to update information needed to be displayed on the web interface. SDN-MON which is webGUI is an improvement of SDNHub viewer [82] include redesign of the interface and integration of sFlow with Mininet integration in the web GUI to satisfy the aim of this chapter.

6.4 Real-Time OpenFlow Monitoring

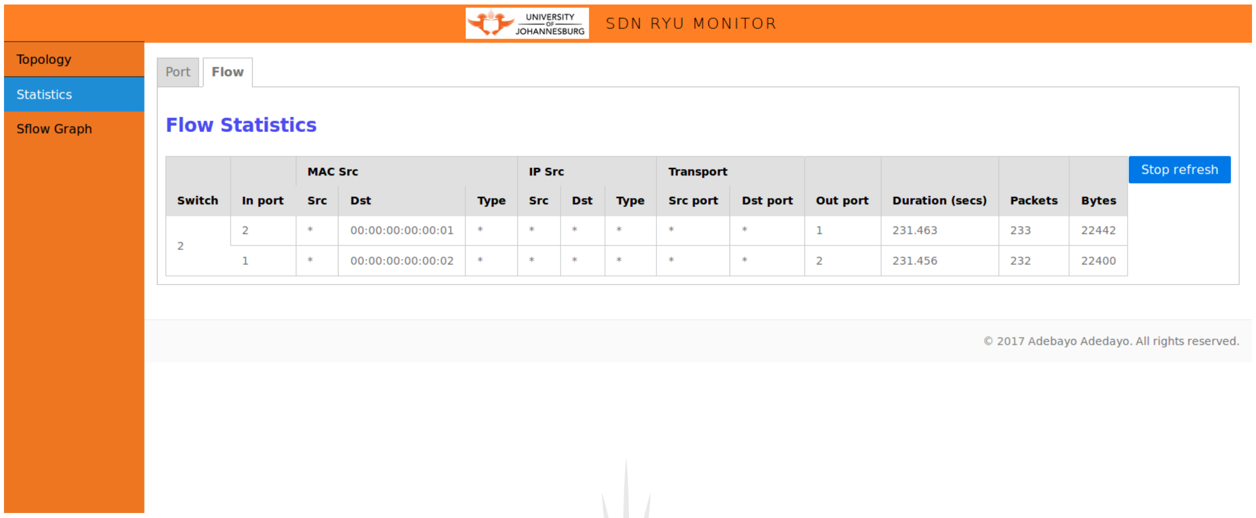
Monitoring SDN network requires the use of SDN controllers gathers data from the switches in the network [85]. Unlike traditional networks where the forwarding plane and the control intelligence reside in the switch, SDN-enabled OpenFlow switch acts more like a dumb switch without connection to the SDN controller. OpenFlow protocol allows communication between OpenFlow-enabled switches and the SDN controller on the network. In this chapter, Ryu controller monitor application is used to examine the OpenFlow network monitoring [43]. The application makes use of OpenFlow specification 1.3 features to monitor the flow statistics and Port statistics information on the switches. Some of the features include "EventOFPStateChange", "OFPFowStatsRequest", "OFPPortStatsRequest", "OFPFlowStatReply" and "OFPPortStatsReply".

EventOFPStateChange is used for confirmation that a switch is being monitored which it does by monitoring connection or disconnection between the controller and the switch. OFPFlowStatsRequest query the switch to get statistical information related to flow entry. This include flow table ID, output port, packets etc. OFPPortStatsRequest sends request to the switch to get statistical information related to ports on the switch. All ports can be queried or specific port can also be queried to get the required result. The OFPFlowStatsReply respond to the request from OFPFlowStatsRequest by providing the statisticall information of each flow entry as requested. OFPPortStatsReply on the other hand also responds to the request from OFPPortStatsRequest by providing statistical information of each ports on the switch as requested. Both results of FlowStats and PortStats can be displayed in JSON format which is useful with RESTAPI for visualizing on a webGUI [66].

Some of the parameters measured and displayed by Flowstats include byte count, duration, packet count, tableID and priority while Portstats measured parameters include port number, receive packet count, receive byte count, receive error count, send packet count, sent byte count, and send error count.

To demonstrate the features of the SDN-MON, we created a tree topology with four switches with limitation to 10Mbps bandwidth. A simple switch Ryu application is also started with REST application and ofctl_rest which is useful for JSON and browser visualisation. Figure 6.1 shows SDN-MON with topology, statistics and sFlow graph tabs. The statistics tab contains the real-time OpenFlow monitoring with the use of inbuilt OpenFlow features. The tab has Port Stats and Flow

Stats section. The flow section shows the communication between host h1 and host h2 with the duration packets and bytes count. This present the flows in the flow table. Figure 6.2 on the other hand shows communication among 4 hosts on the network.



Flow Statistics

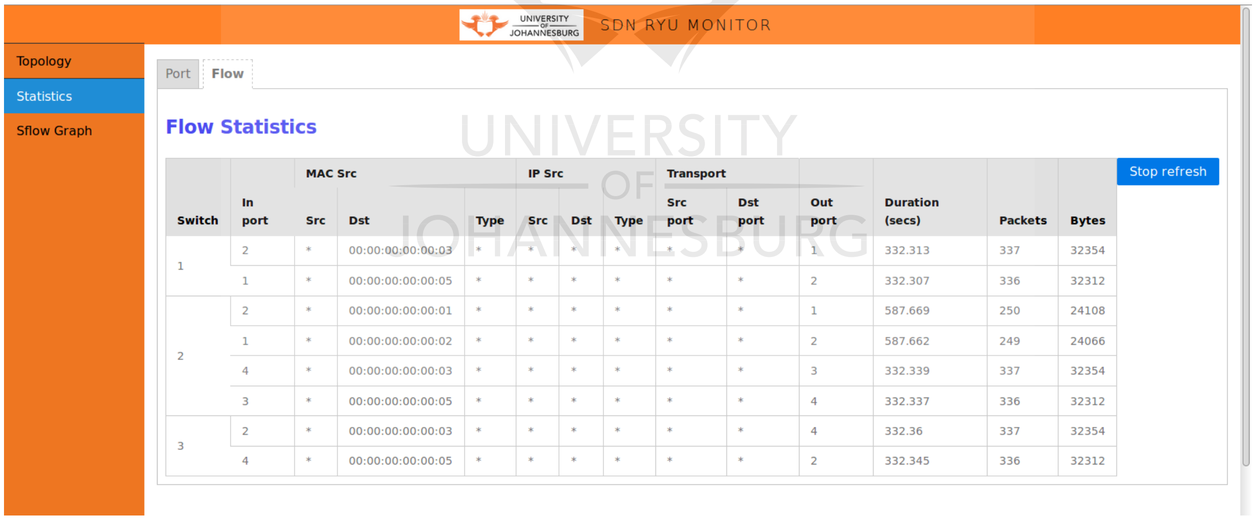
Stop refresh

		MAC Src			IP Src			Transport					
Switch	In port	Src	Dst	Type	Src	Dst	Type	Src port	Dst port	Out port	Duration (secs)	Packets	Bytes
2	2	*	00:00:00:00:00:01	*	*	*	*	*	*	1	231.463	233	22442
	1	*	00:00:00:00:00:02	*	*	*	*	*	*	2	231.456	232	22400

© 2017 Adebayo Adedayo. All rights reserved.

Figure 6.1: Flow Stats of 2 hosts

It shows communication between host h1 and host h2, and between host h3 and h5. The duration in seconds, packets and bytes count are also displayed.



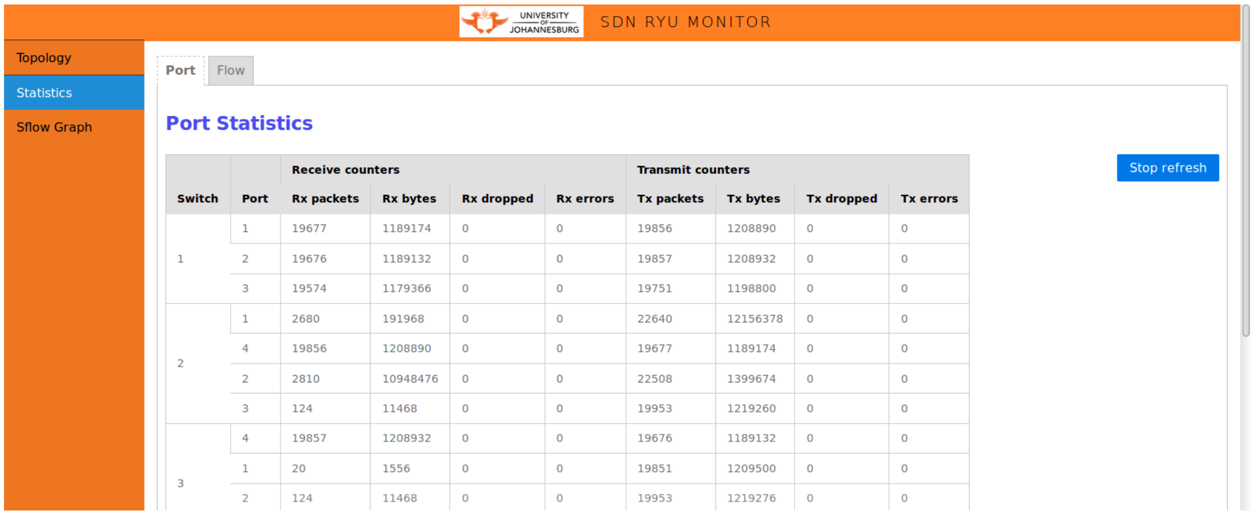
Flow Statistics

Stop refresh

		MAC Src			IP Src			Transport					
Switch	In port	Src	Dst	Type	Src	Dst	Type	Src port	Dst port	Out port	Duration (secs)	Packets	Bytes
1	2	*	00:00:00:00:00:03	*	*	*	*	*	*	1	332.313	337	32354
	1	*	00:00:00:00:00:05	*	*	*	*	*	*	2	332.307	336	32312
2	2	*	00:00:00:00:00:01	*	*	*	*	*	*	1	587.669	250	24108
	1	*	00:00:00:00:00:02	*	*	*	*	*	*	2	587.662	249	24066
	4	*	00:00:00:00:00:03	*	*	*	*	*	*	3	332.339	337	32354
	3	*	00:00:00:00:00:05	*	*	*	*	*	*	4	332.337	336	32312
3	2	*	00:00:00:00:00:03	*	*	*	*	*	*	4	332.36	337	32354
	4	*	00:00:00:00:00:05	*	*	*	*	*	*	2	332.345	336	32312

Figure 6.2: Flow Stats of 4 hosts

Figure 6.3 shows the port stats of the network. It consists of transmitted packets, transmitted bytes, received packets and received bytes. It shows the switches that are being used and the ports on each one of them that are used with updated data in every 5 seconds.



Port Statistics

		Receive counters				Transmit counters			
Switch	Port	Rx packets	Rx bytes	Rx dropped	Rx errors	Tx packets	Tx bytes	Tx dropped	Tx errors
1	1	19677	1189174	0	0	19856	1208890	0	0
	2	19676	1189132	0	0	19857	1208932	0	0
	3	19574	1179366	0	0	19751	1198800	0	0
2	1	2680	191968	0	0	22640	12156378	0	0
	4	19856	1208890	0	0	19677	1189174	0	0
	2	2810	10948476	0	0	22508	1399674	0	0
	3	124	11468	0	0	19953	1219260	0	0
3	4	19857	1208932	0	0	19676	1189132	0	0
	1	20	1556	0	0	19851	1209500	0	0
	2	124	11468	0	0	19953	1219276	0	0

Figure 6.3: Port Stats of 4 hosts

6.5 Statistical Data Monitoring

Statistical data monitoring of the network involves the use of two major players namely; NetFlow and sFlow.

6.5.1 NetFlow

NetFlow was developed by Cisco to collect statistical information on IP traffic and monitor the network traffic as well. NetFlow uses NetFlow collector that collects the data and processes the data to be analysed by the NetFlow analyser. The duo is useful to visualize the source and destination of traffic as well as bandwidth being used. NetFlow uses these 5-tuple (source IP address, destination IP address, source TCP/IP port, destination TCP/IP port, and IP protocol number) to track flow statistics for each flow on the switch or router [86].

NetFlow works by matching incoming packet to the flow cache. If a match occurs, the flow cache entry is updated by incrementing the packet and byte count. NetFlow is a Cisco proprietary protocol and only available on Cisco devices. Though it is also found on virtual switches such as VMware ESX and OpenvSwitch. NetFlow can work in sampling mode by sampling 1 in N packets that travel in or across a switch instead of every switch. The sampled flows are still sent to records on the flow cache and exhibits some timeout which makes it not suitable for low-latency monitoring. Another demerit of NetFlow is that it does not support LAN and VLAN monitoring and requires high-end routers and switches [87].

6.5.2 sFlow

sFlow is a short form for sampled Flow. sFlow is a network technology standard used for network traffic monitoring. sFlow provides a visualized view of the entire network and it is widely used being a vendor independent monitoring technology. sFlow is embedded in multiple network devices with growing support and it is cheaper than NetFlow [85]. Many proprietary network devices already have sFlow agents present on them. In SDN, most OpenFlow enabled switches has sFlow embedded in them, this includes OpenvSwitch. SFlow uses a random sampling technique to sample the data flow in the network [88].

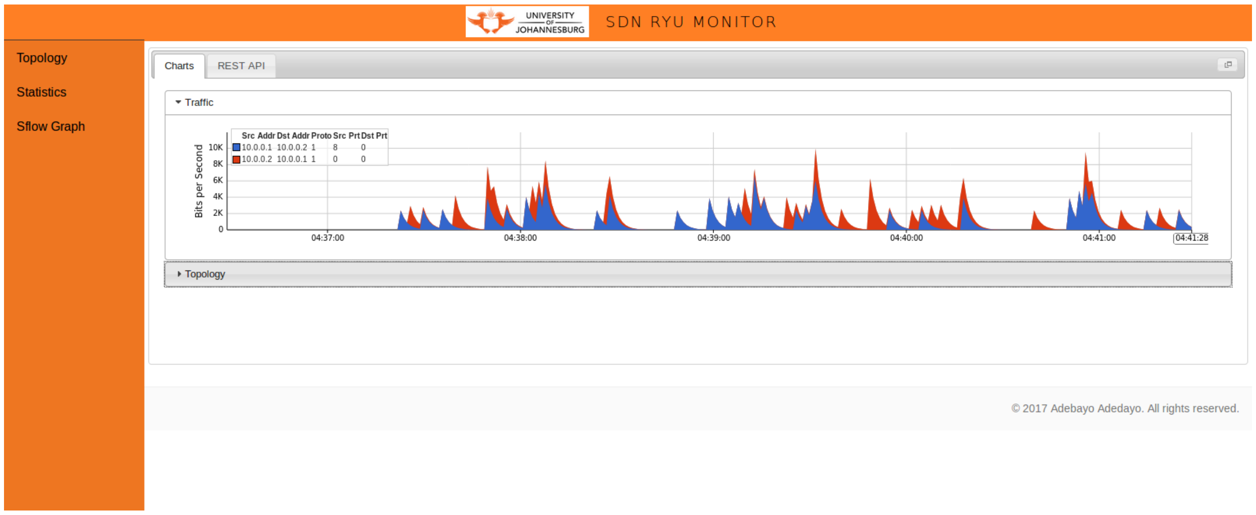
It uses sFlow agent, a software incorporated on the switch to sample packets at a certain configurable rate that can be set by the user [85]. It normally samples 1 in N packets. Based on the size of the network, the sampling varies depending on arrival of the packets [83]. The sampled data are then packaged info sFlow Datagram and sent over the network to an sFlow collector. This has a lot of advantage over NetFlow because it utilizes little memory and CPU usage required by the sFlow agent is minimal. The Datagram sent to the central sFlow collector is analysed to produce a detailed, real-time network traffic information. The collected and analysed information can be viewed using sFlow-RT, sFlowTrend and Ganglia and are accessed using REST API to be equipped on web Graphical User Interface (GUI) [78], [87].

Some of the advantages of sFlow include the following:

- Accuracy: sFlow operates at wire speed and can accommodate and work well under heavy loads.
- Scalable: sFlow scalability makes it easier to monitor network at different network speed 100Mbps, 1Gbps, 10Gbps, 100Gbps and higher. It can also monitor thousands of devices using a single sFlow collector.
- Low Cost: sFlow uses a simple sampling algorithm that is easier to implement and has a negligible processing power on the switching device.
- Timely: sFlow captured packets contain full view of traffic information and are up to date. The Packets can be used to manage QoS and firewall.

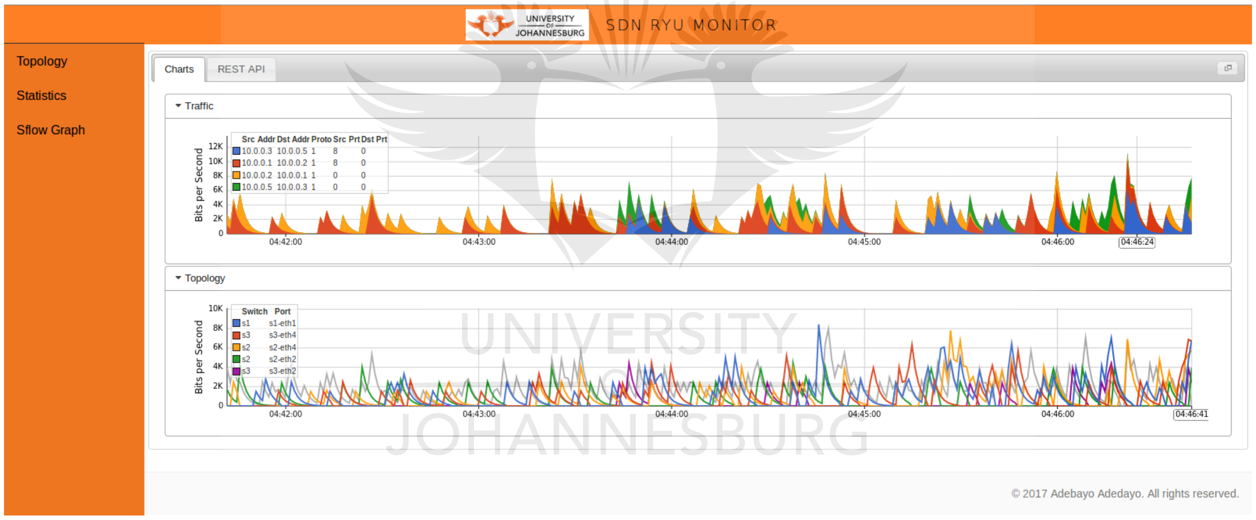
6.5.3 Flow monitoring with sFlow.

Just like traditional network, OpenFlow switches in SDN are configured to use sFlow and communicate with the SDN controller. SDN controller makes use of OpenFlow protocol to communicate with switches on the network, and uses the same to gather information to be used in building the topology of the network. sFlow can be used together with the controller and OpenFlow to provide and create an environment with full network view and flow monitoring. Even though SDN controllers make the forwarding decisions, sFlow integration on all the switches is used for monitoring the traffic flow regarding the forwarding decision made by the controller.



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.1	10.0.0.2	1	8	0
🟥	10.0.0.2	10.0.0.1	1	0	0

Figure 6.4: sFlow Stats of 2 hosts

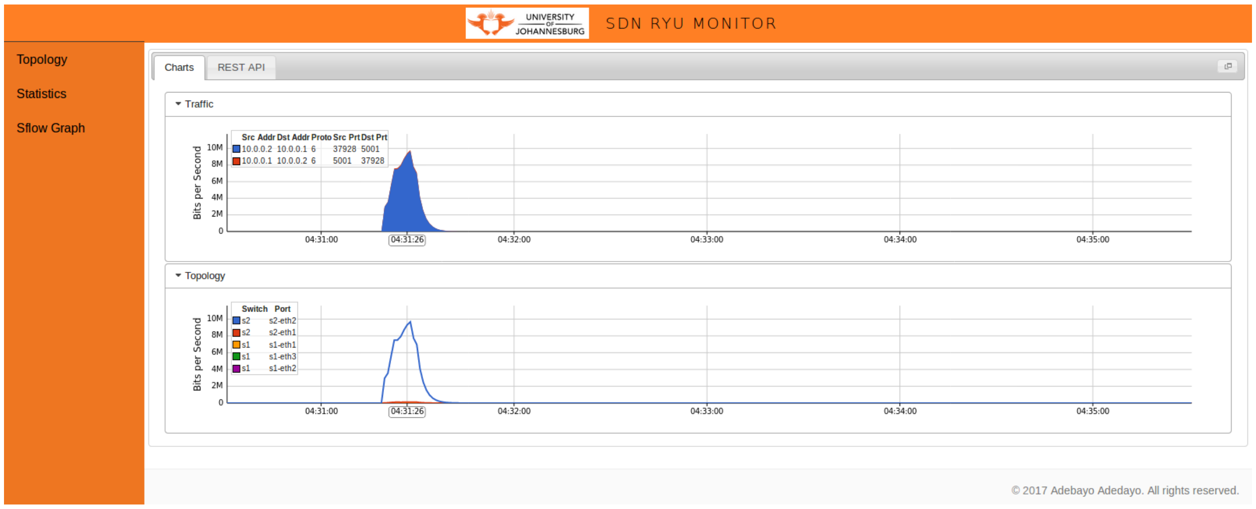


	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.3	10.0.0.5	1	8	0
🟥	10.0.0.1	10.0.0.2	1	8	0
🟧	10.0.0.2	10.0.0.1	1	0	0
🟩	10.0.0.5	10.0.0.3	1	0	0

	Switch	Port
🟦	s1	s1-eth1
🟥	s3	s3-eth4
🟧	s2	s2-eth4
🟩	s2	s2-eth2
🟪	s3	s3-eth2

Figure 6.5: sFlow Stats of 4 hosts

Using the same testbed that was used for real-time monitoring, the diagram in figure 6.4 and 6.5 shows the use of integrated sFlow in monitoring the network. Figure 6.4 shows the commuication between host h1 and host h2. Additional communication between host h3 and host h5 is shown in figure 6.5. The sFlow colour-coded transmission between two host in one direction at a time, i.e transmitted and received traffic are represented with different colour and are noted for easier visualisation. Figure 6.6 shows the throughput information between host h1 and host h2. The limit being 10Mbps as earlier configured in the testbed.



	Src Addr	Dst Addr	Proto	Src Prt	Dst Prt
🟦	10.0.0.2	10.0.0.1	6	37928	5001
🟥	10.0.0.1	10.0.0.2	6	5001	37928

	Switch	Port
🟦	s2	s2-eth2
🟥	s2	s2-eth1
🟧	s1	s1-eth1
🟩	s1	s1-eth3
🟪	s1	s1-eth2

Figure 6.6: sFlow showing Throughput

This chapter shows that SDN can be monitored and visualised by using the inbuilt OpenFlow features and integrated sFlow into a web GUI. The web GUI can be used to monitor the traffic on the network. It can also be modified with additional features as required by the user.

7. CONCLUSION

This conclusion presents the testing of functionality and effectiveness of Software Defined Network in a LAN environment. This thesis starts with the brief introduction to networking and literature review on virtualization, SDN and OpenFlow protocol which is being used by OpenFlow enabled switches in an SDN environment. Basically, SDN is the decoupling or separation of control plane from the data plane in the network device. The separated control plane functions effectively when it is being controlled from a centralised logical point.

OpenFlow uses a secured channel with TLS or SSL to communicate between an SDN controller and the switches on the network. OpenFlow has gained popularity within the network device vendors with newer switches being OpenFlow enabled. More features have been added to OpenFlow since its inception, though specification 1.0 and 1.3 are more popular and used on software and hardware switches. OpenFlow is integrated into the network stack and operates as an encapsulation inside the application layer of the OSI model. Unlike the traditional network, forwarding of incoming packet is based upon information on the flow table rather than forwarding table used by the traditional network. OpenFlow protocol uses the flow table which also contain flow entries for forwarding. Packets are forwarded whenever there is a match in the flow table. In a case where there is no match, the packets are forwarded to the SDN controller for decision making.

To achieve the goal of this research, some experiments have been carried out for testing and analysis. Though the test was carried out in virtual environment with virtual machines and virtual network switches, the expected results have been achieved. Communication between hosts have been tested on a LAN environment and on a VLAN environment. The hosts communicated as expected by transmitting and receiving packets. The research shows communication between hosts in a LAN environment with analysis done on the packets being transmitted and received between hosts on the LAN. The latency and throughput results achieved shows that SDN function well like traditional network with less OpEx and CapEx.

VLAN 802.1q was also setup on the network and used to represent different departments in the faculty. A tree topology testbed was setup to show the functionality of VLAN which is to break broadcast domain thereby reducing broadcast storm and having a lesser load on the network bandwidth. Hosts on one VLAN was not able to communicate with hosts on a different VLAN even when they get their connection from the same switch. Also, hosts on same VLAN can communicate, though default and static route need to be configured on the flow table before this can be a success.

Securing the network require a firewall. The firewall testing uses different scenarios to test the effectiveness of firewall in SDN. The tests make use of ICMP, TCP and HTTP protocols which operate on layer 2 to layer 4 of the OSI model. Just like the traditional network, Access Control Lists (ACL) were added to the flow table to create a policy. The results in the experiments show that any host not listed in the policy are blocked while those listed are granted permission to communicate. Latency and throughput was also measured with firewall in place and was compared with the ones without firewall. The results show that little overhead has been added to the network due to the inclusion of firewall but this little overhead can be ignored because it is insignificant though there may be a significant overhead in a much larger network environment.

Management of network resources is an important function that cannot be neglected by any system administrator. QoS plays a major role in the management of network resources and should be put in place in any network environment. Like traditional network, SDN also makes use of QoS to control network bandwidth, latency and throughput. In the experiments in chapter 6, QoS uses priority given to host attached to each interface on the switch or applications on the network to control the network bandwidth. The experiments show the use of per-interface, IntServ and DiffServ QoS classification. Per-Interface works by dropping excess packet that is more than the configured acceptable bandwidth limit. It uses the ingress policing technique to drop excess packets that are more than the configured bandwidth on a specific interface.

IntServ QoS was used in the per-flow traffic shaping where queues are created in the QoS rules and are used on each interface of the switch to control the traffic. The results show that configured destination port number get only what they are configured for. This has advantage over per-interface QoS because each interface can have multiple queues for different port numbers which also represent different applications. The major disadvantage is the requirement for configuration of each interface on all the switches on the network. Thus, it can be concluded that per-flow with IntServ is suitable for small to medium scale network environment and cannot be used in a larger network.

DiffServ QoS with per-aggregate flow was also tested to solve the major disadvantage of configuring each interface with required QoS rules for specific application or services. DiffServ being a scalable QoS classification made it possible to configure the edge router with the required QoS rules and policies instead of per-interface on every switch on the network which is used by IntServ. The configured QoS uses DSCP values to mark the queues which are installed on the flow tables. Multiple queues are created with different matching to the DSCP values which are used with priority to manage the network bandwidth usage. This makes it easier to prioritise organisation's VOIP calls and video conference calls over video and large file download. A practical example is shown with the use of DSCP value and higher priority given to a live video used to represent a video conference call. The results show that the video scrambled without QoS in place while the one with QoS plays very well without any distortion. This shows that QoS in SDN is functional and effective to be used in the network and can perform well like traditional QoS.

No network environment is complete without a good monitoring interface in place. In the last chapter, we build a webGUI called SDN-MON used for monitoring and visualization. Traffic monitoring and visualization is an important part of network management and should not be neglected. The webGUI consist of a real-time OpenFlow monitoring and a statistical data monitoring. The real-time OpenFlow monitoring makes use of in-built OpenFlow features that monitors the State change, Port Status and Flow Status on the switch. These three features make it easier to monitor the state of the network switches and the results are displayed on the SDN-MON. REST API is used in the displaying and monitoring and the data are displayed in an easy readable format using JSON. Statistical data monitoring makes use of integrated sFlow for better visualisation. SFlow presents visualisation of the entire network traffic. The random sampling technology used by sFlow brings about effective monitoring without adding overheads to the network.

Finally, this thesis has shown that SDN with OpenFlow is functional and effective like the traditional network. Though it is still in working phase and is steadily being adopted by very few organisation. The technology has proved to be a success and can be worked on for improvement in areas such as security, cloud usage and other areas that need improvement.

REFERENCES

[1] N. Feamster, J. Rexford and E. Zegura, “The Road to SDN - An intellectual history of programmable networks,” *Queue*, vol. 11, no. 12, p. 21, December 2013.

[2] S. Azodolmolky, Software Defined Networking with OpenFlow, Birmingham: Packt Publishing Ltd, 2013.

[3] ONF, “Software-Defined Networking: The new norm for networks,” 13 April 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accessed 2 3 2015].

[4] T. D. Nadeau and K. Gray, SDN: Software Defined Networks, Sebastopol: O’Reilly Media, 2013.

[5] G. Dasmalc, “UnderstandingSDNTechV1.pdf,” December 2014. [Online]. Available: https://www.sdxcentral.com/wp-content/uploads/2014/12/UnderstandingSDNTechV1.pdf. [Accessed 4 April 2016].

[6] X. Foukas, M. K. Marina and K. Kontovasilis, “Software Defined Networking Concepts,” in *Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture*, Chichester, John Wiley & Sons, Ltd, 2015.

[7] B. N. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka and T. Turletti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014.

[8] R. Wang, D. Butnariu and J. Rexford, “OpenFlow-based Server Load Balancing Gone Wild,” in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Boston, 2011.

[9] H. Qian and D. Medhi, “Server Operational Cost Optimization for Cloud Computing Service Providers over a Time Horizon,” in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Boston, 2011.

[10] S. Sarkar, R. Mahindru, R. A. Hosn, N. Vogl and H. V. Ramasamy, “Automated Incident Management for a Platform-as-a-service Cloud,” in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Boston, 2011.

[11] D. Breitgand, G. Kutiel and D. Raz, “Cost-aware Live Migration of Services in the Cloud,” in *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, Haifa, 2010.

[12] S. Yegulalp, “Five SDN Benefits Enterprises Should Consider,” 7 December 2013. [Online]. Available: http://www.networkcomputing.com/networking/five-sdn-benefits-enterprises-should-consider/70381323. [Accessed 4 April 2016].

[13] N. Sharma, “Eight Big Benefits of Software-Defined Networking,” IT Business Edge, 20 January 2015. [Online]. Available: http://www.serverwatch.com/server-tutorials/eight-big-benefits-of-software-defined-networking.html. [Accessed 4 April 2016].

[14] Intel Architecture Processors Networking and Communications, “Open, Simplified Networking Based on SDN and Network Functions Virtualization,” [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/sdn-part-1-secured.pdf. [Accessed 4 April 2016].

[15] B. A. Forouzan, Data Communications and Networking, New York: McGraw-Hill, 2007.

[16] A. S. Tanenbaum, Computer Networks, New Jersey: Pearson Prentice Hall, 2011.

[17] T. Lammle, CCNA: Cisco Certified Network Associate Study Guide, Indianapolis: Wiley Publishing, Inc., 2011.

[18] VMware, “Virtualization,” VMware, [Online]. Available: http://www.vmware.com/virtualization/how-it-works. [Accessed 02 11 2015].

[19] TechTarget, “What is virtualization,” Techtarget, [Online]. Available: http://searchservervirtualization.techtarget.com/definition/virtualization. [Accessed 02 11 2015].

[20] L. I. B. López, Á. L. V. Caraguay, L. J. G. Villalba and D. López, “Trends on virtualisation with software defined networking and network function virtualization,” *The Institution of Engineering and Technology*, vol. IV, no. 5, pp. 255-263, 2015.

[21] S. Azodolmolky, P. Wieder and R. Yahyapour, “SDN-Based Cloud Computing Networking,” in *International Conference on Transparent Optical Networks*, Cartagena, 2013.

[22] M. Pretorius, M. Ghassemian and C. Ierotheou, “An Investigation Into Energy Efficiency Of Data Centre Virtualization,” in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Fukuoka, 2010.

[23] A. Hameed and A. N. Mian, “Finding efficient VLAN topology for better broadcast containment,” in *The 2012 Third International Conference on the Network of the Future*, Tunis, 2012.

[24] AT&T: Margaret Chiosi, Steve Wright; BT: Don Clarke, Peter Willis, Andy Reid., “ETSI NFV White paper,” 17 October 2014. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper3.pdf . [Accessed 08 September 2015].

[25] W. Shen, M. Yoshida, T. Kawabata, K. Minato and W. Imajuku, “vConductor: An NFV Management Solution for Realizing End-to-End Virtual Network Services,” in *The Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Hsinchu, 2014.

[26] Z. Bronstein and E. Shraga, “NFV Virtualisation of the Home Environment,” in *IEEE Consumer Communications and Networking Conference (CCNC 2014) Special Seesion: Network Function Virtualization*, Las-Vegas, 2014.

[27] Cisco, “Quality of Service (QoS),” Cisco, [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html. [Accessed 09 November 2015].

[28] Cisco, “QoS on the Cisco ASA Configuration Examples,” Cisco, 19 December 2014. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/security/asa-5500-x-series-next-generation-firewalls/82310-qos-voip-vpn.pdf. [Accessed 09 11 2015].

[29] C. Lewis and S. Pickavance, Selecting MPLS VPN Service, Indianapolis: Cisco Press, 2006.

[30] TechTarget, “What is Data center?,” [Online]. Available: http://searchdatacenter.techtarget.com/definition/data-center. [Accessed 4 May 2016].

[31] Palo Alto Networks, “What is a data centre?,” [Online]. Available: https://www.paloaltonetworks.com/documentation/glossary/what-is-a-data-center. [Accessed 4 May 2016].

[32] Interxion, “What is a Data Centre?,” [Online]. Available: http://www.interxion.com/data-centres/. [Accessed 4 May 2016].

[33] P. Göransson and C. Black, Software Defined Networks: A Comprehensive Approach, Waltham: Morgan Kaufmann, 2014.

[34] H. Khosravi and T. Anderson, “ Requirements for Separation of IP Control and Forwarding,” The Internet Society, 2003.

[35] Stanford University, “Clean Slate Design for the Internet,” [Online]. Available: http://cleanslate.stanford.edu/research_project_ethane.php. [Accessed 12 January 2015].

[36] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown and S. Shenker, “Ethane: Taking Control of the Enterprise,” in *SIGCOMM’07*, Kyoto, 2007.

[37] Z. Khattak, M. Awais and A. Iqbal, “Performance Evaluation of OpenDaylight SDN Controller,” in *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, Hsinchu, 2014.

[38] F. Wang, H. Wang, B. Lei and W. Ma, “A Research on High-Performance SDN Controller,” in *Cloud Computing and Big Data (CCBD), 2014 International Conference on*, Wuhan, 2014.

[39] S. Scott-Hayward, “Design and deployment of secure, robust, and resilient SDN controllers,” in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, London, 2015.

[40] M. Casado, “List of OpenFlow Software Projects,” Stanford University, [Online]. Available: http://yuba.stanford.edu/~casado/of-sw.html. [Accessed 25 January 2016].

[41] Trema, “Trema,” [Online]. Available: http://trema.github.io/trema/. [Accessed 25 January 2016].

[42] Project Floodlight, “Floodlight OpenFlow Controller - Project Floodlight,” Big Switch Networks, [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed 26 January 2016].

[43] Ryu, “Ryu SDN Framework,” [Online]. Available: http://osrg.github.io/ryu/. [Accessed 5 May 2015].

[44] Linux Foundation, “Lithium | Opendaylight,” Linux Foundation, [Online]. Available: https://www.opendaylight.org/lithium. [Accessed 25 January 2016].

[45] Flowvisor, “Home . opennetworkinglab/flowvisor wiki . GitHub,” GitHub, [Online]. Available: https://github.com/OPENNETWORKINGLAB/flowvisor/wiki. [Accessed 26 January 2016].

[46] RouteFlow, “GitHub - routeflow/RouteFlow:Virtual IP Routing Services over OpenFlow networks,” GitHub, [Online]. Available: https://github.com/routeflow/RouteFlow. [Accessed 25 January 2016].

[47] A. Lara, A. Kolasani and B. Ramamurthy, “Network Innovation using OpenFlow: A Survey,” *IEEE COMMUNICATIONS SURVEY & TUTORIALS,* vol. 16, no. 1, pp. 493-512, 2014.

[48] Open Networking Foundation, “Open Flow - Open Networking Foundation,” Open Networking Foundation, [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow. [Accessed 29 January 2016].

[49] SDxCentral, “Comprehensive List: SDN Switching & Routing Products,” SDxCentral, [Online]. Available: https://www.sdxcentral.com/comprehensive-list-hardware-switching-routing/. [Accessed 1 February 2016].

[50] Open vSwitch, “GitHub - openvswitch/ovs: Open vSwitch,” GitHub, [Online]. Available: https://github.com/openvswitch/ovs. [Accessed 2 February 2016].

[51] Open vSwitch, “Open vSwitch,” Open vSwitch, [Online]. Available: http://openvswitch.org/. [Accessed 2 February 2016].

[52] Project Floodlight, “Indigo Virtual Switch - Open Source OpenFlowProject Floodlight,” Big Switch, [Online]. Available: http://www.projectfloodlight.org/indigo-virtual-switch/. [Accessed 02 February 2016].

[53] Ericsson Innovation Center, “GitHub - CPqD/ofsoftswitch13: OpenFlow 1.3 switch,” GitHub, [Online]. Available: https://github.com/CPqD/ofsoftswitch13/. [Accessed 2 February 2016].

[54] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69-74, March 2008.

[55] Open Networking Foundation, “openflow-switch-v1.3.4.pdf,” 27 March 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf. [Accessed 30 March 2015].

[56] F. Hu, Network Innovation through OpenFlow and SDN: Principles and Design, F. Hu, Ed., CRC Press, 2014.

[57] Mininet Team, “Introduction to Mininet . mininet/mininet Wiki . Mininet,” GitHub, [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. [Accessed 1 March 2016].

[58] Mininet Team, “Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet,” Mininet, [Online]. Available: http://mininet.org/. [Accessed 15 March 2015].

[59] R. Khondoker, A. Zaalouk, R. Marx and K. Bayarou, “Feature-based comparison and selection of Software Defined Networking (SDN) controllers,” in *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, Hammamet, 2014.

[60] Ryu, “Ryu Documentation Release 3.21,” [Online]. Available: http://osrg.github.io/ryu/. [Accessed 15 May 2015].

[61] iperf, “iPerf - The TCP, UDP and SCTP network bandwidth measurement tool,” ESnet / Lawrence Berkeley National Laboratory., [Online]. Available: https://iperf.fr/. [Accessed 7 March 2016].

[62] Netperf, “The Netperf Homepage,” [Online]. Available: http://netperf.org/netperf/. [Accessed 7 March 2016].

[63] about.com, “curl - Linux Command Unix Command,” [Online]. Available: http://linux.about.com/od/commands/l/blcmdl1_curl.htm. [Accessed 7 March 2016].

[64] Slashroot.in, “CURL command Tutorial in Linux with Example Usage,” [Online]. Available: http://www.slashroot.in/curl-command-tutorial-linux-example-usage. [Accessed 7 March 2016].

[65] U. Lamping, R. Sharpe and E. Warnicke, “Wireshark User Guide,” [Online]. Available: https://www.wireshark.org/download/docs/user-guide-a4.pdf. [Accessed 7 March 2016].

[66] Ryu, “Ryubook.pdf,” [Online]. Available: https://osrg.github.io/ryu-book/en/Ryubook.pdf. [Accessed 9 August 2017].

[67] Postdot Technologies, Inc, “Postman | Supercharge your API workflow,” [Online]. Available: https://www.getpostman.com/. [Accessed 28 November 2016].

[68] S. Morzhov, I. Alekseev and M. Nikitinskiy, “Firewall Application for Floodlight SDN controller,” in *2016 International Siberian Conference on Control and Communications*, Moscow, 2016.

[69] R. Nivedhitha, S. Abirami, K. R. Bala and N. R. Raajan, “Proficient Toning Mechanism for Firewall Policy Assessment,” in *2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, Tamil Nadu, 2015.

[70] F. Yan, Y. Jian-wen and L. Cheng, “Computer Network Security and technology Research,” in *2015 Seventh International conference on Measuring Technology and Mechatronics Automation*, Nanchang, 2015.

[71] S. Dhaval and R. D. Raviya, “Analysis of Software Defined Network Firewall (SDF),” in *IEEE International Conference on Wireless Communication, Signal Processing and Networking (WiSPNET)*, Chennai, 2016.

[72] S. T. Yakasai and C. G. Guy, “FlowIdentity: Software-Defined Network Access Control,” in *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, San Francisco, 2015.

[73] Lifewire, “Introduction to Latency on Computer Networks,” [Online]. Available: https://www.lifewire.com/latency-on-computer-networks-818119. [Accessed 7 February 2017].

[74] Esnet, “iperf3 - iperf3 3.16 documentation,” Energy Sciences Network, [Online]. Available: http://software.es.net/iperf/. [Accessed 31 January 2017].

[75] M. Devera, “HTB,” Iartc.org, [Online]. Available: http://lartc.org/manpages/tc-htb.html. [Accessed 11 April 2017].

[76] OpenvSwitch, “"Quality of Service (QoS) - Open vSwitch 2.7.90 documentation,” The Linux Foundation, [Online]. Available: http://docs.openvswitch.org/en/latest/faq/qos/. [Accessed 20 April 2017].

[77] T. Szigeti, C. Hattingh, R. Barton and K. Brilev, “End-to-End QoS Network Design,” in *Cisco Press*, Indianapolis, 2014.

[78] sFlow.org, “sFlowOverview.pdf,” [Online]. Available: http://www.sflow.org/sFlowOverview.pdf. [Accessed 20 April 2017].

[79] VideoLan, “Official Download of VLC media player, the best Open Source player - VideoLan,” VideoLan, [Online]. Available: http://www.videolan.org/vlc/index.html. [Accessed 25 April 2017].

[80] L. Zhao, J. Hua, X. Ge and S. Zhong, “Traffic Engineering in Hierarchical SDN Control Plane,” in *IEEE 23rd International Symposium on Quality of Service (IWQoS)*, Portland, 2015.

[81] I. F. Akyildiz, A. Lee, P. Wang, M. Luo and W. Chou, “Research Challenges for Traffic Engineering in Software Defined Networks,” *IEEE Network*, vol. 30, no. 3, pp. 52-58, 2016.

[82] SDN Hub, “SDN starter kit based on Ryu controller platform | SDN Hub,” [Online]. Available: http://sdnhub.org/releases/sdn-starter-kit-ryu/. [Accessed 10 August 2017].

[83] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho and C. Yang, “traffic engineering in software defined networking measurements and management,” *IEEE Access*, pp. 3246-3256, 21 June 2016.

[84] Y.-Y. Yang, W.-H. Cheng, C.-T. Yang, S.-T. Chen and F.-C. Jiang, “The Implementation of Real-Time Network Traffic Monitoring Service with Network Functions Virtualization,” in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing (BDCloud)*, Dalian, 2015.

[85] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet and V. Conan, “StateSec: Stateful Monitoring for DDoS Protection in Software Defined Networks,” in *Network Softwarization (NetSoft), 2017 IEEE Conference on*, Bologna, 2017.

[86] J. Suh, T. Kwon, C. Dixon, W. Felter and J. Carter , “OpenSample: A Low-latency, Sampling-based Measurement Platform for Commodity SDN,” in *2014 IEEE 34th International Conference on Distributed Computing Systems*, Madrid, 2014.

[87] S. U. Rehman, W.-C. Song and M. Kang, “Network-Wide Traffic Visibility in OF@TEIN SDN Testbed using sFlow,” in *2014 16th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Hsinchu, 2014.

[88] L. Huang, X. Zhi, Q. Gao, S. Kausar and S. Zheng, “Design and Implementation of Multicast Routing System over SDN and sFlow,” in *ICCSN-IEEE 2016 : 2016 8th International Conference on Communication Software and Networks (ICCSN 2016)*, Beijing, 2016.